

## Hardware Locality (hwloc)

1.1.1

Generated by Doxygen 1.7.2

Mon Jan 24 2011 17:12:06



# Contents

<b>1</b>	<b>Hardware Locality</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Installation . . . . .	2
1.3	CLI Examples . . . . .	3
1.4	Programming Interface . . . . .	8
1.4.1	Portability . . . . .	9
1.4.2	API Example . . . . .	12
1.5	Questions and Bugs . . . . .	15
1.6	History / Credits . . . . .	16
1.7	Further Reading . . . . .	16
<b>2</b>	<b>Terms and Definitions</b>	<b>17</b>
<b>3</b>	<b>Command-Line Tools</b>	<b>21</b>
3.1	lstopo . . . . .	21
3.2	hwloc-bind . . . . .	21
3.3	hwloc-calc . . . . .	21
3.4	hwloc-distrib . . . . .	22
3.5	hwloc-ps . . . . .	22
<b>4</b>	<b>Environment Variables</b>	<b>23</b>
<b>5</b>	<b>CPU and Memory Binding Overview</b>	<b>25</b>
<b>6</b>	<b>Interoperability With Other Software</b>	<b>27</b>
<b>7</b>	<b>Thread Safety</b>	<b>29</b>
<b>8</b>	<b>Embedding hwloc in Other Software</b>	<b>31</b>
8.1	Using hwloc's M4 Embedding Capabilities . . . . .	31
8.2	Example Embedding hwloc . . . . .	33
<b>9</b>	<b>Switching from PLPA to hwloc</b>	<b>35</b>
9.1	Topology Context vs. Caching . . . . .	35
9.2	Hierarchy vs. Core@Socket . . . . .	35
9.3	Logical vs. Physical/OS Indexes . . . . .	36
9.4	Counting Specification . . . . .	36
<b>10</b>	<b>Frequently Asked Questions</b>	<b>37</b>

10.1 I do not want hwloc to rediscover my enormous machine topology every time I rerun a process . . . . .	37
10.2 How do I handle API upgrades? . . . . .	37
<b>11 Module Index</b> . . . . .	<b>39</b>
11.1 Modules . . . . .	39
<b>12 Data Structure Index</b> . . . . .	<b>41</b>
12.1 Data Structures . . . . .	41
<b>13 Module Documentation</b> . . . . .	<b>43</b>
13.1 API version . . . . .	43
13.1.1 Define Documentation . . . . .	43
13.1.1.1 HWLOC_API_VERSION . . . . .	43
13.1.2 Function Documentation . . . . .	43
13.1.2.1 hwloc_get_api_version . . . . .	43
13.2 Topology context . . . . .	43
13.2.1 Typedef Documentation . . . . .	44
13.2.1.1 hwloc_topology_t . . . . .	44
13.3 Object sets (hwloc_cpuset_t and hwloc_nodeset_t) . . . . .	44
13.3.1 Detailed Description . . . . .	44
13.3.2 Typedef Documentation . . . . .	44
13.3.2.1 hwloc_const_cpuset_t . . . . .	44
13.3.2.2 hwloc_const_nodeset_t . . . . .	44
13.3.2.3 hwloc_cpuset_t . . . . .	44
13.3.2.4 hwloc_nodeset_t . . . . .	45
13.4 Topology Object Types . . . . .	45
13.4.1 Enumeration Type Documentation . . . . .	45
13.4.1.1 hwloc_compare_types_e . . . . .	45
13.4.1.2 hwloc_obj_type_t . . . . .	45
13.4.2 Function Documentation . . . . .	46
13.4.2.1 hwloc_compare_types . . . . .	46
13.5 Topology Objects . . . . .	47
13.5.1 Typedef Documentation . . . . .	47
13.5.1.1 hwloc_obj_t . . . . .	47
13.6 Create and Destroy Topologies . . . . .	47
13.6.1 Function Documentation . . . . .	47
13.6.1.1 hwloc_topology_check . . . . .	47
13.6.1.2 hwloc_topology_destroy . . . . .	48
13.6.1.3 hwloc_topology_init . . . . .	48
13.6.1.4 hwloc_topology_load . . . . .	48
13.7 Configure Topology Detection . . . . .	48
13.7.1 Detailed Description . . . . .	49
13.7.2 Enumeration Type Documentation . . . . .	50
13.7.2.1 hwloc_topology_flags_e . . . . .	50
13.7.3 Function Documentation . . . . .	50
13.7.3.1 hwloc_topology_get_support . . . . .	50
13.7.3.2 hwloc_topology_ignore_all_keep_structure . . . . .	50
13.7.3.3 hwloc_topology_ignore_type . . . . .	51
13.7.3.4 hwloc_topology_ignore_type_keep_structure . . . . .	51

13.7.3.5	<a href="#">hwloc_topology_set_flags</a>	51
13.7.3.6	<a href="#">hwloc_topology_set_fsroot</a>	51
13.7.3.7	<a href="#">hwloc_topology_set_pid</a>	51
13.7.3.8	<a href="#">hwloc_topology_set_synthetic</a>	52
13.7.3.9	<a href="#">hwloc_topology_set_xml</a>	52
13.7.3.10	<a href="#">hwloc_topology_set_xmlbuffer</a>	52
13.8	<a href="#">Tinker with topologies</a>	53
13.8.1	<a href="#">Function Documentation</a>	53
13.8.1.1	<a href="#">hwloc_topology_export_xml</a>	53
13.8.1.2	<a href="#">hwloc_topology_export_xmlbuffer</a>	53
13.8.1.3	<a href="#">hwloc_topology_insert_misc_object_by_cpuset</a>	53
13.8.1.4	<a href="#">hwloc_topology_insert_misc_object_by_parent</a>	53
13.9	<a href="#">Get some Topology Information</a>	54
13.9.1	<a href="#">Enumeration Type Documentation</a>	54
13.9.1.1	<a href="#">hwloc_get_type_depth_e</a>	54
13.9.2	<a href="#">Function Documentation</a>	54
13.9.2.1	<a href="#">hwloc_get_depth_type</a>	54
13.9.2.2	<a href="#">hwloc_get_nbobjs_by_depth</a>	55
13.9.2.3	<a href="#">hwloc_get_nbobjs_by_type</a>	55
13.9.2.4	<a href="#">hwloc_get_type_depth</a>	55
13.9.2.5	<a href="#">hwloc_topology_get_depth</a>	55
13.9.2.6	<a href="#">hwloc_topology_is_thissystem</a>	55
13.10	<a href="#">Retrieve Objects</a>	55
13.10.1	<a href="#">Function Documentation</a>	56
13.10.1.1	<a href="#">hwloc_get_obj_by_depth</a>	56
13.10.1.2	<a href="#">hwloc_get_obj_by_type</a>	56
13.11	<a href="#">Object/String Conversion</a>	56
13.11.1	<a href="#">Function Documentation</a>	56
13.11.1.1	<a href="#">hwloc_obj_attr_snprintf</a>	56
13.11.1.2	<a href="#">hwloc_obj_cpuset_snprintf</a>	57
13.11.1.3	<a href="#">hwloc_obj_get_info_by_name</a>	57
13.11.1.4	<a href="#">hwloc_obj_snprintf</a>	57
13.11.1.5	<a href="#">hwloc_obj_type_of_string</a>	58
13.11.1.6	<a href="#">hwloc_obj_type_snprintf</a>	58
13.11.1.7	<a href="#">hwloc_obj_type_string</a>	58
13.12	<a href="#">CPU binding</a>	58
13.12.1	<a href="#">Detailed Description</a>	59
13.12.2	<a href="#">Enumeration Type Documentation</a>	59
13.12.2.1	<a href="#">hwloc_cpubind_flags_t</a>	59
13.12.3	<a href="#">Function Documentation</a>	60
13.12.3.1	<a href="#">hwloc_get_cpubind</a>	60
13.12.3.2	<a href="#">hwloc_get_proc_cpubind</a>	60
13.12.3.3	<a href="#">hwloc_get_thread_cpubind</a>	61
13.12.3.4	<a href="#">hwloc_set_cpubind</a>	61
13.12.3.5	<a href="#">hwloc_set_proc_cpubind</a>	61
13.12.3.6	<a href="#">hwloc_set_thread_cpubind</a>	61
13.13	<a href="#">Memory binding</a>	62
13.13.1	<a href="#">Detailed Description</a>	63
13.13.2	<a href="#">Enumeration Type Documentation</a>	63
13.13.2.1	<a href="#">hwloc_membind_flags_t</a>	63

13.13.2.2	<a href="#">hwloc_membind_policy_t</a>	64
13.13.3	<a href="#">Function Documentation</a>	65
13.13.3.1	<a href="#">hwloc_alloc</a>	65
13.13.3.2	<a href="#">hwloc_alloc_membind</a>	65
13.13.3.3	<a href="#">hwloc_alloc_membind_nodeset</a>	66
13.13.3.4	<a href="#">hwloc_free</a>	66
13.13.3.5	<a href="#">hwloc_get_area_membind</a>	66
13.13.3.6	<a href="#">hwloc_get_area_membind_nodeset</a>	67
13.13.3.7	<a href="#">hwloc_get_membind</a>	67
13.13.3.8	<a href="#">hwloc_get_membind_nodeset</a>	68
13.13.3.9	<a href="#">hwloc_get_proc_membind</a>	68
13.13.3.10	<a href="#">hwloc_get_proc_membind_nodeset</a>	69
13.13.3.11	<a href="#">hwloc_set_area_membind</a>	70
13.13.3.12	<a href="#">hwloc_set_area_membind_nodeset</a>	70
13.13.3.13	<a href="#">hwloc_set_membind</a>	70
13.13.3.14	<a href="#">hwloc_set_membind_nodeset</a>	70
13.13.3.15	<a href="#">hwloc_set_proc_membind</a>	71
13.13.3.16	<a href="#">hwloc_set_proc_membind_nodeset</a>	71
13.14	<a href="#">Object Type Helpers</a>	71
13.14.1	<a href="#">Function Documentation</a>	72
13.14.1.1	<a href="#">hwloc_get_type_or_above_depth</a>	72
13.14.1.2	<a href="#">hwloc_get_type_or_below_depth</a>	72
13.15	<a href="#">Basic Traversal Helpers</a>	72
13.15.1	<a href="#">Function Documentation</a>	72
13.15.1.1	<a href="#">hwloc_get_ancestor_obj_by_depth</a>	72
13.15.1.2	<a href="#">hwloc_get_ancestor_obj_by_type</a>	73
13.15.1.3	<a href="#">hwloc_get_common_ancestor_obj</a>	73
13.15.1.4	<a href="#">hwloc_get_next_child</a>	73
13.15.1.5	<a href="#">hwloc_get_next_obj_by_depth</a>	73
13.15.1.6	<a href="#">hwloc_get_next_obj_by_type</a>	73
13.15.1.7	<a href="#">hwloc_get_pu_obj_by_os_index</a>	73
13.15.1.8	<a href="#">hwloc_get_root_obj</a>	74
13.15.1.9	<a href="#">hwloc_obj_is_in_subtree</a>	74
13.16	<a href="#">Finding Objects Inside a CPU set</a>	74
13.16.1	<a href="#">Function Documentation</a>	74
13.16.1.1	<a href="#">hwloc_get_first_largest_obj_inside_cpuset</a>	74
13.16.1.2	<a href="#">hwloc_get_largest_objs_inside_cpuset</a>	75
13.16.1.3	<a href="#">hwloc_get_nbobjs_inside_cpuset_by_depth</a>	75
13.16.1.4	<a href="#">hwloc_get_nbobjs_inside_cpuset_by_type</a>	75
13.16.1.5	<a href="#">hwloc_get_next_obj_inside_cpuset_by_depth</a>	75
13.16.1.6	<a href="#">hwloc_get_next_obj_inside_cpuset_by_type</a>	75
13.16.1.7	<a href="#">hwloc_get_obj_inside_cpuset_by_depth</a>	76
13.16.1.8	<a href="#">hwloc_get_obj_inside_cpuset_by_type</a>	76
13.17	<a href="#">Finding a single Object covering at least CPU set</a>	76
13.17.1	<a href="#">Function Documentation</a>	76
13.17.1.1	<a href="#">hwloc_get_child_covering_cpuset</a>	76
13.17.1.2	<a href="#">hwloc_get_obj_covering_cpuset</a>	76
13.18	<a href="#">Finding a set of similar Objects covering at least a CPU set</a>	77
13.18.1	<a href="#">Function Documentation</a>	77
13.18.1.1	<a href="#">hwloc_get_next_obj_covering_cpuset_by_depth</a>	77

13.18.1.2	<a href="#">hwloc_get_next_obj_covering_cpuset_by_type</a>	77
13.19	Cache-specific Finding Helpers	77
13.19.1	Function Documentation	78
13.19.1.1	<a href="#">hwloc_get_cache_covering_cpuset</a>	78
13.19.1.2	<a href="#">hwloc_get_shared_cache_covering_obj</a>	78
13.20	Advanced Traversal Helpers	78
13.20.1	Function Documentation	78
13.20.1.1	<a href="#">hwloc_get_closest_objs</a>	78
13.20.1.2	<a href="#">hwloc_get_obj_below_array_by_type</a>	79
13.20.1.3	<a href="#">hwloc_get_obj_below_by_type</a>	79
13.21	Binding Helpers	79
13.21.1	Function Documentation	80
13.21.1.1	<a href="#">hwloc_alloc_mbind_policy</a>	80
13.21.1.2	<a href="#">hwloc_alloc_mbind_policy_nodeset</a>	80
13.21.1.3	<a href="#">hwloc_distribute</a>	80
13.21.1.4	<a href="#">hwloc_distributev</a>	80
13.22	Cpuset Helpers	81
13.22.1	Function Documentation	81
13.22.1.1	<a href="#">hwloc_topology_get_allowed_cpuset</a>	81
13.22.1.2	<a href="#">hwloc_topology_get_complete_cpuset</a>	81
13.22.1.3	<a href="#">hwloc_topology_get_online_cpuset</a>	82
13.22.1.4	<a href="#">hwloc_topology_get_topology_cpuset</a>	82
13.23	Nodeset Helpers	82
13.23.1	Function Documentation	82
13.23.1.1	<a href="#">hwloc_topology_get_allowed_nodeset</a>	82
13.23.1.2	<a href="#">hwloc_topology_get_complete_nodeset</a>	83
13.23.1.3	<a href="#">hwloc_topology_get_topology_nodeset</a>	83
13.24	Conversion between cpuset and nodeset	83
13.24.1	Detailed Description	84
13.24.2	Function Documentation	84
13.24.2.1	<a href="#">hwloc_cpuset_from_nodeset</a>	84
13.24.2.2	<a href="#">hwloc_cpuset_from_nodeset_strict</a>	84
13.24.2.3	<a href="#">hwloc_cpuset_to_nodeset</a>	84
13.24.2.4	<a href="#">hwloc_cpuset_to_nodeset_strict</a>	85
13.25	The bitmap API	85
13.25.1	Detailed Description	87
13.25.2	Define Documentation	87
13.25.2.1	<a href="#">hwloc_bitmap_foreach_begin</a>	87
13.25.2.2	<a href="#">hwloc_bitmap_foreach_end</a>	87
13.25.3	Typedef Documentation	87
13.25.3.1	<a href="#">hwloc_bitmap_t</a>	87
13.25.3.2	<a href="#">hwloc_const_bitmap_t</a>	87
13.25.4	Function Documentation	88
13.25.4.1	<a href="#">hwloc_bitmap_allbut</a>	88
13.25.4.2	<a href="#">hwloc_bitmap_alloc</a>	88
13.25.4.3	<a href="#">hwloc_bitmap_alloc_full</a>	88
13.25.4.4	<a href="#">hwloc_bitmap_and</a>	88
13.25.4.5	<a href="#">hwloc_bitmap_andnot</a>	88
13.25.4.6	<a href="#">hwloc_bitmap_asprintf</a>	88
13.25.4.7	<a href="#">hwloc_bitmap_clr</a>	88

13.25.4.8	<code>hwloc_bitmap_clr_range</code>	88
13.25.4.9	<code>hwloc_bitmap_compare</code>	89
13.25.4.10	<code>hwloc_bitmap_compare_first</code>	89
13.25.4.11	<code>hwloc_bitmap_copy</code>	89
13.25.4.12	<code>hwloc_bitmap_dup</code>	89
13.25.4.13	<code>hwloc_bitmap_fill</code>	89
13.25.4.14	<code>hwloc_bitmap_first</code>	89
13.25.4.15	<code>hwloc_bitmap_free</code>	89
13.25.4.16	<code>hwloc_bitmap_from_lth_ulong</code>	89
13.25.4.17	<code>hwloc_bitmap_from_ulong</code>	90
13.25.4.18	<code>hwloc_bitmap_intersects</code>	90
13.25.4.19	<code>hwloc_bitmap_isequal</code>	90
13.25.4.20	<code>hwloc_bitmap_isfull</code>	90
13.25.4.21	<code>hwloc_bitmap_isincluded</code>	90
13.25.4.22	<code>hwloc_bitmap_isset</code>	90
13.25.4.23	<code>hwloc_bitmap_iszero</code>	90
13.25.4.24	<code>hwloc_bitmap_last</code>	90
13.25.4.25	<code>hwloc_bitmap_next</code>	90
13.25.4.26	<code>hwloc_bitmap_not</code>	91
13.25.4.27	<code>hwloc_bitmap_only</code>	91
13.25.4.28	<code>hwloc_bitmap_or</code>	91
13.25.4.29	<code>hwloc_bitmap_set</code>	91
13.25.4.30	<code>hwloc_bitmap_set_lth_ulong</code>	91
13.25.4.31	<code>hwloc_bitmap_set_range</code>	91
13.25.4.32	<code>hwloc_bitmap_singlify</code>	91
13.25.4.33	<code>hwloc_bitmap_snprintf</code>	91
13.25.4.34	<code>hwloc_bitmap_sscanf</code>	92
13.25.4.35	<code>hwloc_bitmap_taskset_asprintf</code>	92
13.25.4.36	<code>hwloc_bitmap_taskset_snprintf</code>	92
13.25.4.37	<code>hwloc_bitmap_taskset_sscanf</code>	92
13.25.4.38	<code>hwloc_bitmap_to_lth_ulong</code>	92
13.25.4.39	<code>hwloc_bitmap_to_ulong</code>	92
13.25.4.40	<code>hwloc_bitmap_weight</code>	93
13.25.4.41	<code>hwloc_bitmap_xor</code>	93
13.25.4.42	<code>hwloc_bitmap_zero</code>	93
13.26	Helpers for manipulating glibc sched affinity	93
13.26.1	Function Documentation	93
13.26.1.1	<code>hwloc_cpuset_from_glibc_sched_affinity</code>	93
13.26.1.2	<code>hwloc_cpuset_to_glibc_sched_affinity</code>	93
13.27	Linux-only helpers	94
13.27.1	Detailed Description	94
13.27.2	Function Documentation	94
13.27.2.1	<code>hwloc_linux_get_tid_cpupbind</code>	94
13.27.2.2	<code>hwloc_linux_parse_cpumap_file</code>	94
13.27.2.3	<code>hwloc_linux_set_tid_cpupbind</code>	94
13.28	Helpers for manipulating Linux libnuma unsigned long masks	95
13.28.1	Function Documentation	95
13.28.1.1	<code>hwloc_cpuset_from_linux_libnuma_ulong</code>	95
13.28.1.2	<code>hwloc_cpuset_to_linux_libnuma_ulong</code>	95
13.28.1.3	<code>hwloc_nodeset_from_linux_libnuma_ulong</code>	95



13.28.1.4	<a href="#">hwloc_nodeseq_to_linux_libnuma_ulong</a>	96
13.29	<a href="#">Helpers for manipulating Linux libnuma bitmask</a>	96
13.29.1	<a href="#">Function Documentation</a>	96
13.29.1.1	<a href="#">hwloc_cpuset_from_linux_libnuma_bitmask</a>	96
13.29.1.2	<a href="#">hwloc_cpuset_to_linux_libnuma_bitmask</a>	97
13.29.1.3	<a href="#">hwloc_nodeseq_from_linux_libnuma_bitmask</a>	97
13.29.1.4	<a href="#">hwloc_nodeseq_to_linux_libnuma_bitmask</a>	97
13.30	<a href="#">Helpers for manipulating Linux libnuma nodemask_t</a>	97
13.30.1	<a href="#">Function Documentation</a>	98
13.30.1.1	<a href="#">hwloc_cpuset_from_linux_libnuma_nodemask</a>	98
13.30.1.2	<a href="#">hwloc_cpuset_to_linux_libnuma_nodemask</a>	98
13.30.1.3	<a href="#">hwloc_nodeseq_from_linux_libnuma_nodemask</a>	98
13.30.1.4	<a href="#">hwloc_nodeseq_to_linux_libnuma_nodemask</a>	98
13.31	<a href="#">CUDA Driver API Specific Functions</a>	98
13.31.1	<a href="#">Function Documentation</a>	99
13.31.1.1	<a href="#">hwloc_cuda_get_device_cpuset</a>	99
13.32	<a href="#">CUDA Runtime API Specific Functions</a>	99
13.32.1	<a href="#">Function Documentation</a>	99
13.32.1.1	<a href="#">hwloc_cuda_get_device_cpuset</a>	99
13.33	<a href="#">OpenFabrics-Specific Functions</a>	99
13.33.1	<a href="#">Function Documentation</a>	99
13.33.1.1	<a href="#">hwloc_ibv_get_device_cpuset</a>	99
13.34	<a href="#">Myrinet Express-Specific Functions</a>	100
13.34.1	<a href="#">Function Documentation</a>	100
13.34.1.1	<a href="#">hwloc_mx_board_get_device_cpuset</a>	100
13.34.1.2	<a href="#">hwloc_mx_endpoint_get_device_cpuset</a>	100
<b>14</b>	<b><a href="#">Data Structure Documentation</a></b>	<b>101</b>
14.1	<a href="#">hwloc_obj_attr_u::hwloc_cache_attr_s Struct Reference</a>	101
14.1.1	<a href="#">Detailed Description</a>	101
14.1.2	<a href="#">Field Documentation</a>	101
14.1.2.1	<a href="#">depth</a>	101
14.1.2.2	<a href="#">linesize</a>	101
14.1.2.3	<a href="#">size</a>	101
14.2	<a href="#">hwloc_obj_attr_u::hwloc_group_attr_s Struct Reference</a>	102
14.2.1	<a href="#">Detailed Description</a>	102
14.2.2	<a href="#">Field Documentation</a>	102
14.2.2.1	<a href="#">depth</a>	102
14.3	<a href="#">hwloc_obj Struct Reference</a>	102
14.3.1	<a href="#">Detailed Description</a>	103
14.3.2	<a href="#">Field Documentation</a>	103
14.3.2.1	<a href="#">allowed_cpuset</a>	103
14.3.2.2	<a href="#">allowed_nodeseq</a>	104
14.3.2.3	<a href="#">arity</a>	104
14.3.2.4	<a href="#">attr</a>	104
14.3.2.5	<a href="#">children</a>	104
14.3.2.6	<a href="#">complete_cpuset</a>	104
14.3.2.7	<a href="#">complete_nodeseq</a>	104
14.3.2.8	<a href="#">cpuset</a>	105
14.3.2.9	<a href="#">depth</a>	105

14.3.2.10	first_child	105
14.3.2.11	infos	105
14.3.2.12	infos_count	105
14.3.2.13	last_child	105
14.3.2.14	logical_index	106
14.3.2.15	memory	106
14.3.2.16	name	106
14.3.2.17	next_cousin	106
14.3.2.18	next_sibling	106
14.3.2.19	nodeset	106
14.3.2.20	online_cpuset	106
14.3.2.21	os_index	107
14.3.2.22	os_level	107
14.3.2.23	parent	107
14.3.2.24	prev_cousin	107
14.3.2.25	prev_sibling	107
14.3.2.26	sibling_rank	107
14.3.2.27	type	107
14.3.2.28	userdata	107
14.4	hwloc_obj_attr_u Union Reference	108
14.4.1	Detailed Description	108
14.4.2	Field Documentation	108
14.4.2.1	cache	108
14.4.2.2	group	108
14.5	hwloc_obj_info_s Struct Reference	108
14.5.1	Detailed Description	109
14.5.2	Field Documentation	109
14.5.2.1	name	109
14.5.2.2	value	109
14.6	hwloc_obj_memory_s::hwloc_obj_memory_page_type_s Struct Reference	109
14.6.1	Detailed Description	109
14.6.2	Field Documentation	109
14.6.2.1	count	109
14.6.2.2	size	110
14.7	hwloc_obj_memory_s Struct Reference	110
14.7.1	Detailed Description	110
14.7.2	Field Documentation	110
14.7.2.1	local_memory	110
14.7.2.2	page_types	110
14.7.2.3	page_types_len	110
14.7.2.4	total_memory	111
14.8	hwloc_topology_cpupbind_support Struct Reference	111
14.8.1	Detailed Description	111
14.8.2	Field Documentation	111
14.8.2.1	get_proc_cpupbind	111
14.8.2.2	get_thisproc_cpupbind	111
14.8.2.3	get_thisthread_cpupbind	111
14.8.2.4	get_thread_cpupbind	112
14.8.2.5	set_proc_cpupbind	112
14.8.2.6	set_thisproc_cpupbind	112

---

14.8.2.7	set_thisthread_cpupbind	112
14.8.2.8	set_thread_cpupbind	112
14.9	hwloc_topology_discovery_support Struct Reference	112
14.9.1	Detailed Description	112
14.9.2	Field Documentation	112
14.9.2.1	pu	112
14.10	hwloc_topology_membind_support Struct Reference	113
14.10.1	Detailed Description	113
14.10.2	Field Documentation	113
14.10.2.1	alloc_membind	113
14.10.2.2	bind_membind	113
14.10.2.3	firsttouch_membind	113
14.10.2.4	get_area_membind	114
14.10.2.5	get_proc_membind	114
14.10.2.6	get_thisproc_membind	114
14.10.2.7	get_thisthread_membind	114
14.10.2.8	interleave_membind	114
14.10.2.9	migrate_membind	114
14.10.2.10	nexttouch_membind	114
14.10.2.11	replicate_membind	114
14.10.2.12	set_area_membind	114
14.10.2.13	set_proc_membind	114
14.10.2.14	set_thisproc_membind	114
14.10.2.15	set_thisthread_membind	115
14.11	hwloc_topology_support Struct Reference	115
14.11.1	Detailed Description	115
14.11.2	Field Documentation	115
14.11.2.1	cpubind	115
14.11.2.2	discovery	115
14.11.2.3	membind	115



# Chapter 1

## Hardware Locality

**Portable abstraction of hierarchical architectures for high-performance computing**

### 1.1 Introduction

hwloc provides command line tools and a C API to obtain the hierarchical map of key computing elements, such as: NUMA memory nodes, shared caches, processor sockets, processor cores, and processing units (logical processors or "threads"). hwloc also gathers various attributes such as cache and memory information, and is portable across a variety of different operating systems and platforms.

hwloc primarily aims at helping high-performance computing (HPC) applications, but is also applicable to any project seeking to exploit code and/or data locality on modern computing platforms.

Note that the hwloc project represents the merger of the libtopology project from INRIA and the Portable Linux Processor Affinity (PLPA) sub-project from Open MPI. *Both of these prior projects are now deprecated.* The first hwloc release is essentially a "re-branding" of the libtopology code base, but with both a few genuinely new features and a few PLPA-like features added in. More new features and more PLPA-like features will be added to hwloc over time. See [Switching from PLPA to hwloc](#) for more details about converting your application from PLPA to hwloc.

hwloc supports the following operating systems:

- Linux (including old kernels not having sysfs topology information, with knowledge of cpusets, offline CPUs, ScaleMP vSMP, and Kerrighed support)
- Solaris
- AIX
- Darwin / OS X

- FreeBSD and its variants, such as kFreeBSD/GNU
- OSF/1 (a.k.a., Tru64)
- HP-UX
- Microsoft Windows

hwloc only reports the number of processors on unsupported operating systems; no topology information is available.

For development and debugging purposes, hwloc also offers the ability to work on "fake" topologies:

- Symmetrical tree of resources generated from a list of level arities
- Remote machine simulation through the gathering of Linux sysfs topology files

hwloc can display the topology in a human-readable format, either in graphical mode (X11), or by exporting in one of several different formats, including: plain text, PDF, PNG, and FIG (see [CLI Examples](#) below). Note that some of the export formats require additional support libraries.

hwloc offers a programming interface for manipulating topologies and objects. It also brings a powerful CPU bitmap API that is used to describe topology objects location on physical/logical processors. See the [Programming Interface](#) below. It may also be used to binding applications onto certain cores or memory nodes. Several utility programs are also provided to ease command-line manipulation of topology objects, binding of processes, and so on.

## 1.2 Installation

hwloc (<http://www.open-mpi.org/projects/hwloc/>) is available under the BSD license. It is hosted as a sub-project of the overall Open MPI project (<http://www.open-mpi.org>). Note that hwloc does not require any functionality from Open MPI -- it is a wholly separate (and much smaller!) project and code base. It just happens to be hosted as part of the overall Open MPI project.

Nightly development snapshots are available on the web site. Additionally, the code can be directly checked out of Subversion:

```
shell$ svn checkout http://svn.open-mpi.org/svn/hwloc/trunk hwloc-trunk
shell$ cd hwloc-trunk
shell$ ./autogen.sh
```

Note that GNU Autoconf  $\geq 2.63$ , Automake  $\geq 1.10$  and Libtool  $\geq 2.2.6$  are required when building from a Subversion checkout.

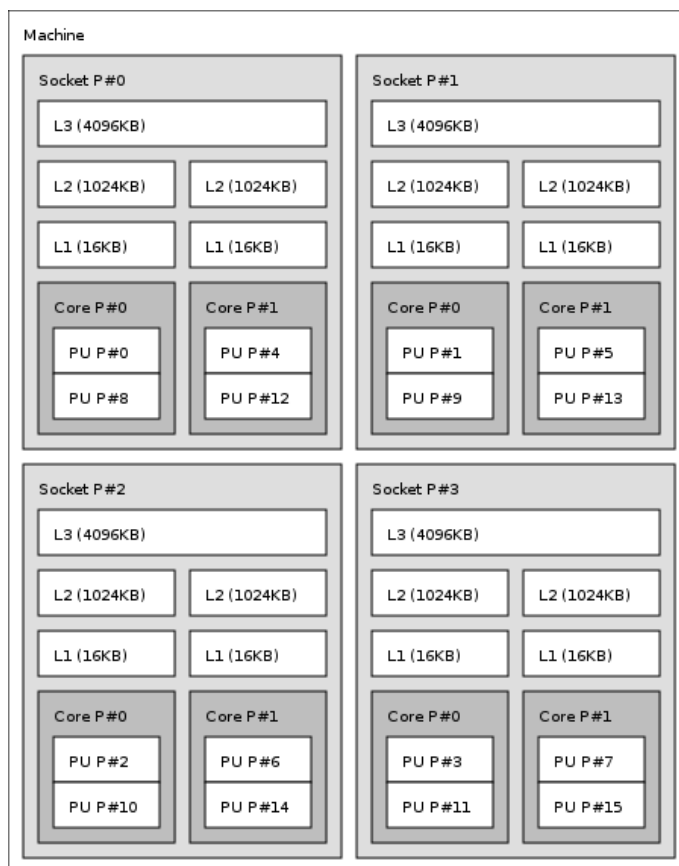
Installation by itself is the fairly common GNU-based process:

```
shell$ ./configure --prefix=...
shell$ make
shell$ make install
```

The hwloc command-line tool "lstopo" produces human-readable topology maps, as mentioned above. It can also export maps to the "fig" file format. Support for PDF, Postscript, and PNG exporting is provided if the "Cairo" development package can be found when hwloc is configured and build. Similarly, lstopo's XML support requires the libxml2 development package.

## 1.3 CLI Examples

On a 4-socket 2-core machine with hyperthreading, the lstopo tool may show the following graphical output:



Here's the equivalent output in textual form:

```
Machine (16GB)
  Socket L#0 + L3 L#0 (4096KB)
    L2 L#0 (1024KB) + L1 L#0 (16KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#8)
    L2 L#1 (1024KB) + L1 L#1 (16KB) + Core L#1
      PU L#2 (P#4)
      PU L#3 (P#12)
  Socket L#1 + L3 L#1 (4096KB)
    L2 L#2 (1024KB) + L1 L#2 (16KB) + Core L#2
```

```

    PU L#4 (P#1)
    PU L#5 (P#9)
    L2 L#3 (1024KB) + L1 L#3 (16KB) + Core L#3
    PU L#6 (P#5)
    PU L#7 (P#13)
    Socket L#2 + L3 L#2 (4096KB)
    L2 L#4 (1024KB) + L1 L#4 (16KB) + Core L#4
    PU L#8 (P#2)
    PU L#9 (P#10)
    L2 L#5 (1024KB) + L1 L#5 (16KB) + Core L#5
    PU L#10 (P#6)
    PU L#11 (P#14)
    Socket L#3 + L3 L#3 (4096KB)
    L2 L#6 (1024KB) + L1 L#6 (16KB) + Core L#6
    PU L#12 (P#3)
    PU L#13 (P#11)
    L2 L#7 (1024KB) + L1 L#7 (16KB) + Core L#7
    PU L#14 (P#7)
    PU L#15 (P#15)

```

Finally, here's the equivalent output in XML. Long lines were artificially broken for document clarity (in the real output, each XML tag is on a single line), and only socket #0 is shown for brevity:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x0000ffff"
    complete_cpuset="0x0000ffff" online_cpuset="0x0000ffff"
    allowed_cpuset="0x0000ffff"
    dmi_board_vendor="Dell Computer Corporation" dmi_board_name="0RD318"
    local_memory="16648183808">
    <page_type size="4096" count="4064498"/>
    <page_type size="2097152" count="0"/>
    <object type="Socket" os_level="-1" os_index="0" cpuset="0x00001111"
      complete_cpuset="0x00001111" online_cpuset="0x00001111"
      allowed_cpuset="0x00001111">
      <object type="Cache" os_level="-1" cpuset="0x00001111"
        complete_cpuset="0x00001111" online_cpuset="0x00001111"
        allowed_cpuset="0x00001111" cache_size="4194304" depth="3"
        cache_linesize="64">
        <object type="Cache" os_level="-1" cpuset="0x00000101"
          complete_cpuset="0x00000101" online_cpuset="0x00000101"
          allowed_cpuset="0x00000101" cache_size="1048576" depth="2"
          cache_linesize="64">
          <object type="Cache" os_level="-1" cpuset="0x00000101"
            complete_cpuset="0x00000101" online_cpuset="0x00000101"
            allowed_cpuset="0x00000101" cache_size="16384" depth="1"
            cache_linesize="64">
            <object type="Core" os_level="-1" os_index="0" cpuset="0x00000101"
              complete_cpuset="0x00000101" online_cpuset="0x00000101"
              allowed_cpuset="0x00000101">
              <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
                complete_cpuset="0x00000001" online_cpuset="0x00000001"
                allowed_cpuset="0x00000001"/>
              <object type="PU" os_level="-1" os_index="8" cpuset="0x00000100"
                complete_cpuset="0x00000100" online_cpuset="0x00000100"
                allowed_cpuset="0x00000100"/>
            </object>
          </object>
        </object>
      </object>
    </object>
  </object>

```

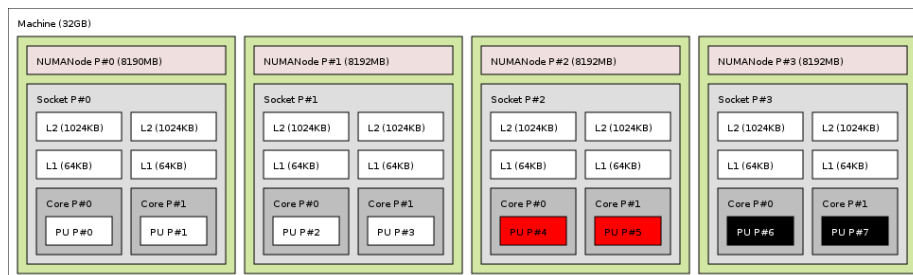


```

<object type="Cache" os_level="-1" cpuset="0x00001010"
  complete_cpuset="0x00001010" online_cpuset="0x00001010"
  allowed_cpuset="0x00001010" cache_size="1048576" depth="2"
  cache_linesize="64">
<object type="Cache" os_level="-1" cpuset="0x00001010"
  complete_cpuset="0x00001010" online_cpuset="0x00001010"
  allowed_cpuset="0x00001010" cache_size="16384" depth="1"
  cache_linesize="64">
<object type="Core" os_level="-1" os_index="1" cpuset="0x00001010"
  complete_cpuset="0x00001010" online_cpuset="0x00001010"
  allowed_cpuset="0x00001010">
  <object type="PU" os_level="-1" os_index="4" cpuset="0x00000010"
    complete_cpuset="0x00000010" online_cpuset="0x00000010"
    allowed_cpuset="0x00000010"/>
  <object type="PU" os_level="-1" os_index="12" cpuset="0x00001000"
    complete_cpuset="0x00001000" online_cpuset="0x00001000"
    allowed_cpuset="0x00001000"/>
  </object>
</object>
</object>
</object>
</object>
<!-- ...other sockets listed here ... -->
</object>
</topology>

```

On a 4-socket 2-core Opteron NUMA machine, the `lstopo` tool may show the following graphical output:



Here's the equivalent output in textual form:

```

Machine (32GB)
  NUMANode L#0 (P#0 8190MB) + Socket L#0
    L2 L#0 (1024KB) + L1 L#0 (64KB) + Core L#0 + PU L#0 (P#0)
    L2 L#1 (1024KB) + L1 L#1 (64KB) + Core L#1 + PU L#1 (P#1)
  NUMANode L#1 (P#1 8192MB) + Socket L#1
    L2 L#2 (1024KB) + L1 L#2 (64KB) + Core L#2 + PU L#2 (P#2)
    L2 L#3 (1024KB) + L1 L#3 (64KB) + Core L#3 + PU L#3 (P#3)
  NUMANode L#2 (P#2 8192MB) + Socket L#2
    L2 L#4 (1024KB) + L1 L#4 (64KB) + Core L#4 + PU L#4 (P#4)
    L2 L#5 (1024KB) + L1 L#5 (64KB) + Core L#5 + PU L#5 (P#5)
  NUMANode L#3 (P#3 8192MB) + Socket L#3
    L2 L#6 (1024KB) + L1 L#6 (64KB) + Core L#6 + PU L#6 (P#6)
    L2 L#7 (1024KB) + L1 L#7 (64KB) + Core L#7 + PU L#7 (P#7)

```

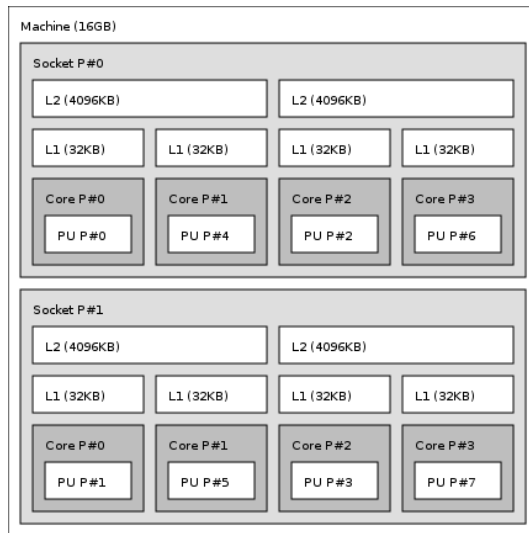
And here's the equivalent output in XML. Similar to above, line breaks were added and only PU #0 is shown for brevity:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x000000ff"
    complete_cpuset="0x000000ff" online_cpuset="0x000000ff"
    allowed_cpuset="0x000000ff" nodeset="0x000000ff"
    complete_nodeset="0x000000ff" allowed_nodeset="0x000000ff"
    dmi_board_vendor="TYAN Computer Corp" dmi_board_name="S4881 ">
    <page_type size="4096" count="0"/>
    <page_type size="2097152" count="0"/>
    <object type="NUMANode" os_level="-1" os_index="0" cpuset="0x00000003"
      complete_cpuset="0x00000003" online_cpuset="0x00000003"
      allowed_cpuset="0x00000003" nodeset="0x00000001"
      complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
      local_memory="7514177536">
      <page_type size="4096" count="1834516"/>
      <page_type size="2097152" count="0"/>
      <object type="Socket" os_level="-1" os_index="0" cpuset="0x00000003"
        complete_cpuset="0x00000003" online_cpuset="0x00000003"
        allowed_cpuset="0x00000003" nodeset="0x00000001"
        complete_nodeset="0x00000001" allowed_nodeset="0x00000001">
        <object type="Cache" os_level="-1" cpuset="0x00000001"
          complete_cpuset="0x00000001" online_cpuset="0x00000001"
          allowed_cpuset="0x00000001" nodeset="0x00000001"
          complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
          cache_size="1048576" depth="2" cache_linesize="64">
          <object type="Cache" os_level="-1" cpuset="0x00000001"
            complete_cpuset="0x00000001" online_cpuset="0x00000001"
            allowed_cpuset="0x00000001" nodeset="0x00000001"
            complete_nodeset="0x00000001" allowed_nodeset="0x00000001"
            cache_size="65536" depth="1" cache_linesize="64">
            <object type="Core" os_level="-1" os_index="0"
              cpuset="0x00000001" complete_cpuset="0x00000001"
              online_cpuset="0x00000001" allowed_cpuset="0x00000001"
              nodeset="0x00000001" complete_nodeset="0x00000001"
              allowed_nodeset="0x00000001">
              <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
                complete_cpuset="0x00000001" online_cpuset="0x00000001"
                allowed_cpuset="0x00000001" nodeset="0x00000001"
                complete_nodeset="0x00000001" allowed_nodeset="0x00000001"/>
            </object>
          </object>
        </object>
      </object>
    </object>
  <!-- ...more objects listed here ... -->
</topology>

```

On a 2-socket quad-core Xeon (pre-Nehalem, with 2 dual-core dies into each socket):



Here's the same output in textual form:

```
Machine (16GB)
Socket L#0
  L2 L#0 (4096KB)
    L1 L#0 (32KB) + Core L#0 + PU L#0 (P#0)
    L1 L#1 (32KB) + Core L#1 + PU L#1 (P#4)
  L2 L#1 (4096KB)
    L1 L#2 (32KB) + Core L#2 + PU L#2 (P#2)
    L1 L#3 (32KB) + Core L#3 + PU L#3 (P#6)
Socket L#1
  L2 L#2 (4096KB)
    L1 L#4 (32KB) + Core L#4 + PU L#4 (P#1)
    L1 L#5 (32KB) + Core L#5 + PU L#5 (P#5)
  L2 L#3 (4096KB)
    L1 L#6 (32KB) + Core L#6 + PU L#6 (P#3)
    L1 L#7 (32KB) + Core L#7 + PU L#7 (P#7)
```

And the same output in XML (line breaks added, only PU #0 shown):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topology SYSTEM "hwloc.dtd">
<topology>
  <object type="Machine" os_level="-1" os_index="0" cpuset="0x000000ff"
    complete_cpuset="0x000000ff" online_cpuset="0x000000ff"
    allowed_cpuset="0x000000ff" dmi_board_vendor="Dell Inc."
    dmi_board_name="0NR282" local_memory="16865292288">
    <page_type size="4096" count="4117503"/>
    <page_type size="2097152" count="0"/>
    <object type="Socket" os_level="-1" os_index="0" cpuset="0x00000055"
      complete_cpuset="0x00000055" online_cpuset="0x00000055"
      allowed_cpuset="0x00000055">
      <object type="Cache" os_level="-1" cpuset="0x00000011"
        complete_cpuset="0x00000011" online_cpuset="0x00000011"
        allowed_cpuset="0x00000011" cache_size="4194304" depth="2"
        cache_linesize="64">
        <object type="Cache" os_level="-1" cpuset="0x00000001"
          complete_cpuset="0x00000001" online_cpuset="0x00000001"
          allowed_cpuset="0x00000001" cache_size="32768" depth="1"
```

```

        cache_linesize="64">
<object type="Core" os_level="-1" os_index="0" cpuset="0x00000001"
    complete_cpuset="0x00000001" online_cpuset="0x00000001"
    allowed_cpuset="0x00000001">
    <object type="PU" os_level="-1" os_index="0" cpuset="0x00000001"
        complete_cpuset="0x00000001" online_cpuset="0x00000001"
        allowed_cpuset="0x00000001"/>
    </object>
</object>
<object type="Cache" os_level="-1" cpuset="0x00000010"
    complete_cpuset="0x00000010" online_cpuset="0x00000010"
    allowed_cpuset="0x00000010" cache_size="32768" depth="1"
    cache_linesize="64">
    <object type="Core" os_level="-1" os_index="1" cpuset="0x00000010"
        complete_cpuset="0x00000010" online_cpuset="0x00000010"
        allowed_cpuset="0x00000010">
        <object type="PU" os_level="-1" os_index="4" cpuset="0x00000010"
            complete_cpuset="0x00000010" online_cpuset="0x00000010"
            allowed_cpuset="0x00000010"/>
        </object>
    </object>
</object>
<!-- ...more objects listed here ... -->
</topology>

```

## 1.4 Programming Interface

The basic interface is available in [hwloc.h](#). It essentially offers low-level routines for advanced programmers that want to manually manipulate objects and follow links between them. Documentation for everything in [hwloc.h](#) are provided later in this document. Developers should also look at [hwloc/helper.h](#) (and also in this document, which provides good higher-level topology traversal examples).

To precisely define the vocabulary used by hwloc, a [Terms and Definitions](#) section is available and should probably be read first.

Each hwloc object contains a cpuset describing the list of processing units that it contains. These bitmaps may be used for [CPU binding](#) and [Memory binding](#). hwloc offers an extensive bitmap manipulation interface in [hwloc/bitmap.h](#).

Moreover, hwloc also comes with additional helpers for interoperability with several commonly used environments. See the [Interoperability With Other Software](#) section for details.

The complete API documentation is available in a full set of HTML pages, man pages, and self-contained PDF files (formatted for both both US letter and A4 formats) in the source tarball in [doc/doxygen-doc/](#).

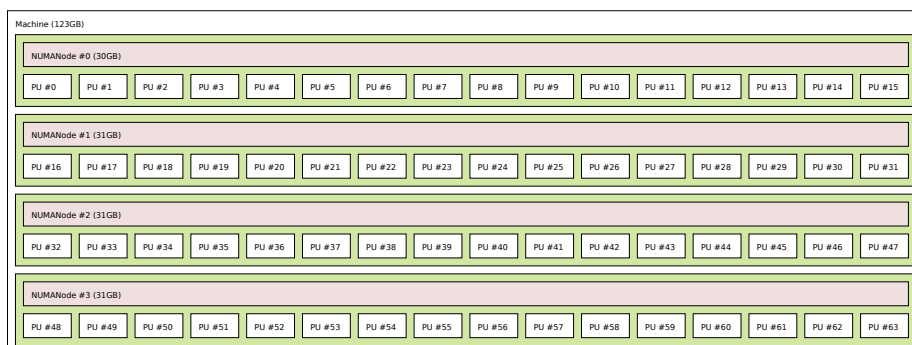
NOTE: If you are building the documentation from a Subversion checkout, you will need to have Doxygen and pdflatex installed -- the documentation will be built during the normal "make" process. The documentation is installed during "make install" to `$prefix/share/doc/hwloc/` and your systems default man page tree (under `$prefix`, of course).

### 1.4.1 Portability

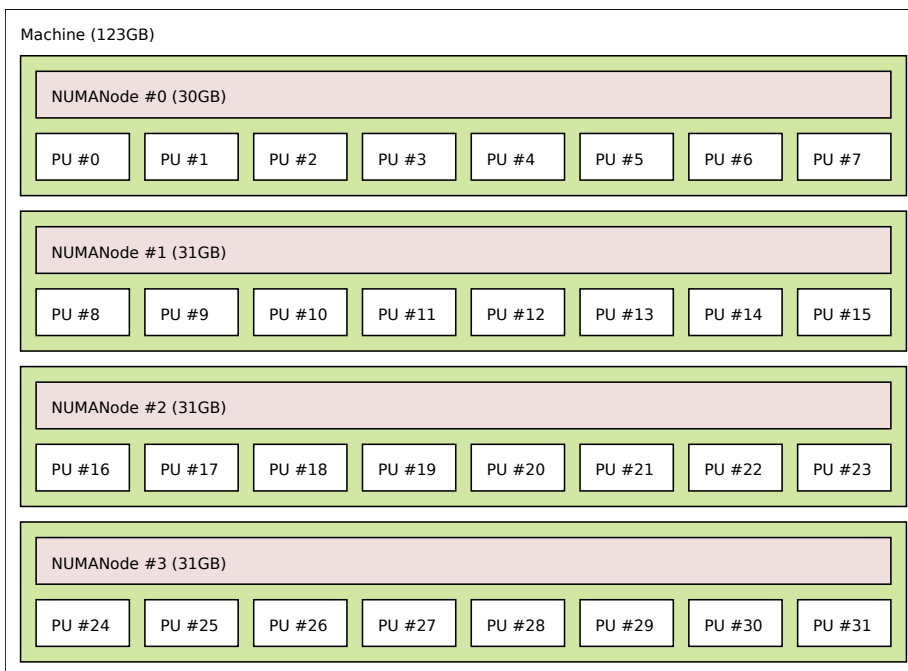
As shown in [CLI Examples](#), hwloc can obtain information on a wide variety of hardware topologies. However, some platforms and/or operating system versions will only report a subset of this information. For example, on an PPC64-based system with 32 cores (each with 2 hardware threads) running a default 2.6.18-based kernel from RHEL 5.4, hwloc is only able to glean information about NUMA nodes and processor units (PUs). No information about caches, sockets, or cores is available.

Similarly, Operating Systems have varying support for CPU and memory binding, e.g. while some Operating Systems provide interfaces for all kinds of CPU and memory bindings, some others provide only interfaces for a limited number of kinds of CPU and memory binding, and some do not provide any binding interface at all. Hwloc's binding functions would then simply return the ENOSYS error (Function not implemented), meaning that the underlying Operating System does not provide any interface for them. [CPU binding](#) and [Memory binding](#) provide more information on which hwloc binding functions should be preferred because interfaces for them are usually available on the supported Operating Systems.

Here's the graphical output from lstopo on this platform when Simultaneous Multi-Threading (SMT) is enabled:



And here's the graphical output from lstopo on this platform when SMT is disabled:



Notice that hwloc only sees half the PUs when SMT is disabled. PU #15, for example, seems to change location from NUMA node #0 to #1. In reality, no PUs "moved" -- they were simply re-numbered when hwloc only saw half as many. Hence, PU #15 in the SMT-disabled picture probably corresponds to PU #30 in the SMT-enabled picture.

This same "PUs have disappeared" effect can be seen on other platforms -- even platforms / OSs that provide much more information than the above PPC64 system. This is an unfortunate side-effect of how operating systems report information to hwloc.

Note that upgrading the Linux kernel on the same PPC64 system mentioned above to 2.6.34, hwloc is able to discover all the topology information. The following picture shows the entire topology layout when SMT is enabled:



Developers using the hwloc API or XML output for portable applications should therefore be extremely careful to not make any assumptions about the structure of data that is returned. For example, per the above reported PPC topology, it is not safe to assume that PUs will always be descendants of cores.

Additionally, future hardware may insert new topology elements that are not available

in this version of hwloc. Long-lived applications that are meant to span multiple different hardware platforms should also be careful about making structure assumptions. For example, there may someday be an element "lower" than a PU, or perhaps a new element may exist between a core and a PU.

### 1.4.2 API Example

The following small C example (named "hwloc-hello.c") prints the topology of the machine and bring the process to the first logical processor of the second core of the machine.

```
/* Example hwloc API program.
 *
 * Copyright © 2009–2010 INRIA
 * Copyright © 2009–2010 Université Bordeaux 1
 * Copyright © 2009–2010 Cisco Systems, Inc. All rights reserved.
 *
 * hwloc-hello.c
 */

#include <hwloc.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void print_children(hwloc_topology_t topology, hwloc_obj_t obj,
                          int depth)
{
    char string[128];
    unsigned i;

    hwloc_obj_snprintf(string, sizeof(string), topology, obj, "#", 0);
    printf("%*s%s\n", 2*depth, "", string);
    for (i = 0; i < obj->arity; i++) {
        print_children(topology, obj->children[i], depth + 1);
    }
}

int main(void)
{
    int depth;
    unsigned i, n;
    unsigned long size;
    int levels;
    char string[128];
    int topodepth;
    hwloc_topology_t topology;
    hwloc_cpuset_t cpuset;
    hwloc_obj_t obj;

    /* Allocate and initialize topology object. */
    hwloc_topology_init(&topology);

    /* ... Optionally, put detection configuration here to ignore
     * some objects types, define a synthetic topology, etc....

    The default is to detect all the objects of the machine that
    the caller is allowed to access. See Configure Topology
    Detection. */
```



```

/* Perform the topology detection. */
hwloc_topology_load(topology);

/* Optionally, get some additional topology information
   in case we need the topology depth later. */
topodepth = hwloc_topology_get_depth(topology);

/*****
 * First example:
 * Walk the topology with an array style, from level 0 (always
 * the system level) to the lowest level (always the proc level).
 *****/
for (depth = 0; depth < topodepth; depth++) {
    printf("*** Objects at level %d\n", depth);
    for (i = 0; i < hwloc_get_nbobjs_by_depth(topology, depth);
         i++) {
        hwloc_obj_snprintf(string, sizeof(string), topology,
                           hwloc_get_obj_by_depth(topology, depth, i),
                           "#", 0);
        printf("Index %u: %s\n", i, string);
    }
}

/*****
 * Second example:
 * Walk the topology with a tree style.
 *****/
printf("*** Printing overall tree\n");
print_children(topology, hwloc_get_root_obj(topology), 0);

/*****
 * Third example:
 * Print the number of sockets.
 *****/
depth = hwloc_get_type_depth(topology, HWLOC_OBJ_SOCKET);
if (depth == HWLOC_TYPE_DEPTH_UNKNOWN) {
    printf("*** The number of sockets is unknown\n");
} else {
    printf("*** %u socket(s)\n",
           hwloc_get_nbobjs_by_depth(topology, depth));
}

/*****
 * Fourth example:
 * Compute the amount of cache that the first logical processor
 * has above it.
 *****/
levels = 0;
size = 0;
for (obj = hwloc_get_obj_by_type(topology, HWLOC_OBJ_PU, 0);
     obj;
     obj = obj->parent)
    if (obj->type == HWLOC_OBJ_CACHE) {
        levels++;
        size += obj->attr->cache.size;
    }
printf("*** Logical processor 0 has %d caches totaling %luKB\n",
       levels, size / 1024);

/*****
 * Fifth example:

```

```

    * Bind to only one thread of the last core of the machine.
    *
    * First find out where cores are, or else smaller sets of CPUs if
    * the OS doesn't have the notion of a "core".
    *****/
    depth = hwloc_get_type_or_below_depth(topology, HWLOC_OBJ_CORE);

    /* Get last core. */
    obj = hwloc_get_obj_by_depth(topology, depth,
                                hwloc_get_nbobjs_by_depth(topology, depth) - 1);
    if (obj) {
        /* Get a copy of its cpuset that we may modify. */
        cpuset = hwloc_bitmap_dup(obj->cpuset);

        /* Get only one logical processor (in case the core is
           SMT/hyperthreaded). */
        hwloc_bitmap_singlify(cpuset);

        /* And try to bind ourself there. */
        if (hwloc_set_cpubind(topology, cpuset, 0)) {
            char *str;
            int error = errno;
            hwloc_bitmap_asprintf(&str, obj->cpuset);
            printf("Couldn't bind to cpuset %s: %s\n", str, strerror(error));
            free(str);
        }

        /* Free our cpuset copy */
        hwloc_bitmap_free(cpuset);
    }

    /******
    * Sixth example:
    * Allocate some memory on the last NUMA node, bind some existing
    * memory to the last NUMA node.
    *****/
    /* Get last node. */
    n = hwloc_get_nbobjs_by_type(topology, HWLOC_OBJ_NODE);
    if (n) {
        void *m;
        size_t size = 1024*1024;

        obj = hwloc_get_obj_by_type(topology, HWLOC_OBJ_NODE, n - 1);
        m = hwloc_alloc_membind_nodeset(topology, size, obj->nodeset,
                                       HWLOC_MEMBIND_DEFAULT, 0);
        hwloc_free(topology, m, size);

        m = malloc(size);
        hwloc_set_area_membind_nodeset(topology, m, size, obj->nodeset,
                                       HWLOC_MEMBIND_DEFAULT, 0);
        free(m);
    }

    /* Destroy topology object. */
    hwloc_topology_destroy(topology);

    return 0;
}

```

hwloc provides a pkg-config executable to obtain relevant compiler and linker flags. For example, it can be used thusly to compile applications that utilize the hwloc

library (assuming GNU Make):

```
CFLAGS += $(pkg-config --cflags hwloc)
LDLIBS += $(pkg-config --libs hwloc)
cc hwloc-hello.c $(CFLAGS) -o hwloc-hello $(LDLIBS)
```

On a machine with 4GB of RAM and 2 processor sockets -- each socket of which has two processing cores -- the output from running `hwloc-hello` could be something like the following:

```
shell$ ./hwloc-hello
*** Objects at level 0
Index 0: Machine(3938MB)
*** Objects at level 1
Index 0: Socket#0
Index 1: Socket#1
*** Objects at level 2
Index 0: Core#0
Index 1: Core#1
Index 2: Core#3
Index 3: Core#2
*** Objects at level 3
Index 0: PU#0
Index 1: PU#1
Index 2: PU#2
Index 3: PU#3
*** Printing overall tree
Machine(3938MB)
  Socket#0
    Core#0
      PU#0
    Core#1
      PU#1
  Socket#1
    Core#3
      PU#2
    Core#2
      PU#3
*** 2 socket(s)
shell$
```

## 1.5 Questions and Bugs

Questions should be sent to the devel mailing list (<http://www.open-mpi.org/community/lists/hwloc.php>). Bug reports should be reported in the tracker (<https://svn.open-mpi.org/trac/hwloc/>).

If `hwloc` discovers an incorrect topology for your machine, the very first thing you should check is to ensure that you have the most recent updates installed for your operating system. Indeed, most of `hwloc` topology discovery relies on hardware information retrieved through the operation system (e.g., via the `/sys` virtual filesystem of the Linux kernel). If upgrading your OS or Linux kernel does not solve your problem, you may also want to ensure that you are running the most recent version of the BIOS for your machine.

If those things fail, contact us on the mailing list for additional help. Please attach the

output of `lstopo` after having given the `--enable-debug` option to `./configure` and rebuilt completely, to get debugging output.

## 1.6 History / Credits

`hwloc` is the evolution and merger of the `libtopology` (<http://runtime.bordeaux.inria.fr/libtopology/>) project and the Portable Linux Processor Affinity (PLPA) (<http://www.open-mpi.org/projects/plpa/>) project. Because of functional and ideological overlap, these two code bases and ideas were merged and released under the name "`hwloc`" as an Open MPI sub-project.

`libtopology` was initially developed by the INRIA Runtime Team-Project (<http://runtime.bordeaux.inria.fr/>) (headed by Raymond Namyst (<http://dept-info.labri.fr/~namyst/>)). PLPA was initially developed by the Open MPI development team as a sub-project. Both are now deprecated in favor of `hwloc`, which is distributed as an Open MPI sub-project.

## 1.7 Further Reading

The documentation chapters include

- [Terms and Definitions](#)
- [Command-Line Tools](#)
- [Environment Variables](#)
- [CPU and Memory Binding Overview](#)
- [Interoperability With Other Software](#)
- [Thread Safety](#)
- [Embedding `hwloc` in Other Software](#)
- [Switching from PLPA to `hwloc`](#)
- [Frequently Asked Questions](#)

Make sure to have had a look at those too!

## Chapter 2

# Terms and Definitions

**Object** Interesting kind of part of the system, such as a Core, a Cache, a Memory node, etc. The different types detected by hwloc are detailed in the [hwloc\\_obj\\_type\\_t](#) enumeration.

They are topologically sorted by CPU set into a tree.

**CPU set** The set of logical processors (or processing units) logically included in an object (if it makes sense). They are always expressed using physical logical processor numbers (as announced by the OS). They are implemented as the [hwloc\\_bitmap\\_t](#) opaque structure. hwloc CPU sets are just masks, they do *not* have any relation with an operating system actual binding notion like Linux' cpusets.

**Node set** The set of NUMA memory nodes logically included in an object (if it makes sense). They are always expressed using physical node numbers (as announced by the OS). They are implemented with the [hwloc\\_bitmap\\_t](#) opaque structure. as bitmaps.

**Bitmap** A possibly-infinite set of bits used for describing sets of objects such as CPUs (CPU sets) or memory nodes (Node sets). They are implemented with the [hwloc\\_bitmap\\_t](#) opaque structure.

**Parent object** The object logically containing the current object, for example because its CPU set includes the CPU set of the current object.

**Ancestor object** The parent object, or its own parent object, and so on.

**Children object(s)** The object (or objects) contained in the current object because their CPU set is included in the CPU set of the current object.

**Arity** The number of children of an object.

**Sibling objects** Objects of the same type which have the same parent.

**Sibling rank** Index to uniquely identify objects of the same type which have the same parent, and is always in the range [0, parent\_arity).

**Cousin objects** Objects of the same type as the current object.

**Level** Set of objects of the same type.

**OS or physical index** The index that the operating system (OS) uses to identify the object. This may be completely arbitrary, or it may depend on the BIOS configuration.

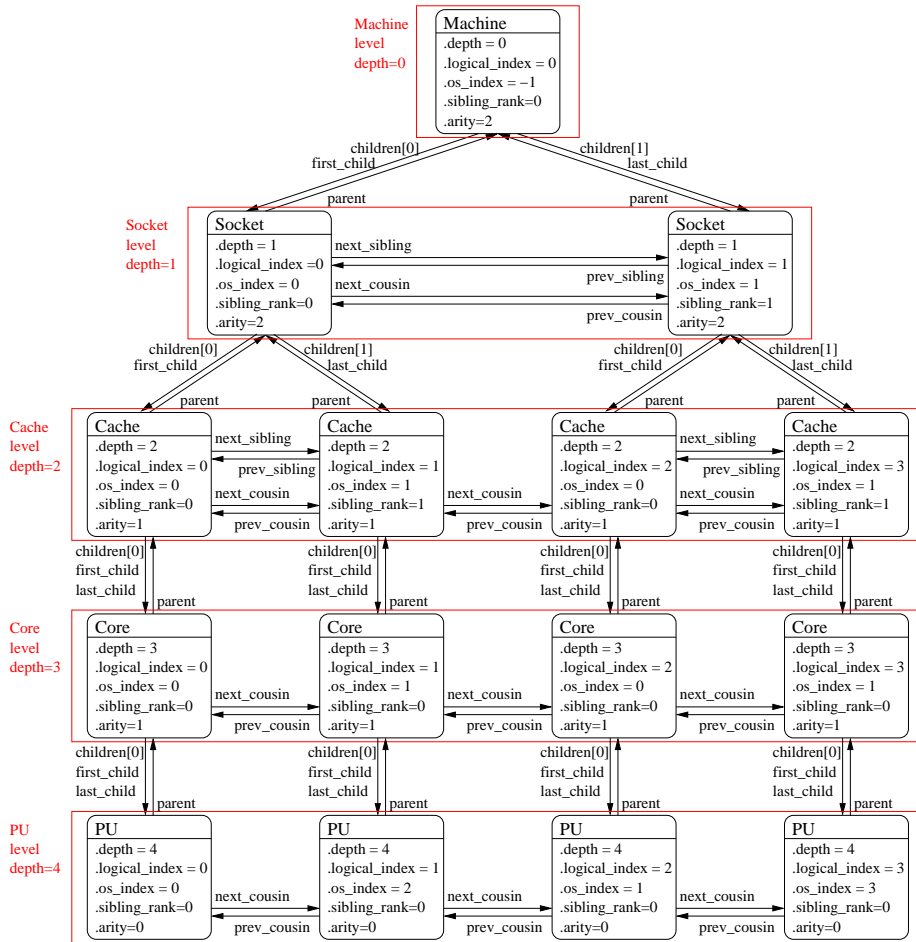
**Depth** Nesting level in the object tree, starting from the 0th object.

**Logical index** Index to uniquely identify objects of the same type. It expresses proximity in a generic way. This index is always linear and in the range [0, num\_objs\_same\_type\_same\_level). Think of it as “cousin rank.” The ordering is based on topology first, and then on OS CPU numbers, so it is stable across everything except firmware CPU renumbering.

**Logical processor**

**Processing unit** The smallest processing element that can be represented by a hwloc object. It may be a single-core processor, a core of a multicore processor, or a single thread in SMT processor.

The following diagram can help to understand the vocabulary of the relationships by showing the example of a machine with two dual core sockets (with no hardware threads); thus, a topology with 4 levels. Each box with rounded corner corresponds to one hwloc\_obj\_t, containing the values of the different integer fields (depth, logical\_index, etc.), and arrows show to which other hwloc\_obj\_t pointers point to (first\_child, parent, etc.)



It should be noted that for PU objects, the logical index -- as computed linearly by hwloc -- is not the same as the OS index.





## Chapter 3

# Command-Line Tools

hwloc comes with an extensive C programming interface and several command line utilities. Each of them is fully documented in its own manual page; the following is a summary of the available command line tools.

### 3.1 lstopo

lstopo (also known as hwloc-info and hwloc-ls) displays the hierarchical topology map of the current system. The output may be graphical or textual, and can also be exported to numerous file formats such as PDF, PNG, XML, and others.

Note that lstopo can read XML files and/or alternate chroot filesystems and display topological maps representing those systems (e.g., use lstopo to output an XML file on one system, and then use lstopo to read in that XML file and display it on a different system).

### 3.2 hwloc-bind

hwloc-bind binds processes to specific hardware objects through a flexible syntax. A simple example is binding an executable to specific cores (or sockets or bitmaps or ...). The hwloc-bind(1) man page provides much more detail on what is possible.

hwloc-bind can also be used to retrieve the current process' binding.

### 3.3 hwloc-calc

hwloc-calc is generally used to create bitmap strings to pass to hwloc-bind. Although hwloc-bind accepts many forms of object specification (i.e., bitmap strings are one of many forms that hwloc-bind understands), they can be useful, compact representations in shell scripts, for example.

`hwloc-calc` generates bitmap strings from given hardware objects with the ability to aggregate them, intersect them, and more. `hwloc-calc` generally uses the same syntax than `hwloc-bind`, but multiple instances may be composed to generate complex combinations.

Note that `hwloc-calc` can also generate lists of logical processors or NUMA nodes that are convenient to pass to some external tools such as `taskset` or `numactl`.

### **3.4 hwloc-distrib**

`hwloc-distrib` generates a set of bitmap strings that are uniformly distributed across the machine for the given number of processes. These strings may be used with `hwloc-bind` to run processes to maximize their memory bandwidth by properly distributing them across the machine.

### **3.5 hwloc-ps**

`hwloc-ps` is a tool to display the bindings of processes that are currently running on the local machine. By default, `hwloc-ps` only lists processes that are bound; unbound process (and Linux kernel threads) are not displayed.

## Chapter 4

# Environment Variables

The behavior of the hwloc library and tools may be tuned thanks to the following environment variables.

`HWLOC_XMLFILE=/path/to/file.xml` enforces the discovery from the given XML file as if `hwloc_topology_set_xml()` had been called. This file may have been generated earlier with `lstopo file.xml`. For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, `HWLOC_THISSYSTEM` should be set 1 in the environment too, to assert that the loaded file is really the underlying system.

`HWLOC_FSROOT=/path/to/linux/filesystem-root/` switches to reading the topology from the specified Linux filesystem root instead of the main file-system root, as if `hwloc_topology_set_fsroot()` had been called. Not using the main file-system root causes `hwloc_topology_is_thissystem()` to return 0. For convenience, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, `HWLOC_THISSYSTEM` should be set 1 in the environment too, to assert that the loaded file is really the underlying system.

`HWLOC_THISSYSTEM=1` enforces the return value of `hwloc_topology_is_thissystem()`.

It means that it makes hwloc assume that the selected backend provides the topology for the system on which we are running, even if it is not the OS-specific backend but the XML backend for instance. This means making the binding functions actually call the OS-specific system calls and really do binding, while the XML backend would otherwise provide empty hooks just returning success. This can be used for efficiency reasons to first detect the topology once, save it to an XML file, and quickly reload it later through the XML backend, but still having binding functions actually do bind.



## Chapter 5

# CPU and Memory Binding Overview

Some operating systems do not systematically provide separate functions for CPU and memory binding. This means that CPU binding functions may have effects on the memory binding policy. Likewise, changing the memory binding policy may change the CPU binding of the current thread. This is often not a problem for applications, so by default hwloc will make use of these functions when they provide better binding support.

If the application does not want the CPU binding to change when changing the memory policy, it needs to use the `HWLOC_MEMBIND_NOCPUBIND` flag to prevent hwloc from using OS functions which would change the CPU binding. Additionally, `HWLOC_CPUBIND_NOMEMBIND` can be passed to CPU binding function to prevent hwloc from using OS functions would change the memory binding policy. Of course, using these flags will reduce hwloc's overall support for binding, so their use is discouraged.

One can avoid using these flags but still closely control both memory and CPU binding by allocating memory, touching each page in the allocated memory, and then changing the CPU binding. The already-really-allocated memory will then be "locked" to physical memory and will not be migrated. Thus, even if the memory binding policy gets changed by the CPU binding order, the already-allocated memory will not change with it. When binding and allocating further memory, the CPU binding should be performed again in case the memory binding altered the previously-selected CPU binding.

Not all operating systems support the notion of a "current" memory binding policy for the current process, but such operating systems often still provide a way to allocate data on a given node set. Conversely, some operating systems support the notion of a "current" memory binding policy and do not permit allocating data on a specific node set without changing the current policy and allocate the data. To provide the most powerful coverage of these facilities, hwloc provides:

- functions that set/get the current memory binding policies (if supported): `hwloc_set/get_membind_*` and `hwloc_set/get_proc_membind()`
- functions that allocate memory bound to specific node set without changing the current memory binding policy (if supported): `hwloc_alloc_membind()` and `hwloc_alloc_membind_nodeuset()`.

- helpers which, if needed, change the current memory binding policy of the process in order to obtain memory binding: [hwloc\\_alloc\\_membind\\_policy\(\)](#) and [hwloc\\_alloc\\_membind\\_policy\\_nodest\(\)](#)

An application can thus use the two first sets of functions if it wants to manage separately the global process binding policy and directed allocation, or use the third set of functions if it does not care about the process memory binding policy.

See [CPU binding](#) and [Memory binding](#) for hwloc's API functions regarding CPU and memory binding, respectively.

## Chapter 6

# Interoperability With Other Software

Although hwloc offers its own portable interface, it still may have to interoperate with specific or non-portable libraries that manipulate similar kinds of objects. hwloc therefore offers several specific "helpers" to assist converting between those specific interfaces and hwloc.

Some external libraries may be specific to a particular OS; others may not always be available. The hwloc core therefore generally does not explicitly depend on these types of libraries. However, when a custom application uses or otherwise depends on such a library, it may optionally include the corresponding hwloc helper to extend the hwloc interface with dedicated helpers.

Linux specific features [hwloc/linux.h](#) offers Linux-specific helpers that utilize some non-portable features of the Linux system, such as binding threads through their thread ID ("tid") or parsing kernel CPU mask files.

Linux libnuma [hwloc/linux-libnuma.h](#) provides conversion helpers between hwloc CPU sets and libnuma-specific types, such as nodemasks and bitmasks. It helps you use libnuma memory-binding functions with hwloc CPU sets.

Glibc [hwloc/glibc-sched.h](#) offers conversion routines between Glibc and hwloc CPU sets in order to use hwloc with functions such as `sched_setaffinity()`.

OpenFabrics Verbs [hwloc/openfabrics-verbs.h](#) helps interoperability with the OpenFabrics Verbs interface. For example, it can return a list of processors near an OpenFabrics device.

Myrinet Express [hwloc/myriexpress.h](#) offers interoperability with the Myrinet Express interface. It can return the list of processors near a Myrinet board managed by the MX driver.

NVIDIA CUDA [hwloc/cuda.h](#) and [hwloc/cudart.h](#) enable interoperability with NVIDIA CUDA Driver and Runtime interfaces. For instance, it may return the list of processors near NVIDIA GPUs.

Taskset command-line tool The taskset command-line tool is widely used for binding processes. It manipulates CPU set strings in a format that is slightly different

from hwloc's one (it does not divide the string in fixed-size subsets and separates them with commas). To ease interoperability, hwloc offers routines to convert hwloc CPU sets from/to taskset-specific string format. Most hwloc command-line tools also support the `--taskset` option to manipulate taskset-specific strings.



## Chapter 7

# Thread Safety

Like most libraries that mainly fill data structures, hwloc is not thread safe but rather reentrant: all state is held in a [hwloc\\_topology\\_t](#) instance without mutex protection. That means, for example, that two threads can safely operate on and modify two different [hwloc\\_topology\\_t](#) instances, but they should not simultaneously invoke functions that modify the *same* instance. Similarly, one thread should not modify a [hwloc\\_topology\\_t](#) instance while another thread is reading or traversing it. However, two threads can safely read or traverse the same [hwloc\\_topology\\_t](#) instance concurrently.

When running in multiprocessor environments, be aware that proper thread synchronization and/or memory coherency protection is needed to pass hwloc data (such as [hwloc\\_topology\\_t](#) pointers) from one processor to another (e.g., a mutex, semaphore, or a memory barrier). Note that this is not a hwloc-specific requirement, but it is worth mentioning.

For reference, [hwloc\\_topology\\_t](#) modification operations include (but may not be limited to):

Creation and destruction [hwloc\\_topology\\_init\(\)](#), [hwloc\\_topology\\_load\(\)](#), [hwloc\\_topology\\_destroy\(\)](#) (see [Create and Destroy Topologies](#)) imply major modifications of the structure, including freeing some objects. No other thread cannot access the topology or any of its objects at the same time.

Also references to objects inside the topology are not valid anymore after these functions return.

Runtime topology modifications [hwloc\\_topology\\_insert\\_misc\\_object\\_by\\_\\*](#) (see [Tinker with topologies.](#)) may modify the topology significantly by adding objects inside the tree, changing the topology depth, etc.

Although references to former objects *may* still be valid after insertion, it is strongly advised to not rely on any such guarantee and always re-consult the topology to reacquire new instances of objects.

Locating topologies [hwloc\\_topology\\_ignore\\*](#), [hwloc\\_topology\\_set\\*](#) (see [Configure Topology Detection](#)) do not modify the topology directly, but they do modify internal structures describing the behavior of the next invocation of

`hwloc_topology_load()`. Hence, all of these functions should not be used concurrently.

Note that these functions do not modify the current topology until it is actually reloaded; it is possible to use them while other threads are only read the current topology.

## Chapter 8

# Embedding hwloc in Other Software

It can be desirable to include hwloc in a larger software package (be sure to check out the LICENSE file) so that users don't have to separately download and install it before installing your software. This can be advantageous to ensure that your software uses a known-tested/good version of hwloc, or for use on systems that do not have hwloc pre-installed.

When used in "embedded" mode, hwloc will:

- not install any header files
- not build any documentation files
- not build or install any executables or tests
- not build `libhwloc.*` -- instead, it will build `libhwloc_embedded.*`

There are two ways to put hwloc into "embedded" mode. The first is directly from the configure command line:

```
shell$ ./configure --enable-embedded-mode ...
```

The second requires that your software project uses the GNU Autoconf / Automake / Libtool tool chain to build your software. If you do this, you can directly integrate hwloc's m4 configure macro into your configure script. You can then invoke hwloc's configuration tests and build setup by calling an m4 macro (see below).

### 8.1 Using hwloc's M4 Embedding Capabilities

Every project is different, and there are many different ways of integrating hwloc into yours. What follows is *one* example of how to do it.

If your project uses recent versions Autoconf, Automake, and Libtool to build, you can use hwloc's embedded m4 capabilities. We have tested the embedded m4 with projects

that use Autoconf 2.65, Automake 1.11.1, and Libtool 2.2.6b. Slightly earlier versions of may also work but are untested. Autoconf versions prior to 2.65 are almost certain to not work.

You can either copy all the config/hwloc\*m4 files from the hwloc source tree to the directory where your project's m4 files reside, or you can tell aclocal to find more m4 files in the embedded hwloc's "config" subdirectory (e.g., add "-Ipath/to/embedded/hwloc/config" to your Makefile.am's ACLOCAL\_AMFLAGS).

The following macros can then be used from your configure script (only HWLOC\_SETUP\_CORE *must* be invoked if using the m4 macros):

- **HWLOC\_SETUP\_CORE**(config-dir-prefix, action-upon-success, action-upon-failure, print\_banner\_or\_not): Invoke the hwloc configuration tests and setup the hwloc tree to build. The first argument is the prefix to use for AC\_OUTPUT files -- it's where the hwloc tree is located relative to \$top\_srcdir. Hence, if your embedded hwloc is located in the source tree at contrib/hwloc, you should pass [contrib/hwloc] as the first argument. If HWLOC\_SETUP\_CORE and the rest of configure completes successfully, then "make" traversals of the hwloc tree with standard Automake targets (all, clean, install, etc.) should behave as expected. For example, it is safe to list the hwloc directory in the SUBDIRS of a higher-level Makefile.am. The last argument, if not empty, will cause the macro to display an announcement banner that it is starting the hwloc core configuration tests.

HWLOC\_SETUP\_CORE will set the following environment variables and AC\_SUBST them: HWLOC\_EMBEDDED\_CFLAGS, HWLOC\_EMBEDDED\_CPPFLAGS, and HWLOC\_EMBEDDED\_LIBS. These flags are filled with the values discovered in the hwloc-specific m4 tests, and can be used in your build process as relevant. The \_CFLAGS, \_CPPFLAGS, and \_LIBS variables are necessary to build libhwloc (or libhwloc\_embedded) itself.

HWLOC\_SETUP\_CORE also sets HWLOC\_EMBEDDED\_LDADD environment variable (and AC\_SUBSTs it) to contain the location of the libhwloc\_embedded.la convenience Libtool archive. It can be used in your build process to link an application or other library against the embedded hwloc library.

NOTE: If the HWLOC\_SET\_SYMBOL\_PREFIX macro is used, it must be invoked before HWLOC\_SETUP\_CORE.

- **HWLOC\_BUILD\_STANDALONE**: HWLOC\_SETUP\_CORE defaults to building hwloc in an "embedded" mode (described above). If HWLOC\_BUILD\_STANDALONE is invoked *\*before\** HWLOC\_SETUP\_CORE, the embedded definitions will not apply (e.g., libhwloc.la will be built, not libhwloc\_embedded.la).
- **HWLOC\_SET\_SYMBOL\_PREFIX**(foo\_): Tells the hwloc to prefix all of hwloc's types and public symbols with "foo\_"; meaning that function hwloc\_init() becomes foo\_hwloc\_init(). Enum values are prefixed with an upper-case translation if the prefix supplied; HWLOC\_OBJ\_SYSTEM becomes FOO\_HWLOC\_OBJ\_SYSTEM. This is recommended behavior if you are including hwloc in middleware -- it is possible that your software will be combined with other software that links to another copy of hwloc. If both uses of hwloc utilize different

symbol prefixes, there will be no type/symbol clashes, and everything will compile, link, and run successfully. If you both embed hwloc without changing the symbol prefix and also link against an external hwloc, you may get multiple symbol definitions when linking your final library or application.

- **HWLOC\_SETUP\_DOCS, HWLOC\_SETUP\_UTILS, HWLOC\_SETUP\_TESTS:** These three macros only apply when hwloc is built in "standalone" mode (i.e., they should NOT be invoked unless HWLOC\_BUILD\_STANDALONE has already been invoked).
- **HWLOC\_DO\_AM\_CONDITIONALS:** If you embed hwloc in a larger project and build it conditionally with Automake (e.g., if HWLOC\_SETUP\_CORE is invoked conditionally), you must unconditionally invoke HWLOC\_DO\_AM\_CONDITIONALS to avoid warnings from Automake (for the cases where hwloc is not selected to be built). This macro is necessary because hwloc uses some AM\_CONDITIONALS to build itself, and AM\_CONDITIONALS cannot be defined conditionally. Note that it is safe (but unnecessary) to call HWLOC\_DO\_AM\_CONDITIONALS even if HWLOC\_SETUP\_CORE is invoked unconditionally. If you are not using Automake to build hwloc, this macro is unnecessary (and will actually cause errors because it invoked AM\_\* macros that will be undefined).

NOTE: When using the HWLOC\_SETUP\_CORE m4 macro, it may be necessary to explicitly invoke AC\_CANONICAL\_TARGET (which requires config.sub and config.guess) and/or AC\_USE\_SYSTEM\_EXTENSIONS macros early in the configure script (e.g., after AC\_INIT but before AM\_INIT\_AUTOMAKE). See the Autoconf documentation for further information.

Also note that hwloc's top-level configure.ac script uses exactly the macros described above to build hwloc in a standalone mode (by default). You may want to examine it for one example of how these macros are used.

## 8.2 Example Embedding hwloc

Here's an example of integrating with a larger project named sandbox that already uses Autoconf, Automake, and Libtool to build itself:

```
# First, cd into the sandbox project source tree
shell$ cd sandbox
shell$ cp -r /somewhere/else/hwloc-<version> my-embedded-hwloc
shell$ edit Makefile.am
1. Add "-Imy-embedded-hwloc/config" to ACLOCAL_AMFLAGS
2. Add "my-embedded-hwloc" to SUBDIRS
3. Add "$(HWLOC_EMBEDDED_LDADD)" and "$(HWLOC_EMBEDDED_LIBS)" to
   sandbox's executable's LDADD line. The former is the name of the
   Libtool convenience library that hwloc will generate. The latter
   is any dependent support libraries that may be needed by
   $(HWLOC_EMBEDDED_LDADD).
4. Add "$(HWLOC_EMBEDDED_CFLAGS)" to AM_CFLAGS
5. Add "$(HWLOC_EMBEDDED_CPPFLAGS)" to AM_CPPFLAGS
shell$ edit configure.ac
```

1. Add `"HWLOC_SET_SYMBOL_PREFIX(sandbox_hwloc_)"` line
2. Add `"HWLOC_SETUP_CORE([my-embedded-hwloc], [happy=yes], [happy=no])"` line
3. Add error checking for `happy=no` case

```
shell$ edit sandbox.c
```

1. Add `#include <hwloc.h>`
2. Add calls to `sandbox_hwloc_init()` and other hwloc API functions

Now you can bootstrap, configure, build, and run the sandbox as normal -- all calls to `"sandbox_hwloc_*`" will use the embedded hwloc rather than any system-provided copy of hwloc.

## Chapter 9

# Switching from PLPA to hwloc

Although PLPA and hwloc share some of the same ideas, their programming interfaces are quite different. After much debate, it was decided *not* to emulate the PLPA API with hwloc's API because hwloc's API is already far more rich than PLPA's.

More specifically, exploiting modern computing architecture *requires* the flexible functionality provided by the hwloc API -- the PLPA API is too rigid in its definitions and practices to handle the evolving server hardware landscape (e.g., PLPA only understands cores and sockets; hwloc understands a much larger set of hardware objects).

As such, even though it is fully possible to emulate the PLPA API with hwloc (e.g., only deal with sockets and cores), and while the documentation below describes how to do this, we encourage any existing PLPA application authors to actually re-think their application in terms of more than just sockets and cores. In short, we encourage you to use the full hwloc API to exploit *all* the hardware.

### 9.1 Topology Context vs. Caching

First, all hwloc functions take a `topology` parameter. This parameter serves as an internal storage for the result of the topology discovery. It replaces PLPA's caching abilities and even lets you manipulate multiple topologies at the same time, if needed.

Thus, all programs should first run `hwloc_topology_init()` and `hwloc_topology_destroy()` as they did `plpa_init()` and `plpa_finalize()` in the past.

### 9.2 Hierarchy vs. Core@Socket

PLPA was designed to understand only cores and sockets. hwloc offers many more different types of objects (e.g., cores, sockets, hardware threads, NUMA nodes, and others) and stores them within a tree of resources.

To emulate the PLPA model, it is possible to find sockets using functions such as `hwloc_get_obj_by_type()`. Iterating over sockets is also possible using `hwloc_get_-`

`next_obj_by_type()`. Then, finding a core within a socket may be done using `hwloc_get_obj_inside_cpuset_by_type()` or `hwloc_get_next_obj_inside_cpuset_by_type()`.

It is also possible to directly find an object "below" another object using `hwloc_get_obj_below_by_type()` (or `hwloc_get_obj_below_array_by_type()`).

### 9.3 Logical vs. Physical/OS Indexes

hwloc manipulates logical indexes, meaning indexes specified with regard to the ordering of objects in the hwloc-provided hierarchical tree. Physical or OS indexes may be entirely hidden if not strictly required. The reason for this is that physical/OS indexes may change with the OS or with the BIOS version. They may be non-consecutive, multiple objects may have the same physical/OS indexes, making their manipulation tricky and highly non-portable.

Note that hwloc tries very hard to always present a hierarchical tree with the same logical ordering, regardless of physical or OS index ordering.

It is still possible to retrieve physical/OS indexes through the `os_index` field of objects, but such practice should be avoided as much as possible for the reasons described above (except perhaps for prettyprinting / debugging purposes).

**HWLOC\_OBJ\_PU** objects are supposed to have different physical/OS indexes since the OS uses them for binding. The `os_index` field of these objects provides the identifier that may be used for such binding, and `hwloc_get_proc_obj_by_os_index()` finds the object associated with a specific OS index.

But as mentioned above, we discourage the use of these conversion methods for actual binding. Instead, hwloc offers its own binding model using the `cpuset` field of objects. These cpusets may be duplicated, modified, combined, etc. (see [hwloc/bitmap.h](#) for details) and then passed to `hwloc_set_cpubind()` for binding.

### 9.4 Counting Specification

PLPA offers a `countspec` parameter to specify whether counting all CPUs, only the online ones or only the offline ones. However, some operating systems do not expose the topology of offline CPUs (i.e., offline CPUs are not reported at all by the OS). Also, some processors may not be visible to the current application due to administrative restrictions. Finally, some processors let you shutdown a single hardware thread in a core, making some of the PLPA features irrelevant.

hwloc stores in the hierarchical tree of objects all CPUs that have known topology information. It then provides the applications with several cpusets that contain the list of CPUs that are actually known, that have topology information, that are online, or that are available to the application. These cpusets may be retrieved with `hwloc_topology_get_online_cpuset()` and other similar functions to filter the object that are relevant or not.



## Chapter 10

# Frequently Asked Questions

### 10.1 I do not want hwloc to rediscover my enormous machine topology every time I rerun a process

Although the topology discovery is not expensive on common machines, its overhead may become significant when multiple processes repeat the discovery on large machines (for instance when starting one process per core in a parallel application). The machine topology usually does not vary much, except if some cores are stopped/restarted or if the administrator restrictions are modified. Thus rediscovering the whole topology again and again may look useless.

For this purpose, hwloc offers XML import/export features. It lets you save the discovered topology to a file (for instance with the `lstopo` program) and reload it later by setting the `HWLOC_XMLFILE` environment variable. Loading a XML topology is usually much faster than querying multiple files or calling multiple functions of the operating system. It is also possible to manipulate such XML files with the C programming interface, and the import/export may also be directed to memory buffer (that may for instance be transmitted between applications through a socket).

### 10.2 How do I handle API upgrades?

The hwloc interface is extended with every new major release. Any application using the hwloc API should be prepared to check at compile-time whether some features are available in the currently installed hwloc distribution.

To check whether hwloc is at least 1.1, you should use:

```
#include <hwloc.h>
#if HWLOC_API_VERSION >= 0x00010100
...
#endif
```

One of the major changes in hwloc 1.1 is the addition of the bitmap API. It supersedes the now deprecated cpuset API which will be removed in a future hwloc release. It

is strongly recommended to switch existing codes to the bitmap API. Keeping support for older hwloc versions is easy. For instance, if your code uses `hwloc_cpuset_alloc`, you should use `hwloc_bitmap_alloc` instead and add the following code to one of your common headers:

```
#include <hwloc.h>
#if HWLOC_API_VERSION < 0x00010100
#define hwloc_bitmap_alloc hwloc_cpuset_alloc
#endif
```

Similarly, the hwloc 1.0 interface may be detected by comparing `HWLOC_API_VERSION` with `0x00010000`.

hwloc 0.9 did not define any `HWLOC_API_VERSION` but this very old release probably does not deserve support from your application anymore.

# Chapter 11

## Module Index

### 11.1 Modules

Here is a list of all modules:

API version . . . . .	43
Topology context . . . . .	43
Object sets (hwloc_cpuset_t and hwloc_nodeset_t) . . . . .	44
Topology Object Types . . . . .	45
Topology Objects . . . . .	47
Create and Destroy Topologies . . . . .	47
Configure Topology Detection . . . . .	48
Tinker with topologies. . . . .	53
Get some Topology Information . . . . .	54
Retrieve Objects . . . . .	55
Object/String Conversion . . . . .	56
CPU binding . . . . .	58
Memory binding . . . . .	62
Object Type Helpers . . . . .	71
Basic Traversal Helpers . . . . .	72
Finding Objects Inside a CPU set . . . . .	74
Finding a single Object covering at least CPU set . . . . .	76
Finding a set of similar Objects covering at least a CPU set . . . . .	77
Cache-specific Finding Helpers . . . . .	77
Advanced Traversal Helpers . . . . .	78
Binding Helpers . . . . .	79
Cpuset Helpers . . . . .	81
Nodeset Helpers . . . . .	82
Conversion between cpuset and nodeset . . . . .	83
The bitmap API . . . . .	85
Helpers for manipulating glibc sched affinity . . . . .	93
Linux-only helpers . . . . .	94
Helpers for manipulating Linux libnuma unsigned long masks . . . . .	95
Helpers for manipulating Linux libnuma bitmask . . . . .	96

Helpers for manipulating Linux libnuma nodemask_t . . . . .	97
CUDA Driver API Specific Functions . . . . .	98
CUDA Runtime API Specific Functions . . . . .	99
OpenFabrics-Specific Functions . . . . .	99
Myrinet Express-Specific Functions . . . . .	100

## Chapter 12

# Data Structure Index

### 12.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">hwloc_obj_attr_u::hwloc_cache_attr_s</a> (Cache-specific Object Attributes ) . . . .	101
<a href="#">hwloc_obj_attr_u::hwloc_group_attr_s</a> (Group-specific Object Attributes ) . . . .	102
<a href="#">hwloc_obj</a> (Structure of a topology object ) . . . . .	102
<a href="#">hwloc_obj_attr_u</a> (Object type-specific Attributes ) . . . . .	108
<a href="#">hwloc_obj_info_s</a> (Object info ) . . . . .	108
<a href="#">hwloc_obj_memory_s::hwloc_obj_memory_page_type_s</a> (Array of local memory page types, NULL if no local memory and <code>page_types</code> is 0 ) . . .	109
<a href="#">hwloc_obj_memory_s</a> (Object memory ) . . . . .	110
<a href="#">hwloc_topology_cpubind_support</a> (Flags describing actual PU binding support for this topology ) . . . . .	111
<a href="#">hwloc_topology_discovery_support</a> (Flags describing actual discovery support for this topology ) . . . . .	112
<a href="#">hwloc_topology_membind_support</a> (Flags describing actual memory binding support for this topology ) . . . . .	113
<a href="#">hwloc_topology_support</a> (Set of flags describing actual support for this topol- ogy ) . . . . .	115



## Chapter 13

# Module Documentation

### 13.1 API version

#### Defines

- `#define HWLOC\_API\_VERSION 0x00010100`

#### Functions

- unsigned [hwloc\\_get\\_api\\_version](#) (void)

#### 13.1.1 Define Documentation

##### 13.1.1.1 `#define HWLOC_API_VERSION 0x00010100`

Indicate at build time which hwloc API version is being used.

#### 13.1.2 Function Documentation

##### 13.1.2.1 `unsigned hwloc_get_api_version ( void )`

Indicate at runtime which hwloc API version was used at build time.

### 13.2 Topology context

#### Typedefs

- `typedef struct hwloc_topology * hwloc\_topology\_t`

### 13.2.1 Typedef Documentation

#### 13.2.1.1 `typedef struct hwloc_topology* hwloc_topology_t`

Topology context.

To be initialized with [hwloc\\_topology\\_init\(\)](#) and built with [hwloc\\_topology\\_load\(\)](#).

## 13.3 Object sets (`hwloc_cpuset_t` and `hwloc_node_t`)

### Typedefs

- `typedef hwloc\_bitmap\_t hwloc_cpuset_t`
- `typedef hwloc\_const\_bitmap\_t hwloc_const_cpuset_t`
- `typedef hwloc\_bitmap\_t hwloc_node_t`
- `typedef hwloc\_const\_bitmap\_t hwloc_const_node_t`

#### 13.3.1 Detailed Description

Hwloc uses bitmaps to represent two distinct kinds of object sets: CPU sets ([hwloc\\_cpuset\\_t](#)) and NUMA node sets ([hwloc\\_node\\_t](#)). These types are both typedefs to a common back end type ([hwloc\\_bitmap\\_t](#)), and therefore all the hwloc bitmap functions are applicable to both [hwloc\\_cpuset\\_t](#) and [hwloc\\_node\\_t](#) (see [The bitmap API](#)).

The rationale for having two different types is that even though the actions one wants to perform on these types are the same (e.g., enable and disable individual items in the set/mask), they're used in very different contexts: one for specifying which processors to use and one for specifying which NUMA nodes to use. Hence, the name difference is really just to reflect the intent of where the type is used.

#### 13.3.2 Typedef Documentation

##### 13.3.2.1 `typedef hwloc\_const\_bitmap\_t hwloc_const_cpuset_t`

A non-modifiable [hwloc\\_cpuset\\_t](#).

##### 13.3.2.2 `typedef hwloc\_const\_bitmap\_t hwloc_const_node_t`

A non-modifiable [hwloc\\_node\\_t](#).

##### 13.3.2.3 `typedef hwloc\_bitmap\_t hwloc_cpuset_t`

A CPU set is a bitmap whose bits are set according to CPU physical OS indexes.

It may be consulted and modified with the bitmap API as any [hwloc\\_bitmap\\_t](#) (see [hwloc/bitmap.h](#)).



**13.3.2.4 typedef hwloc\_bitmap\_t hwloc\_nodest\_t**

A node set is a bitmap whose bits are set according to NUMA memory node physical OS indexes.

It may be consulted and modified with the bitmap API as any [hwloc\\_bitmap\\_t](#) (see [hwloc/bitmap.h](#)).

When binding memory on a system without any NUMA node (when the whole memory is considered as a single memory bank), the nodeset may be either empty (no memory selected) or full (whole system memory selected).

See also [Conversion between cpuset and nodeset](#).

**13.4 Topology Object Types****Enumerations**

- enum [hwloc\\_obj\\_type\\_t](#) {  
[HWLOC\\_OBJ\\_SYSTEM](#), [HWLOC\\_OBJ\\_MACHINE](#), [HWLOC\\_OBJ\\_NODE](#),  
[HWLOC\\_OBJ\\_SOCKET](#),  
[HWLOC\\_OBJ\\_CACHE](#), [HWLOC\\_OBJ\\_CORE](#), [HWLOC\\_OBJ\\_PU](#), [HWLOC\\_OBJ\\_GROUP](#),  
[HWLOC\\_OBJ\\_MISC](#) }
- enum [hwloc\\_compare\\_types\\_e](#) { [HWLOC\\_TYPE\\_UNORDERED](#) }

**Functions**

- int [hwloc\\_compare\\_types](#) ([hwloc\\_obj\\_type\\_t](#) type1, [hwloc\\_obj\\_type\\_t](#) type2)

**13.4.1 Enumeration Type Documentation****13.4.1.1 enum hwloc\_compare\_types\_e**

Enumerator:

[HWLOC\\_TYPE\\_UNORDERED](#) Value returned by [hwloc\\_compare\\_types](#) when types can not be compared.

**13.4.1.2 enum hwloc\_obj\_type\_t**

Type of topology object.

Note

Do not rely on the ordering or completeness of the values as new ones may be defined in the future! If you need to compare types, use [hwloc\\_compare\\_types\(\)](#) instead.

Enumerator:

- HWLOC\_OBJ\_SYSTEM Whole system (may be a cluster of machines). The whole system that is accessible to hwloc. That may comprise several machines in SSI systems like Kerrighed.
- HWLOC\_OBJ\_MACHINE Machine. The typical root object type. A set of processors and memory with cache coherency.
- HWLOC\_OBJ\_NODE NUMA node. A set of processors around memory which the processors can directly access.
- HWLOC\_OBJ\_SOCKET Socket, physical package, or chip. In the physical meaning, i.e. that you can add or remove physically.
- HWLOC\_OBJ\_CACHE Data cache. Can be L1, L2, L3, ...
- HWLOC\_OBJ\_CORE Core. A computation unit (may be shared by several logical processors).
- HWLOC\_OBJ\_PU Processing Unit, or (Logical) Processor. An execution unit (may share a core with some other logical processors, e.g. in the case of an SMT core). Objects of this kind are always reported and can thus be used as fallback when others are not.
- HWLOC\_OBJ\_GROUP Group objects. Objects which do not fit in the above but are detected by hwloc and are useful to take into account for affinity. For instance, some operating systems expose their arbitrary processors aggregation this way. And hwloc may insert such objects to group NUMA nodes according to their distances. These objects are ignored when they do not bring any structure.
- HWLOC\_OBJ\_MISC Miscellaneous objects. Objects without particular meaning, that can e.g. be added by the application for its own use.

### 13.4.2 Function Documentation

#### 13.4.2.1 `int hwloc_compare_types ( hwloc_obj_type_t type1, hwloc_obj_type_t type2 ) const`

Compare the depth of two object types.

Types shouldn't be compared as they are, since newer ones may be added in the future. This function returns less than, equal to, or greater than zero respectively if `type1` objects usually include `type2` objects, are the same as `type2` objects, or are included in `type2` objects. If the types can not be compared (because neither is usually contained in the other), `HWLOC_TYPE_UNORDERED` is returned. Object types containing CPUs can always be compared (usually, a system contains machines which contain nodes which contain sockets which contain caches, which contain cores, which contain processors).

Note

HWLOC\_OBJ\_PU will always be the deepest.

This does not mean that the actual topology will respect that order: e.g. as of today cores may also contain caches, and sockets may also contain nodes. This is thus just to be seen as a fallback comparison method.

## 13.5 Topology Objects

### Data Structures

- struct `hwloc_obj_memory_s`  
*Object memory.*
- struct `hwloc_obj`  
*Structure of a topology object.*
- union `hwloc_obj_attr_u`  
*Object type-specific Attributes.*
- struct `hwloc_obj_info_s`  
*Object info.*

### Typedefs

- typedef struct `hwloc_obj` \* `hwloc_obj_t`

### 13.5.1 Typedef Documentation

#### 13.5.1.1 typedef struct hwloc\_obj\* hwloc\_obj\_t

Convenience typedef; a pointer to a struct `hwloc_obj`.

## 13.6 Create and Destroy Topologies

### Functions

- int `hwloc_topology_init` (`hwloc_topology_t` \*topologyp)
- int `hwloc_topology_load` (`hwloc_topology_t` topology)
- void `hwloc_topology_destroy` (`hwloc_topology_t` topology)
- void `hwloc_topology_check` (`hwloc_topology_t` topology)

### 13.6.1 Function Documentation

#### 13.6.1.1 void hwloc\_topology\_check ( hwloc\_topology\_t topology )

Run internal checks on a topology structure.

#### Parameters

<code>topology</code>	is the topology to be checked
-----------------------	-------------------------------

### 13.6.1.2 void hwloc\_topology\_destroy ( hwloc\_topology\_t *topology* )

Terminate and free a topology context.

Parameters

<i>topology</i>	is the topology to be freed
-----------------	-----------------------------

### 13.6.1.3 int hwloc\_topology\_init ( hwloc\_topology\_t \* *topologyp* )

Allocate a topology context.

Parameters

out	<i>topologyp</i>	is assigned a pointer to the new allocated context.
-----	------------------	---

Returns

0 on success, -1 on error.

### 13.6.1.4 int hwloc\_topology\_load ( hwloc\_topology\_t *topology* )

Build the actual topology.

Build the actual topology once initialized with [hwloc\\_topology\\_init\(\)](#) and tuned with [Configure Topology Detection](#) routines. No other routine may be called earlier using this topology context.

Parameters

<i>topology</i>	is the topology to be loaded with objects.
-----------------	--

Returns

0 on success, -1 on error.

See also

[Configure Topology Detection](#)

## 13.7 Configure Topology Detection

### Data Structures

- struct [hwloc\\_topology\\_discovery\\_support](#)  
*Flags describing actual discovery support for this topology.*
- struct [hwloc\\_topology\\_cpubind\\_support](#)

*Flags describing actual PU binding support for this topology.*

- struct `hwloc_topology_membind_support`  
*Flags describing actual memory binding support for this topology.*
- struct `hwloc_topology_support`  
*Set of flags describing actual support for this topology.*

## Enumerations

- enum `hwloc_topology_flags_e` { `HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM`, `HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM` }

## Functions

- int `hwloc_topology_ignore_type` (`hwloc_topology_t` topology, `hwloc_obj_type_t` type)
- int `hwloc_topology_ignore_type_keep_structure` (`hwloc_topology_t` topology, `hwloc_obj_type_t` type)
- int `hwloc_topology_ignore_all_keep_structure` (`hwloc_topology_t` topology)
- int `hwloc_topology_set_flags` (`hwloc_topology_t` topology, unsigned long flags)
- int `hwloc_topology_set_fsroot` (`hwloc_topology_t` restrict topology, const char \*restrict fsroot\_path)
- int `hwloc_topology_set_pid` (`hwloc_topology_t` restrict topology, `hwloc_pid_t` pid)
- int `hwloc_topology_set_synthetic` (`hwloc_topology_t` restrict topology, const char \*restrict description)
- int `hwloc_topology_set_xml` (`hwloc_topology_t` restrict topology, const char \*restrict xmlpath)
- int `hwloc_topology_set_xmlbuffer` (`hwloc_topology_t` restrict topology, const char \*restrict buffer, int size)
- struct `hwloc_topology_support` \* `hwloc_topology_get_support` (`hwloc_topology_t` restrict topology)

### 13.7.1 Detailed Description

These functions can optionally be called between `hwloc_topology_init()` and `hwloc_topology_load()` to configure how the detection should be performed, e.g. to ignore some objects types, define a synthetic topology, etc.

If none of them is called, the default is to detect all the objects of the machine that the caller is allowed to access.

This default behavior may also be modified through environment variables if the application did not modify it already. Setting `HWLOC_XMLFILE` in the environment enforces the discovery from a XML file as if `hwloc_topology_set_xml()` had been

called. HWLOC\_FSRROOT switches to reading the topology from the specified Linux filesystem root as if `hwloc_topology_set_fsroot()` had been called. Finally, HWLOC\_THISSYSTEM enforces the return value of `hwloc_topology_is_thissystem()`.

### 13.7.2 Enumeration Type Documentation

#### 13.7.2.1 `enum hwloc_topology_flags_e`

Flags to be set onto a topology context before load.

Flags should be given to `hwloc_topology_set_flags()`.

Enumerator:

**HWLOC\_TOPOLOGY\_FLAG\_WHOLE\_SYSTEM** Detect the whole system, ignore reservations and offline settings. Gather all resources, even if some were disabled by the administrator. For instance, ignore Linux Cpusets and gather all processors and memory nodes, and ignore the fact that some resources may be offline.

**HWLOC\_TOPOLOGY\_FLAG\_IS\_THISSYSTEM** Assume that the selected backend provides the topology for the system on which we are running. This forces `hwloc_topology_is_thissystem` to return 1, i.e. makes `hwloc` assume that the selected backend provides the topology for the system on which we are running, even if it is not the OS-specific backend but the XML backend for instance. This means making the binding functions actually call the OS-specific system calls and really do binding, while the XML backend would otherwise provide empty hooks just returning success.

Setting the environment variable `HWLOC_THISSYSTEM` may also result in the same behavior.

This can be used for efficiency reasons to first detect the topology once, save it to an XML file, and quickly reload it later through the XML backend, but still having binding functions actually do bind.

### 13.7.3 Function Documentation

#### 13.7.3.1 `struct hwloc_topology_support* hwloc_topology_get_support ( hwloc_topology_t restrict topology )` [read]

Retrieve the topology support.

#### 13.7.3.2 `int hwloc_topology_ignore_all_keep_structure ( hwloc_topology_t topology )`

Ignore all objects that do not bring any structure.

Ignore all objects that do not bring any structure: Each ignored object should have a single children or be the only child of its parent.

### 13.7.3.3 `int hwloc_topology_ignore_type ( hwloc_topology_t topology, hwloc_obj_type_t type )`

Ignore an object type.

Ignore all objects from the given type. The bottom-level type `HWLOC_OBJ_PU` may not be ignored. The top-level object of the hierarchy will never be ignored, even if this function succeeds.

### 13.7.3.4 `int hwloc_topology_ignore_type_keep_structure ( hwloc_topology_t topology, hwloc_obj_type_t type )`

Ignore an object type if it does not bring any structure.

Ignore all objects from the given type as long as they do not bring any structure: Each ignored object should have a single children or be the only child of its parent. The bottom-level type `HWLOC_OBJ_PU` may not be ignored.

### 13.7.3.5 `int hwloc_topology_set_flags ( hwloc_topology_t topology, unsigned long flags )`

Set OR'ed flags to non-yet-loaded topology.

Set a OR'ed set of `hwloc_topology_flags_e` onto a topology that was not yet loaded.

### 13.7.3.6 `int hwloc_topology_set_fsroot ( hwloc_topology_t restrict topology, const char *restrict fsroot_path )`

Change the file-system root path when building the topology from sysfs/procfs.

On Linux system, use sysfs and procfs files as if they were mounted on the given `fsroot_path` instead of the main file-system root. Setting the environment variable `HWLOC_FSROOT` may also result in this behavior. Not using the main file-system root causes [hwloc\\_topology\\_is\\_thissystem\(\)](#) to return 0.

#### Note

For conveniency, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, the `HWLOC_TOPOLOGY_FLAG_IS_THISSYSTEM` has to be set to assert that the loaded file is really the underlying system.

### 13.7.3.7 `int hwloc_topology_set_pid ( hwloc_topology_t restrict topology, hwloc_pid_t pid )`

Change which pid the topology is viewed from.

On some systems, processes may have different views of the machine, for instance the set of allowed CPUs. By default, hwloc exposes the view from the current process. Calling [hwloc\\_topology\\_set\\_pid\(\)](#) permits to make it expose the topology of the machine from the point of view of another process.

## Note

hwloc\_pid\_t is pid\_t on unix platforms, and HANDLE on native Windows platforms  
 -1 is returned and errno is set to ENOSYS on platforms that do not support this feature.

### 13.7.3.8 int hwloc\_topology\_set\_synthetic ( hwloc\_topology\_t restrict topology, const char \*restrict description )

Enable synthetic topology.

Gather topology information from the given *description* which should be a comma separated string of numbers describing the arity of each level. Each number may be prefixed with a type and a colon to enforce the type of a level. If only some level types are enforced, hwloc will try to choose the other types according to usual topologies, but it may fail and you may have to specify more level types manually.

## Note

For conveniency, this backend provides empty binding hooks which just return success.

### 13.7.3.9 int hwloc\_topology\_set\_xml ( hwloc\_topology\_t restrict topology, const char \*restrict xmlpath )

Enable XML-file based topology.

Gather topology information from the XML file given at *xmlpath*. Setting the environment variable HWLOC\_XMLFILE may also result in this behavior. This file may have been generated earlier with `lstopo file.xml`.

## Note

For conveniency, this backend provides empty binding hooks which just return success. To have hwloc still actually call OS-specific hooks, the HWLOC\_TOPOLOGY\_FLAG\_IS\_THISSYSTEM has to be set to assert that the loaded file is really the underlying system.

### 13.7.3.10 int hwloc\_topology\_set\_xmlbuffer ( hwloc\_topology\_t restrict topology, const char \*restrict buffer, int size )

Enable XML based topology using a memory buffer instead of a file.

Gather topology information from the XML memory buffer given at *buffer* and of length *length*.



## 13.8 Tinker with topologies.

### Functions

- void `hwloc_topology_export_xml` (`hwloc_topology_t` topology, const char \*xmlpath)
- void `hwloc_topology_export_xmlbuffer` (`hwloc_topology_t` topology, char \*\*xmlbuffer, int \*buflen)
- `hwloc_obj_t` `hwloc_topology_insert_misc_object_by_cpuset` (`hwloc_topology_t` topology, `hwloc_const_cpuset_t` cpuset, const char \*name)
- `hwloc_obj_t` `hwloc_topology_insert_misc_object_by_parent` (`hwloc_topology_t` topology, `hwloc_obj_t` parent, const char \*name)

### 13.8.1 Function Documentation

**13.8.1.1** void `hwloc_topology_export_xml` ( `hwloc_topology_t` *topology*, const char \* *xmlpath* )

Export the topology into an XML file.

This file may be loaded later through `hwloc_topology_set_xml()`.

**13.8.1.2** void `hwloc_topology_export_xmlbuffer` ( `hwloc_topology_t` *topology*, char \*\* *xmlbuffer*, int \* *buflen* )

Export the topology into a newly-allocated XML memory buffer.

`xmlbuffer` is allocated by the callee and should be freed with `xmlFree` later in the caller.

This memory buffer may be loaded later through `hwloc_topology_set_xmlbuffer()`.

**13.8.1.3** `hwloc_obj_t` `hwloc_topology_insert_misc_object_by_cpuset` ( `hwloc_topology_t` *topology*, `hwloc_const_cpuset_t` *cpuset*, const char \* *name* )

Add a MISC object to the topology.

A new MISC object will be created and inserted into the topology at the position given by bitmap `cpuset`.

`cpuset` and `name` will be copied.

Returns

the newly-created object

**13.8.1.4** `hwloc_obj_t` `hwloc_topology_insert_misc_object_by_parent` ( `hwloc_topology_t` *topology*, `hwloc_obj_t` *parent*, const char \* *name* )

Add a MISC object to the topology.

A new MISC object will be created and inserted into the topology at the position given by parent.

name will be copied.

Returns

the newly-created object

## 13.9 Get some Topology Information

### Enumerations

- enum `hwloc_get_type_depth_e` { `HWLOC_TYPE_DEPTH_UNKNOWN`, `HWLOC_TYPE_DEPTH_MULTIPLE` }

### Functions

- unsigned `hwloc_topology_get_depth` (`hwloc_topology_t` restrict topology)
- int `hwloc_get_type_depth` (`hwloc_topology_t` topology, `hwloc_obj_type_t` type)
- `hwloc_obj_type_t` `hwloc_get_depth_type` (`hwloc_topology_t` topology, unsigned depth)
- unsigned `hwloc_get_nobjs_by_depth` (`hwloc_topology_t` topology, unsigned depth)
- static inline int `hwloc_get_nobjs_by_type` (`hwloc_topology_t` topology, `hwloc_obj_type_t` type)
- int `hwloc_topology_is_thissystem` (`hwloc_topology_t` restrict topology)

### 13.9.1 Enumeration Type Documentation

#### 13.9.1.1 enum `hwloc_get_type_depth_e`

Enumerator:

`HWLOC_TYPE_DEPTH_UNKNOWN` No object of given type exists in the topology.

`HWLOC_TYPE_DEPTH_MULTIPLE` Objects of given type exist at different depth in the topology.

### 13.9.2 Function Documentation

#### 13.9.2.1 `hwloc_obj_type_t` `hwloc_get_depth_type` ( `hwloc_topology_t` *topology*, unsigned *depth* )

Returns the type of objects at depth `depth`.

Returns

-1 if depth `depth` does not exist.

### 13.9.2.2 `unsigned hwloc_get_nbobjs_by_depth ( hwloc_topology_t topology, unsigned depth )`

Returns the width of level at depth `depth`.

### 13.9.2.3 `static inline int hwloc_get_nbobjs_by_type ( hwloc_topology_t topology, hwloc_obj_type_t type ) [static]`

Returns the width of level type `type`.

If no object for that type exists, 0 is returned. If there are several levels with objects of that type, -1 is returned.

### 13.9.2.4 `int hwloc_get_type_depth ( hwloc_topology_t topology, hwloc_obj_type_t type )`

Returns the depth of objects of type `type`.

If no object of this type is present on the underlying architecture, or if the OS doesn't provide this kind of information, the function returns `HWLOC_TYPE_DEPTH_UNKNOWN`.

If type is absent but a similar type is acceptable, see also [hwloc\\_get\\_type\\_or\\_below\\_depth\(\)](#) and [hwloc\\_get\\_type\\_or\\_above\\_depth\(\)](#).

### 13.9.2.5 `unsigned hwloc_topology_get_depth ( hwloc_topology_t restrict topology )`

Get the depth of the hierarchical tree of objects.

This is the depth of `HWLOC_OBJ_PU` objects plus one.

### 13.9.2.6 `int hwloc_topology_is_thissystem ( hwloc_topology_t restrict topology )`

Does the topology context come from this system?

Returns

1 if this topology context was built using the system running this program.  
0 instead (for instance if using another file-system root, a XML topology file, or a synthetic topology).

## 13.10 Retrieve Objects

### Functions

- [hwloc\\_obj\\_t hwloc\\_get\\_obj\\_by\\_depth \(hwloc\\_topology\\_t topology, unsigned depth, unsigned idx\)](#)
- `static inline hwloc_obj_t hwloc_get_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, unsigned idx)`

### 13.10.1 Function Documentation

**13.10.1.1** `hwloc_obj_t hwloc_get_obj_by_depth ( hwloc_topology_t topology, unsigned depth, unsigned idx )`

Returns the topology object at index `index` from depth `depth`.

**13.10.1.2** `static inline hwloc_obj_t hwloc_get_obj_by_type ( hwloc_topology_t topology, hwloc_obj_type_t type, unsigned idx ) [static]`

Returns the topology object at index `index` with type `type`.

If no object for that type exists, `NULL` is returned. If there are several levels with objects of that type, `NULL` is returned and the caller may fallback to [hwloc\\_get\\_obj\\_by\\_depth\(\)](#).

## 13.11 Object/String Conversion

### Functions

- `const char * hwloc_obj_type_string (hwloc_obj_type_t type)`
- `hwloc_obj_type_t hwloc_obj_type_of_string (const char *string)`
- `int hwloc_obj_type_snprintf (char *restrict string, size_t size, hwloc_obj_t obj, int verbose)`
- `int hwloc_obj_attr_snprintf (char *restrict string, size_t size, hwloc_obj_t obj, const char *restrict separator, int verbose)`
- `int hwloc_obj_snprintf (char *restrict string, size_t size, hwloc_topology_t topology, hwloc_obj_t obj, const char *restrict indexprefix, int verbose)`
- `int hwloc_obj_cpuset_snprintf (char *restrict str, size_t size, size_t nobj, const hwloc_obj_t *restrict objs)`
- `static inline char * hwloc_obj_get_info_by_name (hwloc_obj_t obj, const char *name)`

### 13.11.1 Function Documentation

**13.11.1.1** `int hwloc_obj_attr_snprintf ( char *restrict string, size_t size, hwloc_obj_t obj, const char *restrict separator, int verbose )`

Stringify the attributes of a given topology object into a human-readable form.

Attribute values are separated by `separator`.

Only the major attributes are printed in non-verbose mode.

If `size` is 0, `string` may safely be `NULL`.

**Returns**

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

**13.11.1.2** `int hwloc_obj_cpuset_sprintf ( char *restrict str, size_t size, size_t nobj, const hwloc_obj_t *restrict objs )`

Stringify the cpuset containing a set of objects.

If `size` is 0, `string` may safely be `NULL`.

**Returns**

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

**13.11.1.3** `static inline char* hwloc_obj_get_info_by_name ( hwloc_obj_t obj, const char * name ) [static]`

Search the given key name in object infos and return the corresponding value.

**Returns**

`NULL` if no such key exists.

**13.11.1.4** `int hwloc_obj_snprintf ( char *restrict string, size_t size, hwloc_topology_t topology, hwloc_obj_t obj, const char *restrict indexprefix, int verbose )`

Stringify a given topology object into a human-readable form.

**Note**

This function is deprecated in favor of [hwloc\\_obj\\_type\\_snprintf\(\)](#) and [hwloc\\_obj\\_attr\\_snprintf\(\)](#) since it is not very flexible and only prints physical/OS indexes.

Fill `string` up to `size` characters with the description of topology object `obj` in topology `topology`.

If `verbose` is set, a longer description is used. Otherwise a short description is used.

`indexprefix` is used to prefix the `os_index` attribute number of the object in the description. If `NULL`, the `#` character is used.

If `size` is 0, `string` may safely be `NULL`.

**Returns**

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

### 13.11.1.5 hwloc\_obj\_type\_t hwloc\_obj\_type\_of\_string ( const char \* *string* )

Return an object type from the string.

Returns

-1 if unrecognized.

### 13.11.1.6 int hwloc\_obj\_type\_snprintf ( char \*restrict *string*, size\_t *size*, hwloc\_obj\_t *obj*, int *verbose* )

Stringify the type of a given topology object into a human-readable form.

It differs from [hwloc\\_obj\\_type\\_string\(\)](#) because it prints type attributes such as cache depth.

If *size* is 0, *string* may safely be NULL.

Returns

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

### 13.11.1.7 const char\* hwloc\_obj\_type\_string ( hwloc\_obj\_type\_t *type* ) const

Return a stringified topology object type.

## 13.12 CPU binding

### Enumerations

- enum [hwloc\\_cpubind\\_flags\\_t](#) { [HWLOC\\_CPUBIND\\_PROCESS](#), [HWLOC\\_CPUBIND\\_THREAD](#), [HWLOC\\_CPUBIND\\_STRICT](#), [HWLOC\\_CPUBIND\\_NOMEMBIND](#) }

### Functions

- int [hwloc\\_set\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_const\\_cpuset\\_t](#) set, int flags)
- int [hwloc\\_get\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_cpuset\\_t](#) set, int flags)
- int [hwloc\\_set\\_proc\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_pid\\_t](#) pid, [hwloc\\_const\\_cpuset\\_t](#) set, int flags)
- int [hwloc\\_get\\_proc\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_pid\\_t](#) pid, [hwloc\\_cpuset\\_t](#) set, int flags)
- int [hwloc\\_set\\_thread\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_thread\\_t](#) tid, [hwloc\\_const\\_cpuset\\_t](#) set, int flags)
- int [hwloc\\_get\\_thread\\_cpubind](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_thread\\_t](#) tid, [hwloc\\_cpuset\\_t](#) set, int flags)

### 13.12.1 Detailed Description

It is often useful to call [hwloc\\_bitmap\\_singlify\(\)](#) first so that a single CPU remains in the set. This way, the process will not even migrate between different CPUs. Some operating systems also only support that kind of binding.

#### Note

Some operating systems do not provide all hwloc-supported mechanisms to bind processes, threads, etc. and the corresponding binding functions may fail. -1 is returned and `errno` is set to `ENOSYS` when it is not possible to bind the requested kind of object processes/threads. `errno` is set to `EXDEV` when the requested cpuset can not be enforced (e.g. some systems only allow one CPU, and some other systems only allow one NUMA node).

The most portable version that should be preferred over the others, whenever possible, is

```
hwloc_set_cpubind(topology, set, 0),
```

as it just binds the current program, assuming it is single-threaded, or

```
hwloc_set_cpubind(topology, set, HWLOC_CPUBIND_THREAD),
```

which binds the current thread of the current program (which may be multithreaded).

#### Note

To unbind, just call the binding function with either a full cpuset or a cpuset equal to the system cpuset.

On some operating systems, CPU binding may have effects on memory binding, see [HWLOC\\_CPUBIND\\_NOMEMBIND](#)

### 13.12.2 Enumeration Type Documentation

#### 13.12.2.1 enum hwloc\_cpubind\_flags\_t

Process/Thread binding flags.

These bit flags can be used to refine the binding policy.

The default (0) is to bind the current process, assumed to be single-threaded, in a non-strict way. This is the most portable way to bind as all operating systems usually provide it.

#### Note

Not all systems support all kinds of binding. See the "Detailed Description" section of [CPU binding](#) for a description of errors that can occur.

Enumerator:

**HWLOC\_CPUBIND\_PROCESS** Bind all threads of the current (possibly) multithreaded process.

**HWLOC\_CPUBIND\_THREAD** Bind current thread of current process.

**HWLOC\_CPUBIND\_STRICT** Request for strict binding from the OS. By default, when the designated CPUs are all busy while other CPUs are idle, operating systems may execute the thread/process on those other CPUs instead of the designated CPUs, to let them progress anyway. Strict binding means that the thread/process will *\_never\_* execute on other cpus than the designated CPUs, even when those are busy with other tasks and other CPUs are idle.

Note

Depending on the operating system, strict binding may not be possible (e.g., the OS does not implement it) or not allowed (e.g., for administrative reasons), and the function will fail in that case.

When retrieving the binding of a process, this flag checks whether all its threads actually have the same binding. If the flag is not given, the binding of each thread will be accumulated.

Note

This flag is meaningless when retrieving the binding of a thread.

**HWLOC\_CPUBIND\_NOMEMBIND** Avoid any effect on memory binding. On some operating systems, some CPU binding function would also bind the memory on the corresponding NUMA node. It is often not a problem for the application, but if it is, setting this flag will make hwloc avoid using OS functions that would also bind memory. This will however reduce the support of CPU bindings, i.e. potentially return -1 with `errno` set to `ENOSYS` in some cases.

### 13.12.3 Function Documentation

**13.12.3.1** `int hwloc_get_cpupbind ( hwloc_topology_t topology, hwloc_cpuset_t set, int flags )`

Get current process or thread binding.

**13.12.3.2** `int hwloc_get_proc_cpupbind ( hwloc_topology_t topology, hwloc_pid_t pid, hwloc_cpuset_t set, int flags )`

Get the current binding of process `pid`.

Note

`hwloc_pid_t` is `pid_t` on unix platforms, and `HANDLE` on native Windows platforms

**HWLOC\_CPUBIND\_THREAD** can not be used in `flags`.



**13.12.3.3** `int hwloc_get_thread_cpubind ( hwloc_topology_t topology, hwloc_thread_t tid,  
hwloc_cpuset_t set, int flags )`

Get the current binding of thread `tid`.

Note

`hwloc_thread_t` is `pthread_t` on unix platforms, and `HANDLE` on native Windows platforms  
`HWLOC_CPUBIND_PROCESS` can not be used in `flags`.

**13.12.3.4** `int hwloc_set_cpubind ( hwloc_topology_t topology, hwloc_const_cpuset_t  
set, int flags )`

Bind current process or thread on cpus given in bitmap `set`.

Returns

-1 with `errno` set to `ENOSYS` if the action is not supported  
-1 with `errno` set to `EXDEV` if the binding cannot be enforced

**13.12.3.5** `int hwloc_set_proc_cpubind ( hwloc_topology_t topology, hwloc_pid_t pid,  
hwloc_const_cpuset_t set, int flags )`

Bind a process `pid` on cpus given in bitmap `set`.

Note

`hwloc_pid_t` is `pid_t` on unix platforms, and `HANDLE` on native Windows platforms  
`HWLOC_CPUBIND_THREAD` can not be used in `flags`.

**13.12.3.6** `int hwloc_set_thread_cpubind ( hwloc_topology_t topology, hwloc_thread_t tid,  
hwloc_const_cpuset_t set, int flags )`

Bind a thread `tid` on cpus given in bitmap `set`.

Note

`hwloc_thread_t` is `pthread_t` on unix platforms, and `HANDLE` on native Windows platforms  
`HWLOC_CPUBIND_PROCESS` can not be used in `flags`.

## 13.13 Memory binding

### Enumerations

- enum `hwloc_membind_policy_t` {  
`HWLOC_MEMBIND_DEFAULT`, `HWLOC_MEMBIND_FIRSTTOUCH`, `HWLOC_MEMBIND_BIND`, `HWLOC_MEMBIND_INTERLEAVE`,  
`HWLOC_MEMBIND_REPLICATE`, `HWLOC_MEMBIND_NEXTTOUCH`, `HWLOC_MEMBIND_MIXED` }
- enum `hwloc_membind_flags_t` {  
`HWLOC_MEMBIND_PROCESS`, `HWLOC_MEMBIND_THREAD`, `HWLOC_MEMBIND_STRICT`, `HWLOC_MEMBIND_MIGRATE`,  
`HWLOC_MEMBIND_NOCPUBIND` }

### Functions

- int `hwloc_set_membind_nodest` (`hwloc_topology_t` topology, `hwloc_const_nodest_t` nodest, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_set_membind` (`hwloc_topology_t` topology, `hwloc_const_cpuset_t` cpuset, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_get_membind_nodest` (`hwloc_topology_t` topology, `hwloc_nodest_t` nodest, `hwloc_membind_policy_t` \*policy, int flags)
- int `hwloc_get_membind` (`hwloc_topology_t` topology, `hwloc_cpuset_t` cpuset, `hwloc_membind_policy_t` \*policy, int flags)
- int `hwloc_set_proc_membind_nodest` (`hwloc_topology_t` topology, `hwloc_pid_t` pid, `hwloc_const_nodest_t` nodest, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_set_proc_membind` (`hwloc_topology_t` topology, `hwloc_pid_t` pid, `hwloc_const_cpuset_t` cpuset, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_get_proc_membind_nodest` (`hwloc_topology_t` topology, `hwloc_pid_t` pid, `hwloc_nodest_t` nodest, `hwloc_membind_policy_t` \*policy, int flags)
- int `hwloc_get_proc_membind` (`hwloc_topology_t` topology, `hwloc_pid_t` pid, `hwloc_cpuset_t` cpuset, `hwloc_membind_policy_t` \*policy, int flags)
- int `hwloc_set_area_membind_nodest` (`hwloc_topology_t` topology, const void \*addr, size\_t len, `hwloc_const_nodest_t` nodest, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_set_area_membind` (`hwloc_topology_t` topology, const void \*addr, size\_t len, `hwloc_const_cpuset_t` cpuset, `hwloc_membind_policy_t` policy, int flags)
- int `hwloc_get_area_membind_nodest` (`hwloc_topology_t` topology, const void \*addr, size\_t len, `hwloc_nodest_t` nodest, `hwloc_membind_policy_t` \*policy, int flags)
- int `hwloc_get_area_membind` (`hwloc_topology_t` topology, const void \*addr, size\_t len, `hwloc_cpuset_t` cpuset, `hwloc_membind_policy_t` \*policy, int flags)
- void \* `hwloc_alloc` (`hwloc_topology_t` topology, size\_t len)
- void \* `hwloc_alloc_membind_nodest` (`hwloc_topology_t` topology, size\_t len, `hwloc_const_nodest_t` nodest, `hwloc_membind_policy_t` policy, int flags)

- void \* [hwloc\\_alloc\\_mbind](#) ([hwloc\\_topology\\_t](#) topology, size\_t len, [hwloc\\_const\\_cpuset\\_t](#) cpuset, [hwloc\\_mbind\\_policy\\_t](#) policy, int flags)
- int [hwloc\\_free](#) ([hwloc\\_topology\\_t](#) topology, void \*addr, size\_t len)

### 13.13.1 Detailed Description

#### Note

Not all operating systems support all ways to bind existing allocated memory (e.g., migration), future memory allocation, explicit memory allocation, etc. Using a binding flag or policy that is not supported by the underlying OS will cause hwloc's binding functions to fail and return -1. `errno` will be set to `ENOSYS` when the system does support the specified action or policy (e.g., some systems only allow binding memory on a per-thread basis, whereas other systems only allow binding memory for all threads in a process). `errno` will be set to `EXDEV` when the requested cpuset can not be enforced (e.g., some systems only allow binding memory to a single NUMA node).

The most portable form that should be preferred over the others whenever possible is as follows:

```
hwloc_alloc_mbind_policy(topology, size, set,
                        HWLOC_MEMBIND_DEFAULT, 0);
```

This will allocate some memory hopefully bound to the specified set. To do so, hwloc will possibly have to change the current memory binding policy in order to actually get the memory bound, if the OS does not provide any other way to simply allocate bound memory without changing the policy for all allocations. That is the difference with [hwloc\\_alloc\\_mbind\(\)](#), which will never change the current memory binding policy. Note that since `HWLOC_MEMBIND_STRICT` was not specified, failures to bind will not be reported -- generally, only memory allocation failures will be reported (e.g., even a plain `malloc()` would have failed with `ENOMEM`).

Each hwloc memory binding function is available in two forms: one that takes a CPU set argument and another that takes a NUMA memory node set argument (see [Object sets](#) ([hwloc\\_cpuset\\_t](#) and [hwloc\\_nodeset\\_t](#)) and [The bitmap API](#) for a discussion of CPU sets and NUMA memory node sets). The names of the latter form end with `_nodeset`. It is also possible to convert between CPU set and node set using [hwloc\\_cpuset\\_to\\_nodeset\(\)](#) or [hwloc\\_cpuset\\_from\\_nodeset\(\)](#).

#### Note

On some operating systems, memory binding affects the CPU binding; see [HWLOC\\_MEMBIND\\_NOCPUBIND](#)

### 13.13.2 Enumeration Type Documentation

#### 13.13.2.1 enum hwloc\_mbind\_flags\_t

Memory binding flags.

These flags can be used to refine the binding policy. All flags can be logically OR'ed together with the exception of `HWLOC_MEMBIND_PROCESS` and `HWLOC_MEMBIND_THREAD`; these two flags are mutually exclusive.

#### Note

Not all systems support all kinds of binding. See the "Detailed Description" section of [Memory binding](#) for a description of errors that can occur.

#### Enumerator:

- `HWLOC_MEMBIND_PROCESS` Set policy for all threads of the specified (possibly multithreaded) process. This flag is mutually exclusive with `HWLOC_MEMBIND_THREAD`.
- `HWLOC_MEMBIND_THREAD` Set policy for a specific thread of the current process. This flag is mutually exclusive with `HWLOC_MEMBIND_PROCESS`.
- `HWLOC_MEMBIND_STRICT` Request strict binding from the OS. The function will fail if the binding can not be guaranteed / completely enforced. This flag has slightly different meanings depending on which function it is used with.
- `HWLOC_MEMBIND_MIGRATE` Migrate existing allocated memory. If the memory cannot be migrated and the `HWLOC_MEMBIND_STRICT` flag is passed, an error will be returned.
- `HWLOC_MEMBIND_NOCPUBIND` Avoid any effect on CPU binding. On some operating systems, some underlying memory binding functions also bind the application to the corresponding CPU(s). Using this flag will cause `hwloc` to avoid using OS functions that could potentially affect CPU bindings. Note, however, that using `NOCPUBIND` may reduce `hwloc`'s overall memory binding support. Specifically: some of `hwloc`'s memory binding functions may fail with `errno` set to `ENOSYS` when used with `NOCPUBIND`.

#### 13.13.2.2 enum hwloc\_membind\_policy\_t

Memory binding policy.

These constants can be used to choose the binding policy. Only one policy can be used at a time (i.e., the values cannot be OR'ed together).

#### Note

Not all systems support all kinds of binding. See the "Detailed Description" section of [Memory binding](#) for a description of errors that can occur.

#### Enumerator:

- `HWLOC_MEMBIND_DEFAULT` Reset the memory allocation policy to the system default.

**HWLOC\_MEMBIND\_FIRSTTOUCH** Allocate memory but do not immediately bind it to a specific locality. Instead, each page in the allocation is bound only when it is first touched. Pages are individually bound to the local NUMA node of the first thread that touches it.

**HWLOC\_MEMBIND\_BIND** Allocate memory on the specified nodes.

**HWLOC\_MEMBIND\_INTERLEAVE** Allocate memory on the given nodes in an interleaved / round-robin manner. The precise layout of the memory across multiple NUMA nodes is OS/system specific. Interleaving can be useful when threads distributed across the specified NUMA nodes will all be accessing the whole memory range concurrently, since the interleave will then balance the memory references.

**HWLOC\_MEMBIND\_REPLICATE** Replicate memory on the given nodes; reads from this memory will attempt to be serviced from the NUMA node local to the reading thread. Replicating can be useful when multiple threads from the specified NUMA nodes will be sharing the same read-only data. This policy can only be used with existing memory allocations (i.e., the `hwloc_set_membind*()` functions); it cannot be used with functions that allocate new memory (i.e., the `hwloc_alloc*()` functions).

**HWLOC\_MEMBIND\_NEXTTOUCH** For each page bound with this policy, by next time it is touched (and next time only), it is moved from its current location to the local NUMA node of the thread where the memory reference occurred (if it needs to be moved at all).

**HWLOC\_MEMBIND\_MIXED** Returned by `hwloc_get_membind*()` functions when multiple threads or parts of a memory area have differing memory binding policies.

### 13.13.3 Function Documentation

#### 13.13.3.1 `void* hwloc_alloc ( hwloc_topology_t topology, size_t len )`

Allocate some memory.

This is equivalent to `malloc()`, except that it tries to allocate page-aligned memory from the OS.

Note

The allocated memory should be freed with [hwloc\\_free\(\)](#).

#### 13.13.3.2 `void* hwloc_alloc_membind ( hwloc_topology_t topology, size_t len, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags )`

Allocate some memory on memory nodes near the given cpuset `cpuset`.

Returns

-1 with `errno` set to `ENOSYS` if the action is not supported and `HWLOC_MEMBIND_STRICT` is given

-1 with `errno` set to `EXDEV` if the binding cannot be enforced and `HWLOC_MEMBIND_STRICT` is given

Note

The allocated memory should be freed with `hwloc_free()`.

**13.13.3.3** `void* hwloc_alloc_membind_nodeset ( hwloc_topology_t topology, size_t len, hwloc_const_nodeset_t nodeset, hwloc_membind_policy_t policy, int flags )`

Allocate some memory on the given nodeset `nodeset`.

Returns

-1 with `errno` set to `ENOSYS` if the action is not supported and `HWLOC_MEMBIND_STRICT` is given  
 -1 with `errno` set to `EXDEV` if the binding cannot be enforced and `HWLOC_MEMBIND_STRICT` is given

Note

The allocated memory should be freed with `hwloc_free()`.

**13.13.3.4** `int hwloc_free ( hwloc_topology_t topology, void * addr, size_t len )`

Free memory that was previously allocated by `hwloc_alloc()` or `hwloc_alloc_membind()`.

**13.13.3.5** `int hwloc_get_area_membind ( hwloc_topology_t topology, const void * addr, size_t len, hwloc_cpuset_t cpuset, hwloc_membind_policy_t * policy, int flags )`

Query the CPUs near the NUMA node(s) and binding policy of the memory identified by (`addr`, `len`).

This function has two output parameters: `cpuset` and `policy`. The values returned in these parameters depend on both the `flags` passed in and the memory binding policies and nodesets of the pages in the address range.

If `HWLOC_MEMBIND_STRICT` is specified, the target pages are first checked to see if they all have the same memory binding policy and nodeset. If they do not, -1 is returned and `errno` is set to `EXDEV`. If they are identical across all pages, the policy is returned in `policy`. `cpuset` is set to the union of CPUs near the NUMA node(s) in the nodeset.

If `HWLOC_MEMBIND_STRICT` is not specified, the union of all NUMA node(s) containing pages in the address range is calculated. `cpuset` is then set to the CPUs near the NUMA node(s) in this union. If all pages in the target have the same policy, it is returned in `policy`. Otherwise, `policy` is set to `HWLOC_MEMBIND_MIXED`.

If any other flags are specified, -1 is returned and `errno` is set to `EINVAL`.

**13.13.3.6** `int hwloc_get_area_mbind_nodeset ( hwloc_topology_t topology, const void *  
addr, size_t len, hwloc_nodeset_t nodeset, hwloc_mbind_policy_t *  
policy, int flags )`

Query the NUMA node(s) and binding policy of the memory identified by (*addr*, *len*).

This function has two output parameters: *nodeset* and *policy*. The values returned in these parameters depend on both the *flags* passed in and the memory binding policies and nodesets of the pages in the address range.

If `HWLOC_MEMBIND_STRICT` is specified, the target pages are first checked to see if they all have the same memory binding policy and nodeset. If they do not, -1 is returned and *errno* is set to `EXDEV`. If they are identical across all pages, the *nodeset* and *policy* are returned in *nodeset* and *policy*, respectively.

If `HWLOC_MEMBIND_STRICT` is not specified, *nodeset* is set to the union of all NUMA node(s) containing pages in the address range. If all pages in the target have the same policy, it is returned in *policy*. Otherwise, *policy* is set to `HWLOC_MEMBIND_MIXED`.

If any other flags are specified, -1 is returned and *errno* is set to `EINVAL`.

**13.13.3.7** `int hwloc_get_mbind ( hwloc_topology_t topology, hwloc_cpuset_t cpuset,  
hwloc_mbind_policy_t * policy, int flags )`

Query the default memory binding policy and locality of the current process or thread (the locality is returned in *cpuset* as CPUs near the locality's actual NUMA node(s)).

This function has two output parameters: *cpuset* and *policy*. The values returned in these parameters depend on both the *flags* passed in and the current memory binding policies and nodesets in the queried target.

Passing the `HWLOC_MEMBIND_PROCESS` flag specifies that the query target is the current policies and nodesets for all the threads in the current process. Passing `HWLOC_MEMBIND_THREAD` specifies that the query target is the current policy and nodeset for only the thread invoking this function.

If neither of these flags are passed (which is the most portable method), the process is assumed to be single threaded. This allows *hwloc* to use either process-based OS functions or thread-based OS functions, depending on which are available.

`HWLOC_MEMBIND_STRICT` is only meaningful when `HWLOC_MEMBIND_PROCESS` is also specified. In this case, *hwloc* will check the default memory policies and nodesets for all threads in the process. If they are not identical, -1 is returned and *errno* is set to `EXDEV`. If they are identical, the policy is returned in *policy*. *cpuset* is set to the union of CPUs near the NUMA node(s) in the *nodeset*.

Otherwise, if `HWLOC_MEMBIND_PROCESS` is specified (and `HWLOC_MEMBIND_STRICT` is *not* specified), the default nodeset from each thread is logically OR'ed together. *cpuset* is set to the union of CPUs near the NUMA node(s) in the resulting nodeset. If all threads' default policies are the same, *policy* is set to that policy. If they are different, *policy* is set to `HWLOC_MEMBIND_MIXED`.

In the `HWLOC_MEMBIND_THREAD` case (or when neither `HWLOC_MEMBIND_PROCESS` or `HWLOC_MEMBIND_THREAD` is specified), there is only one nodeset and policy. The policy is returned in `policy`; `cpuset` is set to the union of CPUs near the NUMA node(s) in the `nodeset`.

If any other flags are specified, -1 is returned and `errno` is set to `EINVAL`.

**13.13.3.8** `int hwloc_get_membind_nodeset( hwloc_topology_t topology, hwloc_nodeset_t nodeset, hwloc_membind_policy_t * policy, int flags )`

Query the default memory binding policy and locality of the current process or thread.

This function has two output parameters: `nodeset` and `policy`. The values returned in these parameters depend on both the `flags` passed in and the current memory binding policies and nodesets in the queried target.

Passing the `HWLOC_MEMBIND_PROCESS` flag specifies that the query target is the current policies and nodesets for all the threads in the current process. Passing `HWLOC_MEMBIND_THREAD` specifies that the query target is the current policy and nodeset for only the thread invoking this function.

If neither of these flags are passed (which is the most portable method), the process is assumed to be single threaded. This allows `hwloc` to use either process-based OS functions or thread-based OS functions, depending on which are available.

`HWLOC_MEMBIND_STRICT` is only meaningful when `HWLOC_MEMBIND_PROCESS` is also specified. In this case, `hwloc` will check the default memory policies and nodesets for all threads in the process. If they are not identical, -1 is returned and `errno` is set to `EXDEV`. If they are identical, the values are returned in `nodeset` and `policy`.

Otherwise, if `HWLOC_MEMBIND_PROCESS` is specified (and `HWLOC_MEMBIND_STRICT` is *not* specified), `nodeset` is set to the logical OR of all threads' default nodeset. If all threads' default policies are the same, `policy` is set to that policy. If they are different, `policy` is set to `HWLOC_MEMBIND_MIXED`.

In the `HWLOC_MEMBIND_THREAD` case (or when neither `HWLOC_MEMBIND_PROCESS` or `HWLOC_MEMBIND_THREAD` is specified), there is only one nodeset and policy; they are returned in `nodeset` and `policy`, respectively.

If any other flags are specified, -1 is returned and `errno` is set to `EINVAL`.

**13.13.3.9** `int hwloc_get_proc_membind( hwloc_topology_t topology, hwloc_pid_t pid, hwloc_cpuset_t cpuset, hwloc_membind_policy_t * policy, int flags )`

Query the default memory binding policy and locality of the specified process (the locality is returned in `cpuset` as CPUs near the locality's actual NUMA node(s)).

This function has two output parameters: `cpuset` and `policy`. The values returned in these parameters depend on both the `flags` passed in and the current memory binding policies and nodesets in the queried target.

Passing the `HWLOC_MEMBIND_PROCESS` flag specifies that the query target is the current policies and nodesets for all the threads in the specified process. If `HWLOC-`



MEMBIND\_PROCESS is not specified (which is the most portable method), the process is assumed to be single threaded. This allows hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

Note that it does not make sense to pass HWLOC\_MEMBIND\_THREAD to this function.

If HWLOC\_MEMBIND\_STRICT is specified, hwloc will check the default memory policies and nodesets for all threads in the specified process. If they are not identical, -1 is returned and errno is set to EXDEV. If they are identical, the policy is returned in *policy*. *cpuset* is set to the union of CPUs near the NUMA node(s) in the nodeset.

Otherwise, the default nodeset from each thread is logically OR'ed together. *cpuset* is set to the union of CPUs near the NUMA node(s) in the resulting nodeset. If all threads' default policies are the same, *policy* is set to that policy. If they are different, *policy* is set to HWLOC\_MEMBIND\_MIXED.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

**13.13.3.10** `int hwloc_get_proc_membind_nodeset ( hwloc_topology_t topology, hwloc_pid_t pid, hwloc_nodeset_t nodeset, hwloc_membind_policy_t* policy, int flags )`

Query the default memory binding policy and locality of the specified process.

This function has two output parameters: *nodeset* and *policy*. The values returned in these parameters depend on both the *flags* passed in and the current memory binding policies and nodesets in the queried target.

Passing the HWLOC\_MEMBIND\_PROCESS flag specifies that the query target is the current policies and nodesets for all the threads in the specified process. If HWLOC\_MEMBIND\_PROCESS is not specified (which is the most portable method), the process is assumed to be single threaded. This allows hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

Note that it does not make sense to pass HWLOC\_MEMBIND\_THREAD to this function.

If HWLOC\_MEMBIND\_STRICT is specified, hwloc will check the default memory policies and nodesets for all threads in the specified process. If they are not identical, -1 is returned and errno is set to EXDEV. If they are identical, the values are returned in *nodeset* and *policy*.

Otherwise, *nodeset* is set to the logical OR of all threads' default nodeset. If all threads' default policies are the same, *policy* is set to that policy. If they are different, *policy* is set to HWLOC\_MEMBIND\_MIXED.

If any other flags are specified, -1 is returned and errno is set to EINVAL.

**13.13.3.11** `int hwloc_set_area_mbind ( hwloc_topology_t topology, const void * addr, size_t len, hwloc_const_cpuset_t cpuset, hwloc_mbind_policy_t policy, int flags )`

Bind the already-allocated memory identified by (*addr*, *len*) to the NUMA node(s) near *cpuset*.

Returns

- 1 with *errno* set to ENOSYS if the action is not supported
- 1 with *errno* set to EXDEV if the binding cannot be enforced

**13.13.3.12** `int hwloc_set_area_mbind_nodest ( hwloc_topology_t topology, const void * addr, size_t len, hwloc_const_nodest_t nodest, hwloc_mbind_policy_t policy, int flags )`

Bind the already-allocated memory identified by (*addr*, *len*) to the NUMA node(s) in *nodest*.

Returns

- 1 with *errno* set to ENOSYS if the action is not supported
- 1 with *errno* set to EXDEV if the binding cannot be enforced

**13.13.3.13** `int hwloc_set_mbind ( hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, hwloc_mbind_policy_t policy, int flags )`

Set the default memory binding policy of the current process or thread to prefer the NUMA node(s) near the specified *cpuset*.

If neither HWLOC\_MEMBIND\_PROCESS nor HWLOC\_MEMBIND\_THREAD is specified, the current process is assumed to be single-threaded. This is the most portable form as it permits hwloc to use either process-based OS functions or thread-based OS functions, depending on which are available.

Returns

- 1 with *errno* set to ENOSYS if the action is not supported
- 1 with *errno* set to EXDEV if the binding cannot be enforced

**13.13.3.14** `int hwloc_set_mbind_nodest ( hwloc_topology_t topology, hwloc_const_nodest_t nodest, hwloc_mbind_policy_t policy, int flags )`

Set the default memory binding policy of the current process or thread to prefer the NUMA node(s) specified by *nodest*.

If neither `HWLOC_MEMBIND_PROCESS` nor `HWLOC_MEMBIND_THREAD` is specified, the current process is assumed to be single-threaded. This is the most portable form as it permits `hwloc` to use either process-based OS functions or thread-based OS functions, depending on which are available.

#### Returns

- 1 with `errno` set to `ENOSYS` if the action is not supported
- 1 with `errno` set to `EXDEV` if the binding cannot be enforced

**13.13.3.15** `int hwloc_set_proc_membind ( hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy, int flags )`

Set the default memory binding policy of the specified process to prefer the NUMA node(s) near the specified `cpuset`.

#### Returns

- 1 with `errno` set to `ENOSYS` if the action is not supported
- 1 with `errno` set to `EXDEV` if the binding cannot be enforced

**13.13.3.16** `int hwloc_set_proc_membind_nodeset ( hwloc_topology_t topology, hwloc_pid_t pid, hwloc_const_nodeset_t nodeset, hwloc_membind_policy_t policy, int flags )`

Set the default memory binding policy of the specified process to prefer the NUMA node(s) specified by `nodeset`.

#### Returns

- 1 with `errno` set to `ENOSYS` if the action is not supported
- 1 with `errno` set to `EXDEV` if the binding cannot be enforced

## 13.14 Object Type Helpers

### Functions

- static inline int `hwloc_get_type_or_below_depth (hwloc_topology_t topology, hwloc_obj_type_t type)`
- static inline int `hwloc_get_type_or_above_depth (hwloc_topology_t topology, hwloc_obj_type_t type)`

### 13.14.1 Function Documentation

**13.14.1.1** `static inline int hwloc_get_type_or_above_depth ( hwloc_topology_t topology,  
hwloc_obj_type_t type ) [static]`

Returns the depth of objects of type `type` or above.

If no object of this type is present on the underlying architecture, the function returns the depth of the first "present" object typically containing `type`.

**13.14.1.2** `static inline int hwloc_get_type_or_below_depth ( hwloc_topology_t topology,  
hwloc_obj_type_t type ) [static]`

Returns the depth of objects of type `type` or below.

If no object of this type is present on the underlying architecture, the function returns the depth of the first "present" object typically found inside `type`.

## 13.15 Basic Traversal Helpers

### Functions

- static inline `hwloc_obj_t hwloc_get_root_obj (hwloc_topology_t topology)`
- static inline `hwloc_obj_t hwloc_get_ancestor_obj_by_depth (hwloc_topology_t topology, unsigned depth, hwloc_obj_t obj)`
- static inline `hwloc_obj_t hwloc_get_ancestor_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t obj)`
- static inline `hwloc_obj_t hwloc_get_next_obj_by_depth (hwloc_topology_t topology, unsigned depth, hwloc_obj_t prev)`
- static inline `hwloc_obj_t hwloc_get_next_obj_by_type (hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t prev)`
- static inline `hwloc_obj_t hwloc_get_pu_obj_by_os_index (hwloc_topology_t topology, unsigned os_index)`
- static inline `hwloc_obj_t hwloc_get_next_child (hwloc_topology_t topology, hwloc_obj_t parent, hwloc_obj_t prev)`
- static inline `hwloc_obj_t hwloc_get_common_ancestor_obj (hwloc_topology_t topology, hwloc_obj_t obj1, hwloc_obj_t obj2)`
- static inline `int hwloc_obj_is_in_subtree (hwloc_topology_t topology, hwloc_obj_t obj, hwloc_obj_t subtree_root)`

### 13.15.1 Function Documentation

**13.15.1.1** `static inline hwloc_obj_t hwloc_get_ancestor_obj_by_depth ( hwloc_topology_t  
topology, unsigned depth, hwloc_obj_t obj ) [static]`

Returns the ancestor object of `obj` at depth `depth`.

**13.15.1.2** `static inline hwloc_obj_t hwloc_get_ancestor_obj_by_type ( hwloc_topology_t topology , hwloc_obj_type_t type, hwloc_obj_t obj )` [static]

Returns the ancestor object of *obj* with type *type*.

**13.15.1.3** `static inline hwloc_obj_t hwloc_get_common_ancestor_obj ( hwloc_topology_t topology , hwloc_obj_t obj1, hwloc_obj_t obj2 )` [static]

Returns the common parent object to objects *lv1* and *lv2*.

**13.15.1.4** `static inline hwloc_obj_t hwloc_get_next_child ( hwloc_topology_t topology , hwloc_obj_t parent, hwloc_obj_t prev )` [static]

Return the next child.

If *prev* is NULL, return the first child.

**13.15.1.5** `static inline hwloc_obj_t hwloc_get_next_obj_by_depth ( hwloc_topology_t topology, unsigned depth, hwloc_obj_t prev )` [static]

Returns the next object at depth *depth*.

If *prev* is NULL, return the first object at depth *depth*.

**13.15.1.6** `static inline hwloc_obj_t hwloc_get_next_obj_by_type ( hwloc_topology_t topology, hwloc_obj_type_t type, hwloc_obj_t prev )` [static]

Returns the next object of type *type*.

If *prev* is NULL, return the first object at type *type*. If there are multiple or no depth for given type, return NULL and let the caller fallback to [hwloc\\_get\\_next\\_obj\\_by\\_depth\(\)](#).

**13.15.1.7** `static inline hwloc_obj_t hwloc_get_pu_obj_by_os_index ( hwloc_topology_t topology, unsigned os_index )` [static]

Returns the object of type [HWLOC\\_OBJ\\_PU](#) with *os\_index*.

#### Note

The *os\_index* field of object should most of the times only be used for pretty-printing purpose. Type [HWLOC\\_OBJ\\_PU](#) is the only case where *os\_index* could actually be useful, when manually binding to processors. However, using CPU sets to hide this complexity should often be preferred.

**13.15.1.8** `static inline hwloc_obj_t hwloc_get_root_obj ( hwloc_topology_t topology )`  
`[static]`

Returns the top-object of the topology-tree.

Its type is typically `HWLOC_OBJ_MACHINE` but it could be different for complex topologies. This function replaces the old deprecated `hwloc_get_system_obj()`.

**13.15.1.9** `static inline int hwloc_obj_is_in_subtree ( hwloc_topology_t topology ,`  
`hwloc_obj_t obj, hwloc_obj_t subtree_root )` `[static]`

Returns true if `_obj_` is inside the subtree beginning with `subtree_root`.

## 13.16 Finding Objects Inside a CPU set

### Functions

- `static inline hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set)`
- `int hwloc_get_largest_objs_inside_cpuset (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_t *restrict objs, int max)`
- `static inline hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth, hwloc_obj_t prev)`
- `static inline hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type, hwloc_obj_t prev)`
- `static inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth, unsigned idx)`
- `static inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type, unsigned idx)`
- `static inline unsigned hwloc_get_nobjs_inside_cpuset_by_depth (hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth)`
- `static inline int hwloc_get_nobjs_inside_cpuset_by_type (hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type)`

### 13.16.1 Function Documentation

**13.16.1.1** `static inline hwloc_obj_t hwloc_get_first_largest_obj_inside_cpuset (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set )` `[static]`

Get the first largest object included in the given cpuset `set`.

Returns

the first object that is included in `set` and whose parent is not.

This is convenient for iterating over all largest objects within a CPU set by doing a loop getting the first largest object and clearing its CPU set from the remaining CPU set.

**13.16.1.2** `int hwloc_get_largest_objs_inside_cpuset ( hwloc_topology_t topology,  
hwloc_const_cpuset_t set, hwloc_obj_t *restrict objs, int max )`

Get the set of largest objects covering exactly a given cpuset *set*.

Returns

the number of objects returned in *objs*.

**13.16.1.3** `static inline unsigned hwloc_get_nbobjs_inside_cpuset_by_depth (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth )`  
`[static]`

Return the number of objects at depth *depth* included in CPU set *set*.

**13.16.1.4** `static inline int hwloc_get_nbobjs_inside_cpuset_by_type ( hwloc_topology_t`  
`topology, hwloc_const_cpuset_t set, hwloc_obj_type_t type ) [static]`

Return the number of objects of type *type* included in CPU set *set*.

If no object for that type exists inside CPU set *set*, 0 is returned. If there are several levels with objects of that type inside CPU set *set*, -1 is returned.

**13.16.1.5** `static inline hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_depth (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth,`  
`hwloc_obj_t prev ) [static]`

Return the next object at depth *depth* included in CPU set *set*.

If *prev* is NULL, return the first object at depth *depth* included in *set*. The next invocation should pass the previous return value in *prev* so as to obtain the next object in *set*.

**13.16.1.6** `static inline hwloc_obj_t hwloc_get_next_obj_inside_cpuset_by_type (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t`  
`type, hwloc_obj_t prev ) [static]`

Return the next object of type *type* included in CPU set *set*.

If there are multiple or no depth for given type, return NULL and let the caller fallback to [hwloc\\_get\\_next\\_obj\\_inside\\_cpuset\\_by\\_depth\(\)](#).

**13.16.1.7** `static inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_depth (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set, unsigned depth,  
unsigned idx ) [static]`

Return the `idx`-th object at depth `depth` included in CPU set `set`.

**13.16.1.8** `static inline hwloc_obj_t hwloc_get_obj_inside_cpuset_by_type (`  
`hwloc_topology_t topology, hwloc_const_cpuset_t set, hwloc_obj_type_t  
type, unsigned idx ) [static]`

Return the `idx`-th object of type `type` included in CPU set `set`.

If there are multiple or no depth for given type, return `NULL` and let the caller fallback to [hwloc\\_get\\_obj\\_inside\\_cpuset\\_by\\_depth\(\)](#).

## 13.17 Finding a single Object covering at least CPU set

### Functions

- static inline [hwloc\\_obj\\_t hwloc\\_get\\_child\\_covering\\_cpuset](#) ([hwloc\\_topology\\_t topology](#) , [hwloc\\_const\\_cpuset\\_t set](#), [hwloc\\_obj\\_t parent](#))
- static inline [hwloc\\_obj\\_t hwloc\\_get\\_obj\\_covering\\_cpuset](#) ([hwloc\\_topology\\_t topology](#), [hwloc\\_const\\_cpuset\\_t set](#))

### 13.17.1 Function Documentation

**13.17.1.1** `static inline hwloc_obj_t hwloc_get_child_covering_cpuset (` [hwloc\\_topology\\_t topology](#) , [hwloc\\_const\\_cpuset\\_t set](#), [hwloc\\_obj\\_t parent](#) ) `[static]`

Get the child covering at least CPU set `set`.

#### Returns

`NULL` if no child matches or if `set` is empty.

**13.17.1.2** `static inline hwloc_obj_t hwloc_get_obj_covering_cpuset (` [hwloc\\_topology\\_t topology](#), [hwloc\\_const\\_cpuset\\_t set](#) ) `[static]`

Get the lowest object covering at least CPU set `set`.

#### Returns

`NULL` if no object matches or if `set` is empty.



## 13.18 Finding a set of similar Objects covering at least a CPU set

### Functions

- static inline `hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_depth` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t set`, unsigned `depth`, `hwloc_obj_t prev`)
- static inline `hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_type` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t set`, `hwloc_obj_type_t type`, `hwloc_obj_t prev`)

### 13.18.1 Function Documentation

**13.18.1.1** `static inline hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_depth` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t set`, unsigned `depth`, `hwloc_obj_t prev`) [`static`]

Iterate through same-depth objects covering at least CPU set `set`.

If object `prev` is `NULL`, return the first object at depth `depth` covering at least part of CPU set `set`. The next invocation should pass the previous return value in `prev` so as to obtain the next object covering at least another part of `set`.

**13.18.1.2** `static inline hwloc_obj_t hwloc_get_next_obj_covering_cpuset_by_type` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t set`, `hwloc_obj_type_t type`, `hwloc_obj_t prev`) [`static`]

Iterate through same-type objects covering at least CPU set `set`.

If object `prev` is `NULL`, return the first object of type `type` covering at least part of CPU set `set`. The next invocation should pass the previous return value in `prev` so as to obtain the next object of type `type` covering at least another part of `set`.

If there are no or multiple depths for type `type`, `NULL` is returned. The caller may fallback to `hwloc_get_next_obj_covering_cpuset_by_depth()` for each depth.

## 13.19 Cache-specific Finding Helpers

### Functions

- static inline `hwloc_obj_t hwloc_get_cache_covering_cpuset` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t set`)
- static inline `hwloc_obj_t hwloc_get_shared_cache_covering_obj` (`hwloc_topology_t topology`, `hwloc_obj_t obj`)

### 13.19.1 Function Documentation

**13.19.1.1** `static inline hwloc_obj_t hwloc_get_cache_covering_cpuset ( hwloc_topology_t topology, hwloc_const_cpuset_t set ) [static]`

Get the first cache covering a cpuset *set*.

Returns

NULL if no cache matches

**13.19.1.2** `static inline hwloc_obj_t hwloc_get_shared_cache_covering_obj ( hwloc_topology_t topology , hwloc_obj_t obj ) [static]`

Get the first cache shared between an object and somebody else.

Returns

NULL if no cache matches

## 13.20 Advanced Traversal Helpers

### Functions

- unsigned `hwloc_get_closest_objs (hwloc_topology_t topology, hwloc_obj_t src, hwloc_obj_t *restrict objs, unsigned max)`
- static inline `hwloc_obj_t hwloc_get_obj_below_by_type (hwloc_topology_t topology, hwloc_obj_type_t type1, unsigned idx1, hwloc_obj_type_t type2, unsigned idx2)`
- static inline `hwloc_obj_t hwloc_get_obj_below_array_by_type (hwloc_topology_t topology, int nr, hwloc_obj_type_t *typev, unsigned *idxv)`

### 13.20.1 Function Documentation

**13.20.1.1** `unsigned hwloc_get_closest_objs ( hwloc_topology_t topology, hwloc_obj_t src, hwloc_obj_t *restrict objs, unsigned max )`

Do a depth-first traversal of the topology to find and sort.

all objects that are at the same depth than *src*. Report in *objs* up to *max* physically closest ones to *src*.

Returns

the number of objects returned in *objs*.

**13.20.1.2** `static inline hwloc_obj_t hwloc_get_obj_below_array_by_type ( hwloc_topology_t topology, int nr, hwloc_obj_type_t* typev, unsigned * idxv ) [static]`

Find an object below a chain of objects specified by types and indexes.

This is a generalized version of [hwloc\\_get\\_obj\\_below\\_by\\_type\(\)](#).

Arrays `typev` and `idxv` must contain `nr` types and indexes.

Start from the top system object and walk the arrays `typev` and `idxv`. For each type and index couple in the arrays, look under the previously found object to find the index-th object of the given type. Indexes are specified within the parent, not withing the entire system.

For instance, if `nr` is 3, `typev` contains `NODE`, `SOCKET` and `CORE`, and `idxv` contains 0, 1 and 2, return the third core object below the second socket below the first NUMA node.

**13.20.1.3** `static inline hwloc_obj_t hwloc_get_obj_below_by_type ( hwloc_topology_t topology, hwloc_obj_type_t type1, unsigned idx1, hwloc_obj_type_t type2, unsigned idx2 ) [static]`

Find an object below another object, both specified by types and indexes.

Start from the top system object and find object of type `type1` and index `idx1`. Then look below this object and find another object of type `type2` and index `idx2`. Indexes are specified within the parent, not withing the entire system.

For instance, if `type1` is `SOCKET`, `idx1` is 2, `type2` is `CORE` and `idx2` is 3, return the fourth core object below the third socket.

## 13.21 Binding Helpers

### Functions

- static inline void [hwloc\\_distributev](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_obj\\_t](#) \*root, unsigned n\_roots, [hwloc\\_cpuset\\_t](#) \*cpuset, unsigned n, unsigned until)
- static inline void [hwloc\\_distribute](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_obj\\_t](#) root, [hwloc\\_cpuset\\_t](#) \*cpuset, unsigned n, unsigned until)
- static inline void \* [hwloc\\_alloc\\_membind\\_policy\\_nodeset](#) ([hwloc\\_topology\\_t](#) topology, size\_t len, [hwloc\\_const\\_nodeset\\_t](#) nodeset, [hwloc\\_membind\\_policy\\_t](#) policy, int flags)
- static inline void \* [hwloc\\_alloc\\_membind\\_policy](#) ([hwloc\\_topology\\_t](#) topology, size\_t len, [hwloc\\_const\\_cpuset\\_t](#) cpuset, [hwloc\\_membind\\_policy\\_t](#) policy, int flags)

### 13.21.1 Function Documentation

**13.21.1.1** `static inline void* hwloc_alloc_membind_policy ( hwloc_topology_t topology,  
size_t len, hwloc_const_cpuset_t cpuset, hwloc_membind_policy_t policy,  
int flags )` `[static]`

Allocate some memory on the memory nodes near given cpuset *cpuset*.

This is similar to `hwloc_alloc_membind_policy_nodeset`, but for a given cpuset.

**13.21.1.2** `static inline void* hwloc_alloc_membind_policy_nodeset ( hwloc_topology_t  
topology, size_t len, hwloc_const_nodeset_t nodeset,  
hwloc_membind_policy_t policy, int flags )` `[static]`

Allocate some memory on the given nodeset *nodeset*.

This is similar to `hwloc_alloc_membind` except that it is allowed to change the current memory binding policy, thus providing more binding support, at the expense of changing the current state.

**13.21.1.3** `static inline void hwloc_distribute ( hwloc_topology_t topology, hwloc_obj_t  
root, hwloc_cpuset_t* cpuset, unsigned n, unsigned until )` `[static]`

**13.21.1.4** `static inline void hwloc_distributev ( hwloc_topology_t topology, hwloc_obj_t*  
roots, unsigned n.roots, hwloc_cpuset_t* cpuset, unsigned n, unsigned until )`  
`[static]`

Distribute *n* items over the topology under *root*.

Distribute *n* items over the topology under *roots*.

Array *cpuset* will be filled with *n* cpusets recursively distributed linearly over the topology under *root*, down to depth *until* (which can be `INT_MAX` to distribute down to the finest level).

This is typically useful when an application wants to distribute *n* threads over a machine, giving each of them as much private cache as possible and keeping them locally in number order.

The caller may typically want to also call `hwloc_bitmap_singlify()` before binding a thread so that it does not move at all.

This is the same as `hwloc_distribute`, but takes an array of roots instead of just one root.

## 13.22 Cpuset Helpers

### Functions

- static inline `hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t topology)`
- static inline `hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (hwloc_topology_t topology)`
- static inline `hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (hwloc_topology_t topology)`
- static inline `hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t topology)`

### 13.22.1 Function Documentation

**13.22.1.1** static inline `hwloc_const_cpuset_t hwloc_topology_get_allowed_cpuset (hwloc_topology_t topology)` [static]

Get allowed CPU set.

#### Returns

the CPU set of allowed logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

#### Note

The returned cpuset is not newly allocated and should thus not be changed or freed, `hwloc_cpuset_dup` must be used to obtain a local copy.

**13.22.1.2** static inline `hwloc_const_cpuset_t hwloc_topology_get_complete_cpuset (hwloc_topology_t topology)` [static]

Get complete CPU set.

#### Returns

the complete CPU set of logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

#### Note

The returned cpuset is not newly allocated and should thus not be changed or freed; `hwloc_cpuset_dup` must be used to obtain a local copy.

**13.22.1.3** `static inline hwloc_const_cpuset_t hwloc_topology_get_online_cpuset (`  
`hwloc_topology_t topology ) [static]`

Get online CPU set.

Returns

the CPU set of online logical processors of the system. If the topology is the result of a combination of several systems, NULL is returned.

Note

The returned cpuset is not newly allocated and should thus not be changed or freed; `hwloc_cpuset_dup` must be used to obtain a local copy.

**13.22.1.4** `static inline hwloc_const_cpuset_t hwloc_topology_get_topology_cpuset (`  
`hwloc_topology_t topology ) [static]`

Get topology CPU set.

Returns

the CPU set of logical processors of the system for which hwloc provides topology information. This is equivalent to the cpuset of the system object. If the topology is the result of a combination of several systems, NULL is returned.

Note

The returned cpuset is not newly allocated and should thus not be changed or freed; `hwloc_cpuset_dup` must be used to obtain a local copy.

## 13.23 Nodeset Helpers

### Functions

- static inline [hwloc\\_const\\_nodeset\\_t hwloc\\_topology\\_get\\_complete\\_nodeset \(hwloc\\_topology\\_t topology\)](#)
- static inline [hwloc\\_const\\_nodeset\\_t hwloc\\_topology\\_get\\_topology\\_nodeset \(hwloc\\_topology\\_t topology\)](#)
- static inline [hwloc\\_const\\_nodeset\\_t hwloc\\_topology\\_get\\_allowed\\_nodeset \(hwloc\\_topology\\_t topology\)](#)

### 13.23.1 Function Documentation

**13.23.1.1** `static inline hwloc_const_nodeset_t hwloc_topology_get_allowed_nodeset (`  
`hwloc_topology_t topology ) [static]`

Get allowed node set.

**Returns**

the node set of allowed memory of the system. If the topology is the result of a combination of several systems, NULL is returned.

**Note**

The returned nodeset is not newly allocated and should thus not be changed or freed; `hwloc_nodeset_dup` must be used to obtain a local copy.

**13.23.1.2** `static inline hwloc_const_nodeset_t hwloc_topology_get_complete_nodeset (hwloc_topology_t topology)` `[static]`

Get complete node set.

**Returns**

the complete node set of memory of the system. If the topology is the result of a combination of several systems, NULL is returned.

**Note**

The returned nodeset is not newly allocated and should thus not be changed or freed; `hwloc_nodeset_dup` must be used to obtain a local copy.

**13.23.1.3** `static inline hwloc_const_nodeset_t hwloc_topology_get_topology_nodeset (hwloc_topology_t topology)` `[static]`

Get topology node set.

**Returns**

the node set of memory of the system for which `hwloc` provides topology information. This is equivalent to the nodeset of the system object. If the topology is the result of a combination of several systems, NULL is returned.

**Note**

The returned nodeset is not newly allocated and should thus not be changed or freed; `hwloc_nodeset_dup` must be used to obtain a local copy.

## 13.24 Conversion between cpuset and nodeset

**Functions**

- static inline void `hwloc_cpuset_to_nodeset (hwloc_topology_t topology, hwloc_const_cpuset_t cpuset, hwloc_nodeset_t nodeset)`

- static inline void [hwloc\\_cpuset\\_to\\_nodeset\\_strict](#) (struct hwloc\_topology \*topology, [hwloc\\_const\\_cpuset\\_t](#) cpuset, [hwloc\\_nodeset\\_t](#) nodeset)
- static inline void [hwloc\\_cpuset\\_from\\_nodeset](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_cpuset\\_t](#) cpuset, [hwloc\\_const\\_nodeset\\_t](#) nodeset)
- static inline void [hwloc\\_cpuset\\_from\\_nodeset\\_strict](#) (struct hwloc\_topology \*topology, [hwloc\\_cpuset\\_t](#) cpuset, [hwloc\\_const\\_nodeset\\_t](#) nodeset)

### 13.24.1 Detailed Description

There are two semantics for converting cpusets to nodesets depending on how non-NUMA machines are handled.

When manipulating nodesets for memory binding, non-NUMA machines should be considered as having a single NUMA node. The standard conversion routines below should be used so that marking the first bit of the nodeset means that memory should be bound to a non-NUMA whole machine.

When manipulating nodesets as an actual list of NUMA nodes without any need to handle memory binding on non-NUMA machines, the strict conversion routines may be used instead.

### 13.24.2 Function Documentation

**13.24.2.1** static inline void [hwloc\\_cpuset\\_from\\_nodeset](#) ( [hwloc\\_topology\\_t](#) *topology*, [hwloc\\_cpuset\\_t](#) *cpuset*, [hwloc\\_const\\_nodeset\\_t](#) *nodeset* ) [static]

Convert a NUMA node set into a CPU set and handle non-NUMA cases.

If the topology contains no NUMA nodes, the machine is considered as a single memory node, and the following behavior is used: If *nodeset* is empty, *cpuset* will be emptied as well. Otherwise *cpuset* will be entirely filled. This is useful for manipulating memory binding sets.

**13.24.2.2** static inline void [hwloc\\_cpuset\\_from\\_nodeset\\_strict](#) ( [struct hwloc\\_topology](#) \* *topology*, [hwloc\\_cpuset\\_t](#) *cpuset*, [hwloc\\_const\\_nodeset\\_t](#) *nodeset* ) [static]

Convert a NUMA node set into a CPU set without handling non-NUMA cases.

This is the strict variant of [hwloc\\_cpuset\\_from\\_nodeset](#). It does not fix non-NUMA cases. If the topology contains some NUMA nodes, behave exactly the same. However, if the topology contains no NUMA nodes, return an empty cpuset.

**13.24.2.3** static inline void [hwloc\\_cpuset\\_to\\_nodeset](#) ( [hwloc\\_topology\\_t](#) *topology*, [hwloc\\_const\\_cpuset\\_t](#) *cpuset*, [hwloc\\_nodeset\\_t](#) *nodeset* ) [static]

Convert a CPU set into a NUMA node set and handle non-NUMA cases.



If some NUMA nodes have no CPUs at all, this function never sets their indexes in the output node set, even if a full CPU set is given in input.

If the topology contains no NUMA nodes, the machine is considered as a single memory node, and the following behavior is used: If `cpuset` is empty, `nodeset` will be emptied as well. Otherwise `nodeset` will be entirely filled.

**13.24.2.4** `static inline void hwloc_cpuset_to_nodeset_strict ( struct hwloc_topology * topology, hwloc_const_cpuset_t cpuset, hwloc_nodeset_t nodeset ) [static]`

Convert a CPU set into a NUMA node set without handling non-NUMA cases.

This is the strict variant of `hwloc_cpuset_to_nodeset`. It does not fix non-NUMA cases. If the topology contains some NUMA nodes, behave exactly the same. However, if the topology contains no NUMA nodes, return an empty nodeset.

## 13.25 The bitmap API

### Defines

- `#define hwloc_bitmap_foreach_begin(id, bitmap)`
- `#define hwloc_bitmap_foreach_end()`

### Typedefs

- `typedef struct hwloc_bitmap_s * hwloc_bitmap_t`
- `typedef struct hwloc_bitmap_s * hwloc_const_bitmap_t`

### Functions

- `hwloc_bitmap_t hwloc_bitmap_alloc (void)`
- `hwloc_bitmap_t hwloc_bitmap_alloc_full (void)`
- `void hwloc_bitmap_free (hwloc_bitmap_t bitmap)`
- `hwloc_bitmap_t hwloc_bitmap_dup (hwloc_const_bitmap_t bitmap)`
- `void hwloc_bitmap_copy (hwloc_bitmap_t dst, hwloc_const_bitmap_t src)`
- `int hwloc_bitmap_snprintf (char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)`
- `int hwloc_bitmap_asprintf (char **strp, hwloc_const_bitmap_t bitmap)`
- `int hwloc_bitmap_sscanf (hwloc_bitmap_t bitmap, const char *restrict string)`
- `int hwloc_bitmap_taskset_snprintf (char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap)`
- `int hwloc_bitmap_taskset_asprintf (char **strp, hwloc_const_bitmap_t bitmap)`
- `int hwloc_bitmap_taskset_sscanf (hwloc_bitmap_t bitmap, const char *restrict string)`
- `void hwloc_bitmap_zero (hwloc_bitmap_t bitmap)`
- `void hwloc_bitmap_fill (hwloc_bitmap_t bitmap)`

- void `hwloc_bitmap_only` (`hwloc_bitmap_t` bitmap, unsigned id)
- void `hwloc_bitmap_allbut` (`hwloc_bitmap_t` bitmap, unsigned id)
- void `hwloc_bitmap_from_ulong` (`hwloc_bitmap_t` bitmap, unsigned long mask)
- void `hwloc_bitmap_from_ith_ulong` (`hwloc_bitmap_t` bitmap, unsigned i, unsigned long mask)
- void `hwloc_bitmap_set` (`hwloc_bitmap_t` bitmap, unsigned id)
- void `hwloc_bitmap_set_range` (`hwloc_bitmap_t` bitmap, unsigned begin, unsigned end)
- void `hwloc_bitmap_set_ith_ulong` (`hwloc_bitmap_t` bitmap, unsigned i, unsigned long mask)
- void `hwloc_bitmap_clr` (`hwloc_bitmap_t` bitmap, unsigned id)
- void `hwloc_bitmap_clr_range` (`hwloc_bitmap_t` bitmap, unsigned begin, unsigned end)
- void `hwloc_bitmap_singlify` (`hwloc_bitmap_t` bitmap)
- unsigned long `hwloc_bitmap_to_ulong` (`hwloc_const_bitmap_t` bitmap)
- unsigned long `hwloc_bitmap_to_ith_ulong` (`hwloc_const_bitmap_t` bitmap, unsigned i)
- int `hwloc_bitmap_isset` (`hwloc_const_bitmap_t` bitmap, unsigned id)
- int `hwloc_bitmap_iszero` (`hwloc_const_bitmap_t` bitmap)
- int `hwloc_bitmap_isfull` (`hwloc_const_bitmap_t` bitmap)
- int `hwloc_bitmap_first` (`hwloc_const_bitmap_t` bitmap)
- int `hwloc_bitmap_next` (`hwloc_const_bitmap_t` bitmap, unsigned prev)
- int `hwloc_bitmap_last` (`hwloc_const_bitmap_t` bitmap)
- int `hwloc_bitmap_weight` (`hwloc_const_bitmap_t` bitmap)
- void `hwloc_bitmap_or` (`hwloc_bitmap_t` res, `hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- void `hwloc_bitmap_and` (`hwloc_bitmap_t` res, `hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- void `hwloc_bitmap_andnot` (`hwloc_bitmap_t` res, `hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- void `hwloc_bitmap_xor` (`hwloc_bitmap_t` res, `hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- void `hwloc_bitmap_not` (`hwloc_bitmap_t` res, `hwloc_const_bitmap_t` bitmap)
- int `hwloc_bitmap_intersects` (`hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- int `hwloc_bitmap_isincluded` (`hwloc_const_bitmap_t` sub\_bitmap, `hwloc_const_bitmap_t` super\_bitmap)
- int `hwloc_bitmap_isequal` (`hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- int `hwloc_bitmap_compare_first` (`hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)
- int `hwloc_bitmap_compare` (`hwloc_const_bitmap_t` bitmap1, `hwloc_const_bitmap_t` bitmap2)

### 13.25.1 Detailed Description

The `hwloc_bitmap_t` type represents a set of objects, typically OS processors -- which may actually be hardware threads (represented by `hwloc_cpuset_t`, which is a typedef for `hwloc_bitmap_t`) -- or memory nodes (represented by `hwloc_nodeset_t`, which is also a typedef for `hwloc_bitmap_t`).

*Both CPU and node sets are always indexed by OS physical number.*

#### Note

CPU sets and nodesets are described in [Object sets \(`hwloc\_cpuset\_t` and `hwloc\_nodeset\_t`\)](#).

A bitmap may be of infinite size.

### 13.25.2 Define Documentation

#### 13.25.2.1 `#define hwloc_bitmap_foreach_begin( id, bitmap )`

Loop macro iterating on bitmap `bitmap`.

`index` is the loop variable; it should be an unsigned int. The first iteration will set `index` to the lowest index in the bitmap. Successive iterations will iterate through, in order, all remaining indexes that in the bitmap. To be specific: each iteration will return a value for `index` such that `hwloc_bitmap_isset(bitmap, index)` is true.

The assert prevents the loop from being infinite if the bitmap is infinite.

#### 13.25.2.2 `#define hwloc_bitmap_foreach_end( )`

End of loop. Needs a terminating `;`.

See also

[hwloc\\_bitmap\\_foreach\\_begin](#)

### 13.25.3 Typedef Documentation

#### 13.25.3.1 `typedef struct hwloc_bitmap_s* hwloc_bitmap_t`

Set of bits represented as an opaque pointer to an internal bitmap.

#### 13.25.3.2 `typedef struct hwloc_bitmap_s* hwloc_const_bitmap_t`

a non-modifiable [hwloc\\_bitmap\\_t](#)

### 13.25.4 Function Documentation

#### 13.25.4.1 void hwloc\_bitmap\_allbut ( hwloc\_bitmap\_t *bitmap*, unsigned *id* )

Fill the bitmap and clear the index *id*.

#### 13.25.4.2 hwloc\_bitmap\_t hwloc\_bitmap\_alloc ( void )

Allocate a new empty bitmap.

Returns

A valid bitmap or NULL.

The bitmap should be freed by a corresponding call to [hwloc\\_bitmap\\_free\(\)](#).

#### 13.25.4.3 hwloc\_bitmap\_t hwloc\_bitmap\_alloc\_full ( void )

Allocate a new full bitmap.

#### 13.25.4.4 void hwloc\_bitmap\_and ( hwloc\_bitmap\_t *res*, hwloc\_const\_bitmap\_t *bitmap1*, hwloc\_const\_bitmap\_t *bitmap2* )

And bitmaps *bitmap1* and *bitmap2* and store the result in bitmap *res*.

#### 13.25.4.5 void hwloc\_bitmap\_andnot ( hwloc\_bitmap\_t *res*, hwloc\_const\_bitmap\_t *bitmap1*, hwloc\_const\_bitmap\_t *bitmap2* )

And bitmap *bitmap1* and the negation of *bitmap2* and store the result in bitmap *res*.

#### 13.25.4.6 int hwloc\_bitmap\_asprintf ( char \*\* *strp*, hwloc\_const\_bitmap\_t *bitmap* )

Stringify a bitmap into a newly allocated string.

#### 13.25.4.7 void hwloc\_bitmap\_clr ( hwloc\_bitmap\_t *bitmap*, unsigned *id* )

Remove index *id* from bitmap *bitmap*.

#### 13.25.4.8 void hwloc\_bitmap\_clr\_range ( hwloc\_bitmap\_t *bitmap*, unsigned *begin*, unsigned *end* )

Remove index from *begin* to *end* in bitmap *bitmap*.

**13.25.4.9** `int hwloc_bitmap_compare ( hwloc_const_bitmap_t bitmap1,  
hwloc_const_bitmap_t bitmap2 )`

Compare bitmaps `bitmap1` and `bitmap2` using their highest index.

Higher most significant bit is higher. The empty bitmap is considered lower than anything.

**13.25.4.10** `int hwloc_bitmap_compare_first ( hwloc_const_bitmap_t bitmap1,  
hwloc_const_bitmap_t bitmap2 )`

Compare bitmaps `bitmap1` and `bitmap2` using their lowest index.

Smaller least significant bit is smaller. The empty bitmap is considered higher than anything.

**13.25.4.11** `void hwloc_bitmap_copy ( hwloc_bitmap_t dst, hwloc_const_bitmap_t src )`

Copy the contents of bitmap `src` into the already allocated bitmap `dst`.

**13.25.4.12** `hwloc_bitmap_t hwloc_bitmap_dup ( hwloc_const_bitmap_t bitmap )`

Duplicate bitmap `bitmap` by allocating a new bitmap and copying `bitmap` contents.

**13.25.4.13** `void hwloc_bitmap_fill ( hwloc_bitmap_t bitmap )`

Fill bitmap `bitmap` with all possible indexes (even if those objects don't exist or are otherwise unavailable)

**13.25.4.14** `int hwloc_bitmap_first ( hwloc_const_bitmap_t bitmap )`

Compute the first index (least significant bit) in bitmap `bitmap`.

Returns

-1 if no index is set.

**13.25.4.15** `void hwloc_bitmap_free ( hwloc_bitmap_t bitmap )`

Free bitmap `bitmap`.

**13.25.4.16** `void hwloc_bitmap_from_ith_ulong ( hwloc_bitmap_t bitmap, unsigned i,  
unsigned long mask )`

Setup bitmap `bitmap` from unsigned long `mask` used as `i`-th subset.

**13.25.4.17** void `hwloc_bitmap_from_ulong` ( `hwloc_bitmap_t` *bitmap*, unsigned long *mask* )

Setup bitmap `bitmap` from unsigned long `mask`.

**13.25.4.18** int `hwloc_bitmap_intersects` ( `hwloc_const_bitmap_t` *bitmap1*,  
`hwloc_const_bitmap_t` *bitmap2* )

Test whether bitmaps `bitmap1` and `bitmap2` intersects.

**13.25.4.19** int `hwloc_bitmap_isequal` ( `hwloc_const_bitmap_t` *bitmap1*,  
`hwloc_const_bitmap_t` *bitmap2* )

Test whether bitmap `bitmap1` is equal to bitmap `bitmap2`.

**13.25.4.20** int `hwloc_bitmap_isfull` ( `hwloc_const_bitmap_t` *bitmap* )

Test whether bitmap `bitmap` is completely full.

**13.25.4.21** int `hwloc_bitmap_isincluded` ( `hwloc_const_bitmap_t` *sub\_bitmap*,  
`hwloc_const_bitmap_t` *super\_bitmap* )

Test whether bitmap `sub_bitmap` is part of bitmap `super_bitmap`.

**13.25.4.22** int `hwloc_bitmap_isset` ( `hwloc_const_bitmap_t` *bitmap*, unsigned *id* )

Test whether index `id` is part of bitmap `bitmap`.

**13.25.4.23** int `hwloc_bitmap_iszero` ( `hwloc_const_bitmap_t` *bitmap* )

Test whether bitmap `bitmap` is empty.

**13.25.4.24** int `hwloc_bitmap_last` ( `hwloc_const_bitmap_t` *bitmap* )

Compute the last index (most significant bit) in bitmap `bitmap`.

Returns

-1 if no index is bitmap, or if the index `bitmap` is infinite.

**13.25.4.25** int `hwloc_bitmap_next` ( `hwloc_const_bitmap_t` *bitmap*, unsigned *prev* )

Compute the next index in bitmap `bitmap` which is after index `prev`.

Returns

-1 if no index with higher index is bitmap.

**13.25.4.26** `void hwloc_bitmap_not ( hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap )`

Negate bitmap *bitmap* and store the result in bitmap *res*.

**13.25.4.27** `void hwloc_bitmap_only ( hwloc_bitmap_t bitmap, unsigned id )`

Empty the bitmap *bitmap* and add bit *id*.

**13.25.4.28** `void hwloc_bitmap_or ( hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1,  
hwloc_const_bitmap_t bitmap2 )`

Or bitmaps *bitmap1* and *bitmap2* and store the result in bitmap *res*.

**13.25.4.29** `void hwloc_bitmap_set ( hwloc_bitmap_t bitmap, unsigned id )`

Add index *id* in bitmap *bitmap*.

**13.25.4.30** `void hwloc_bitmap_set_ith_ulong ( hwloc_bitmap_t bitmap, unsigned i,  
unsigned long mask )`

Replace *i*-th subset of bitmap *bitmap* with unsigned long *mask*.

**13.25.4.31** `void hwloc_bitmap_set_range ( hwloc_bitmap_t bitmap, unsigned begin,  
unsigned end )`

Add indexes from *begin* to *end* in bitmap *bitmap*.

**13.25.4.32** `void hwloc_bitmap_singlify ( hwloc_bitmap_t bitmap )`

Keep a single index among those set in bitmap *bitmap*.

May be useful before binding so that the process does not have a chance of migrating between multiple logical CPUs in the original mask.

**13.25.4.33** `int hwloc_bitmap_snprintf ( char *restrict buf, size_t buflen,  
hwloc_const_bitmap_t bitmap )`

Stringify a bitmap.

Up to *buflen* characters may be written in buffer *buf*.

If *buflen* is 0, *buf* may safely be NULL.

## Returns

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

**13.25.4.34** `int hwloc_bitmap_sscanf ( hwloc_bitmap_t bitmap, const char *restrict string )`

Parse a bitmap string and stores it in bitmap `bitmap`.

**13.25.4.35** `int hwloc_bitmap_taskset_asprintf ( char ** strp, hwloc_const_bitmap_t bitmap )`

Stringify a bitmap into a newly allocated taskset-specific string.

**13.25.4.36** `int hwloc_bitmap_taskset_sprintf ( char *restrict buf, size_t buflen, hwloc_const_bitmap_t bitmap )`

Stringify a bitmap in the taskset-specific format.

The taskset command manipulates bitmap strings that contain a single (possible very long) hexadecimal number starting with `0x`.

Up to `buflen` characters may be written in buffer `buf`.

If `buflen` is 0, `buf` may safely be `NULL`.

## Returns

the number of character that were actually written if not truncating, or that would have been written (not including the ending `\0`).

**13.25.4.37** `int hwloc_bitmap_taskset_sscanf ( hwloc_bitmap_t bitmap, const char *restrict string )`

Parse a taskset-specific bitmap string and stores it in bitmap `bitmap`.

**13.25.4.38** `unsigned long hwloc_bitmap_to_ith_ulong ( hwloc_const_bitmap_t bitmap, unsigned i )`

Convert the `i`-th subset of bitmap `bitmap` into unsigned long mask.

**13.25.4.39** `unsigned long hwloc_bitmap_to_ulong ( hwloc_const_bitmap_t bitmap )`

Convert the beginning part of bitmap `bitmap` into unsigned long mask.



**13.25.4.40** `int hwloc_bitmap_weight ( hwloc_const_bitmap_t bitmap )`

Compute the "weight" of bitmap *bitmap* (i.e., number of indexes that are in the bitmap).

Returns

the number of indexes that are in the bitmap.

**13.25.4.41** `void hwloc_bitmap_xor ( hwloc_bitmap_t res, hwloc_const_bitmap_t bitmap1, hwloc_const_bitmap_t bitmap2 )`

Xor bitmaps *bitmap1* and *bitmap2* and store the result in bitmap *res*.

**13.25.4.42** `void hwloc_bitmap_zero ( hwloc_bitmap_t bitmap )`

Empty the bitmap *bitmap*.

## 13.26 Helpers for manipulating glibc sched affinity

### Functions

- static inline int `hwloc_cpuset_to_glibc_sched_affinity` (`hwloc_topology_t` topology, `hwloc_const_cpuset_t` hwlocset, `cpu_set_t` \*schedset, `size_t` schedsetsize)
- static inline int `hwloc_cpuset_from_glibc_sched_affinity` (`hwloc_topology_t` topology, `hwloc_cpuset_t` hwlocset, `const cpu_set_t` \*schedset, `size_t` schedsetsize)

### 13.26.1 Function Documentation

**13.26.1.1** `static inline int hwloc_cpuset_from_glibc_sched_affinity ( hwloc_topology_t topology, hwloc_cpuset_t hwlocset, const cpu_set_t * schedset, size_t schedsetsize )` `[static]`

Convert glibc sched affinity CPU set *schedset* into hwloc CPU set.

This function may be used before calling `sched_setaffinity` or any other function that takes a `cpu_set_t` as input parameter.

*schedsetsize* should be `sizeof(cpu_set_t)` unless *schedset* was dynamically allocated with `CPU_ALLOC`

**13.26.1.2** `static inline int hwloc_cpuset_to_glibc_sched_affinity ( hwloc_topology_t topology, hwloc_const_cpuset_t hwlocset, cpu_set_t * schedset, size_t schedsetsize )` `[static]`

Convert hwloc CPU set *toposet* into glibc sched affinity CPU set *schedset*.

This function may be used before calling `sched_setaffinity` or any other function that takes a `cpu_set_t` as input parameter.

`schedsetsize` should be `sizeof(cpu_set_t)` unless `schedset` was dynamically allocated with `CPU_ALLOC`

## 13.27 Linux-only helpers

### Functions

- `int hwloc_linux_parse_cpumap_file (FILE *file, hwloc_cpuset_t set)`
- `int hwloc_linux_set_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_const_cpuset_t set)`
- `int hwloc_linux_get_tid_cpubind (hwloc_topology_t topology, pid_t tid, hwloc_cpuset_t set)`

#### 13.27.1 Detailed Description

This includes helpers for manipulating linux kernel cpumap files, and hwloc equivalents of the Linux `sched_setaffinity` and `sched_getaffinity` system calls.

#### 13.27.2 Function Documentation

**13.27.2.1** `int hwloc_linux_get_tid_cpubind ( hwloc_topology_t topology, pid_t tid, hwloc_cpuset_t set )`

Get the current binding of thread `tid`.

The behavior is exactly the same as the Linux `sched_setaffinity` system call, but uses a hwloc cpuset.

**13.27.2.2** `int hwloc_linux_parse_cpumap_file ( FILE * file, hwloc_cpuset_t set )`

Convert a linux kernel cpumap file `file` into hwloc CPU set.

Might be used when reading CPU set from sysfs attributes such as topology and caches for processors, or `local_cpus` for devices.

**13.27.2.3** `int hwloc_linux_set_tid_cpubind ( hwloc_topology_t topology, pid_t tid, hwloc_const_cpuset_t set )`

Bind a thread `tid` on cpus given in cpuset `set`.

The behavior is exactly the same as the Linux `sched_setaffinity` system call, but uses a hwloc cpuset.

## 13.28 Helpers for manipulating Linux libnuma unsigned long masks

### Functions

- static inline int `hwloc_cpuset_to_linux_libnuma_ulongs` (`hwloc_topology_t` topology, `hwloc_const_cpuset_t` cpuset, unsigned long \*mask, unsigned long \*maxnode)
- static inline int `hwloc_nodeuset_to_linux_libnuma_ulongs` (`hwloc_topology_t` topology, `hwloc_const_nodeuset_t` nodeset, unsigned long \*mask, unsigned long \*maxnode)
- static inline int `hwloc_cpuset_from_linux_libnuma_ulongs` (`hwloc_topology_t` topology, `hwloc_cpuset_t` cpuset, const unsigned long \*mask, unsigned long maxnode)
- static inline int `hwloc_nodeuset_from_linux_libnuma_ulongs` (`hwloc_topology_t` topology, `hwloc_nodeuset_t` nodeset, const unsigned long \*mask, unsigned long maxnode)

### 13.28.1 Function Documentation

**13.28.1.1** static inline int `hwloc_cpuset_from_linux_libnuma_ulongs` ( `hwloc_topology_t` topology, `hwloc_cpuset_t` cpuset, const unsigned long \* mask, unsigned long maxnode ) [static]

Convert the array of unsigned long mask into hwloc CPU set.

mask is a array of unsigned long that will be read. maxnode contains the maximal node number that may be read in mask.

This function may be used after calling get\_mempolicy or any other function that takes an array of unsigned long as output parameter (and possibly a maximal node number as input parameter).

**13.28.1.2** static inline int `hwloc_cpuset_to_linux_libnuma_ulongs` ( `hwloc_topology_t` topology, `hwloc_const_cpuset_t` cpuset, unsigned long \* mask, unsigned long \* maxnode ) [static]

Convert hwloc CPU set cpuset into the array of unsigned long mask.

mask is the array of unsigned long that will be filled. maxnode contains the maximal node number that may be stored in mask. maxnode will be set to the maximal node number that was found, plus one.

This function may be used before calling set\_mempolicy, mbind, migrate\_pages or any other function that takes an array of unsigned long and a maximal node number as input parameter.

**13.28.1.3** static inline int `hwloc_nodeuset_from_linux_libnuma_ulongs` ( `hwloc_topology_t` topology, `hwloc_nodeuset_t` nodeset, const unsigned long \* mask, unsigned long maxnode ) [static]

Convert the array of unsigned long mask into hwloc NUMA node set.

`mask` is a array of unsigned long that will be read. `maxnode` contains the maximal node number that may be read in `mask`.

This function may be used after calling `get_mempolicy` or any other function that takes an array of unsigned long as output parameter (and possibly a maximal node number as input parameter).

**13.28.1.4** `static inline int hwloc_nodeset_to_linux_libnuma_ulongs ( hwloc_topology_t topology, hwloc_const_nodeset_t nodeset, unsigned long * mask, unsigned long * maxnode ) [static]`

Convert hwloc NUMA node set `nodeset` into the array of unsigned long `mask`.

`mask` is the array of unsigned long that will be filled. `maxnode` contains the maximal node number that may be stored in `mask`. `maxnode` will be set to the maximal node number that was found, plus one.

This function may be used before calling `set_mempolicy`, `mbind`, `migrate_pages` or any other function that takes an array of unsigned long and a maximal node number as input parameter.

## 13.29 Helpers for manipulating Linux libnuma bitmask

### Functions

- static inline struct bitmask \* `hwloc_cpuset_to_linux_libnuma_bitmask` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t cpuset`)
- static inline struct bitmask \* `hwloc_nodeset_to_linux_libnuma_bitmask` (`hwloc_topology_t topology`, `hwloc_const_nodeset_t nodeset`)
- static inline int `hwloc_cpuset_from_linux_libnuma_bitmask` (`hwloc_topology_t topology`, `hwloc_cpuset_t cpuset`, const struct bitmask \*`bitmask`)
- static inline int `hwloc_nodeset_from_linux_libnuma_bitmask` (`hwloc_topology_t topology`, `hwloc_nodeset_t nodeset`, const struct bitmask \*`bitmask`)

### 13.29.1 Function Documentation

**13.29.1.1** `static inline int hwloc_cpuset_from_linux_libnuma_bitmask ( hwloc_topology_t topology, hwloc_cpuset_t cpuset, const struct bitmask * bitmask ) [static]`

Convert libnuma bitmask `bitmask` into hwloc CPU set `cpuset`.

This function may be used after calling many `numa_` functions that use a struct bitmask as an output parameter.

**13.29.1.2** `static inline struct bitmask* hwloc_cpuset_to_linux_libnuma_bitmask ( hwloc_topology_t topology, hwloc_const_cpuset_t cpuset )` [static, read]

Convert hwloc CPU set `cpuset` into the returned libnuma bitmask.

The returned bitmask should later be freed with `numa_bitmask_free`.

This function may be used before calling many `numa_` functions that use a struct bitmask as an input parameter.

#### Returns

newly allocated struct bitmask.

**13.29.1.3** `static inline int hwloc_nodese_t_from_linux_libnuma_bitmask ( hwloc_topology_t topology, hwloc_nodese_t nodese_t, const struct bitmask * bitmask )` [static]

Convert libnuma bitmask `bitmask` into hwloc NUMA node set `nodese_t`.

This function may be used after calling many `numa_` functions that use a struct bitmask as an output parameter.

**13.29.1.4** `static inline struct bitmask* hwloc_nodese_t_to_linux_libnuma_bitmask ( hwloc_topology_t topology, hwloc_const_nodese_t nodese_t )` [static, read]

Convert hwloc NUMA node set `nodese_t` into the returned libnuma bitmask.

The returned bitmask should later be freed with `numa_bitmask_free`.

This function may be used before calling many `numa_` functions that use a struct bitmask as an input parameter.

#### Returns

newly allocated struct bitmask.

## 13.30 Helpers for manipulating Linux libnuma nodemask\_t

### Functions

- static inline int `hwloc_cpuset_to_linux_libnuma_nodemask` (`hwloc_topology_t topology`, `hwloc_const_cpuset_t cpuset`, `nodemask_t *nodemask`)
- static inline int `hwloc_nodese_t_to_linux_libnuma_nodemask` (`hwloc_topology_t topology`, `hwloc_const_nodese_t nodese_t`, `nodemask_t *nodemask`)
- static inline int `hwloc_cpuset_from_linux_libnuma_nodemask` (`hwloc_topology_t topology`, `hwloc_cpuset_t cpuset`, `const nodemask_t *nodemask`)

- static inline int [hwloc\\_nodese](#)[t\\_from\\_linux\\_libnuma\\_nodemask](#) ([hwloc\\_topology\\_t](#) topology, [hwloc\\_nodese](#)[t](#) nodeset, const nodemask\_t \*nodemask)

### 13.30.1 Function Documentation

**13.30.1.1** static inline int [hwloc\\_cpuset\\_from\\_linux\\_libnuma\\_nodemask](#) ( [hwloc\\_topology\\_t](#) topology, [hwloc\\_cpuset\\_t](#) cpuset, const nodemask\_t \* nodemask )  
[static]

Convert libnuma nodemask nodemask into hwloc CPU set cpuset.

This function may be used before calling some old libnuma functions that use a nodemask\_t as an output parameter.

**13.30.1.2** static inline int [hwloc\\_cpuset\\_to\\_linux\\_libnuma\\_nodemask](#) ( [hwloc\\_topology\\_t](#) topology, [hwloc\\_const\\_cpuset\\_t](#) cpuset, nodemask\_t \* nodemask )  
[static]

Convert hwloc CPU set cpuset into libnuma nodemask nodemask.

This function may be used before calling some old libnuma functions that use a nodemask\_t as an input parameter.

**13.30.1.3** static inline int [hwloc\\_nodese](#)[t\\_from\\_linux\\_libnuma\\_nodemask](#) ( [hwloc\\_topology\\_t](#) topology, [hwloc\\_nodese](#)[t](#) nodeset, const nodemask\_t \* nodemask )  
[static]

Convert libnuma nodemask nodemask into hwloc NUMA node set nodeset.

This function may be used before calling some old libnuma functions that use a nodemask\_t as an output parameter.

**13.30.1.4** static inline int [hwloc\\_nodese](#)[t\\_to\\_linux\\_libnuma\\_nodemask](#) ( [hwloc\\_topology\\_t](#) topology, [hwloc\\_const\\_nodese](#)[t](#) nodeset, nodemask\_t \* nodemask )  
[static]

Convert hwloc NUMA node set nodeset into libnuma nodemask nodemask.

This function may be used before calling some old libnuma functions that use a nodemask\_t as an input parameter.

## 13.31 CUDA Driver API Specific Functions

### Functions

- static inline int [hwloc\\_cuda\\_get\\_device\\_cpuset](#) ([hwloc\\_topology\\_t](#) topology , CUdevice cuddevice, [hwloc\\_cpuset\\_t](#) set)

### 13.31.1 Function Documentation

**13.31.1.1** `static inline int hwloc_cuda_get_device_cpuset ( hwloc_topology_t topology , CUdevice cudevice, hwloc_cpuset_t set ) [static]`

Get the CPU set of logical processors that are physically close to device *cudevice*.

For the given CUDA Driver API device *cudevice*, read the corresponding kernel-provided cpumap file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full *cpuset*.

## 13.32 CUDA Runtime API Specific Functions

### Functions

- static inline int `hwloc_cudart_get_device_cpuset (hwloc_topology_t topology , int device, hwloc_cpuset_t set)`

### 13.32.1 Function Documentation

**13.32.1.1** `static inline int hwloc_cudart_get_device_cpuset ( hwloc_topology_t topology , int device, hwloc_cpuset_t set ) [static]`

Get the CPU set of logical processors that are physically close to device *cudevice*.

For the given CUDA Runtime API device *cudevice*, read the corresponding kernel-provided cpumap file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full *cpuset*.

## 13.33 OpenFabrics-Specific Functions

### Functions

- static inline int `hwloc_ibv_get_device_cpuset (hwloc_topology_t topology , struct ibv_device *ibdev, hwloc_cpuset_t set)`

### 13.33.1 Function Documentation

**13.33.1.1** `static inline int hwloc_ibv_get_device_cpuset ( hwloc_topology_t topology , struct ibv_device * ibdev, hwloc_cpuset_t set ) [static]`

Get the CPU set of logical processors that are physically close to device *ibdev*.

For the given OpenFabrics device `ibdev`, read the corresponding kernel-provided `cpumap` file and return the corresponding CPU set. This function is currently only implemented in a meaningful way for Linux; other systems will simply get a full `cpuset`.

## 13.34 Myrinet Express-Specific Functions

### Functions

- static inline int `hwloc_mx_board_get_device_cpuset` (`hwloc_topology_t` topology, unsigned `id`, `hwloc_cpuset_t` set)
- static inline int `hwloc_mx_endpoint_get_device_cpuset` (`hwloc_topology_t` topology, `mx_endpoint_t` endpoint, `hwloc_cpuset_t` set)

#### 13.34.1 Function Documentation

**13.34.1.1** static inline int `hwloc_mx_board_get_device_cpuset` ( `hwloc_topology_t` *topology*, unsigned *id*, `hwloc_cpuset_t` *set* ) [static]

Get the CPU set of logical processors that are physically close the MX board `id`.

For the given Myrinet Express board index `id`, read the OS-provided NUMA node and return the corresponding CPU set.

**13.34.1.2** static inline int `hwloc_mx_endpoint_get_device_cpuset` ( `hwloc_topology_t` *topology*, `mx_endpoint_t` *endpoint*, `hwloc_cpuset_t` *set* ) [static]

Get the CPU set of logical processors that are physically close to endpoint `endpoint`.

For the given Myrinet Express endpoint `endpoint`, read the OS-provided NUMA node and return the corresponding CPU set.



## Chapter 14

# Data Structure Documentation

### 14.1 hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s Struct Reference

Cache-specific Object Attributes.

```
#include <hwloc.h>
```

#### Data Fields

- uint64\_t [size](#)
- unsigned [depth](#)
- unsigned [linesize](#)

#### 14.1.1 Detailed Description

Cache-specific Object Attributes.

#### 14.1.2 Field Documentation

##### 14.1.2.1 unsigned hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s::depth

Depth of cache.

##### 14.1.2.2 unsigned hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s::linesize

Cache-line size in bytes.

##### 14.1.2.3 uint64\_t hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s::size

Size of cache in bytes.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.2 hwloc\_obj\_attr\_u::hwloc\_group\_attr\_s Struct Reference

Group-specific Object Attributes.

```
#include <hwloc.h>
```

### Data Fields

- unsigned [depth](#)

### 14.2.1 Detailed Description

Group-specific Object Attributes.

### 14.2.2 Field Documentation

#### 14.2.2.1 unsigned hwloc\_obj\_attr\_u::hwloc\_group\_attr\_s::depth

Depth of group object.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.3 hwloc\_obj Struct Reference

Structure of a topology object.

```
#include <hwloc.h>
```

### Data Fields

- [hwloc\\_obj\\_type\\_t](#) type
- unsigned [os\\_index](#)
- char \* [name](#)
- struct [hwloc\\_obj\\_memory\\_s](#) memory
- union [hwloc\\_obj\\_attr\\_u](#) \* attr
- unsigned [depth](#)
- unsigned [logical\\_index](#)
- signed [os\\_level](#)

- struct [hwloc\\_obj](#) \* [next\\_cousin](#)
- struct [hwloc\\_obj](#) \* [prev\\_cousin](#)
- struct [hwloc\\_obj](#) \* [parent](#)
- unsigned [sibling\\_rank](#)
- struct [hwloc\\_obj](#) \* [next\\_sibling](#)
- struct [hwloc\\_obj](#) \* [prev\\_sibling](#)
- unsigned [arity](#)
- struct [hwloc\\_obj](#) \*\* [children](#)
- struct [hwloc\\_obj](#) \* [first\\_child](#)
- struct [hwloc\\_obj](#) \* [last\\_child](#)
- void \* [userdata](#)
- [hwloc\\_cpuset\\_t](#) [cpuset](#)
- [hwloc\\_cpuset\\_t](#) [complete\\_cpuset](#)
- [hwloc\\_cpuset\\_t](#) [online\\_cpuset](#)
- [hwloc\\_cpuset\\_t](#) [allowed\\_cpuset](#)
- [hwloc\\_nodeset\\_t](#) [nodeset](#)
- [hwloc\\_nodeset\\_t](#) [complete\\_nodeset](#)
- [hwloc\\_nodeset\\_t](#) [allowed\\_nodeset](#)
- struct [hwloc\\_obj\\_info\\_s](#) \* [infos](#)
- unsigned [infos\\_count](#)

### 14.3.1 Detailed Description

Structure of a topology object. Applications must not modify any field except [hwloc\\_obj.userdata](#).

### 14.3.2 Field Documentation

#### 14.3.2.1 [hwloc\\_cpuset\\_t](#) [hwloc\\_obj::allowed\\_cpuset](#)

The CPU set of allowed logical processors.

This includes the CPUs contained in this object which are allowed for binding, i.e. passing them to the hwloc binding functions should not return permission errors. This is usually restricted by administration rules. Some of them may however be offline so binding to them may still not be possible, see [online\\_cpuset](#).

#### Note

Its value must not be changed, [hwloc\\_bitmap\\_dup](#) must be used instead.

**14.3.2.2** `hwloc_nodest_t hwloc_obj::allowed_nodest`

The set of allowed NUMA memory nodes.

This includes the NUMA memory nodes contained in this object which are allowed for memory allocation, i.e. passing them to NUMA node-directed memory allocation should not return permission errors. This is usually restricted by administration rules.

If there are no NUMA nodes in the machine, all the memory is close to this object, so `allowed_nodest` is full.

Note

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

**14.3.2.3** `unsigned hwloc_obj::arity`

Number of children.

**14.3.2.4** `union hwloc_obj_attr_u* hwloc_obj::attr`

Object type-specific Attributes, may be `NULL` if no attribute value was found.

**14.3.2.5** `struct hwloc_obj** hwloc_obj::children`

Children, `children[0 .. arity - 1]`.

**14.3.2.6** `hwloc_cpuset_t hwloc_obj::complete_cpuset`

The complete CPU set of logical processors of this object,.

This includes not only the same as the `cpuset` field, but also the CPUs for which topology information is unknown or incomplete, and the CPUs that are ignored when the `HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM` flag is not set. Thus no corresponding PU object may be found in the topology, because the precise position is undefined. It is however known that it would be somewhere under this object.

Note

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

**14.3.2.7** `hwloc_nodest_t hwloc_obj::complete_nodest`

The complete NUMA node set of this object,.

This includes not only the same as the `nodest` field, but also the NUMA nodes for which topology information is unknown or incomplete, and the nodes that are ignored when the `HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM` flag is not set. Thus no

corresponding NODE object may be found in the topology, because the precise position is undefined. It is however known that it would be somewhere under this object.

If there are no NUMA nodes in the machine, all the memory is close to this object, so `complete_nodest` is full.

#### Note

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

#### 14.3.2.8 `hwloc_cpuset_t hwloc_obj::cpuset`

CPUs covered by this object.

This is the set of CPUs for which there are PU objects in the topology under this object, i.e. which are known to be physically contained in this object and known how (the children path between this object and the PU objects).

If the `HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM` configuration flag is set, some of these CPUs may be offline, or not allowed for binding, see `online_cpuset` and `allowed_cpuset`.

#### Note

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

#### 14.3.2.9 `unsigned hwloc_obj::depth`

Vertical index in the hierarchy.

#### 14.3.2.10 `struct hwloc_obj* hwloc_obj::first_child`

First child.

#### 14.3.2.11 `struct hwloc_obj_info_s* hwloc_obj::infos`

Array of stringified info type=name.

#### 14.3.2.12 `unsigned hwloc_obj::infos_count`

Size of infos array.

#### 14.3.2.13 `struct hwloc_obj* hwloc_obj::last_child`

Last child.

**14.3.2.14** `unsigned hwloc_obj::logical_index`

Horizontal index in the whole list of similar objects, could be a "cousin\_rank" since it's the rank within the "cousin" list below.

**14.3.2.15** `struct hwloc_obj_memory_s hwloc_obj::memory`

Memory attributes.

**14.3.2.16** `char* hwloc_obj::name`

Object description if any.

**14.3.2.17** `struct hwloc_obj* hwloc_obj::next_cousin`

Next object of same type.

**14.3.2.18** `struct hwloc_obj* hwloc_obj::next_sibling`

Next object below the same parent.

**14.3.2.19** `hwloc_nodeset_t hwloc_obj::nodeset`

NUMA nodes covered by this object or containing this object.

This is the set of NUMA nodes for which there are NODE objects in the topology under or above this object, i.e. which are known to be physically contained in this object or containing it and known how (the children path between this object and the NODE objects).

In the end, these nodes are those that are close to the current object.

If the `HWLOC_TOPOLOGY_FLAG_WHOLE_SYSTEM` configuration flag is set, some of these nodes may not be allowed for allocation, see `allowed_nodeset`.

If there are no NUMA nodes in the machine, all the memory is close to this object, so `nodeset` is full.

Note

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

**14.3.2.20** `hwloc_cpuset_t hwloc_obj::online_cpuset`

The CPU set of online logical processors.

This includes the CPUs contained in this object that are online, i.e. draw power and can execute threads. It may however not be allowed to bind to them due to administration rules, see `allowed_cpuset`.

**Note**

Its value must not be changed, `hwloc_bitmap_dup` must be used instead.

**14.3.2.21 unsigned hwloc\_obj::os\_index**

OS-provided physical index number.

**14.3.2.22 signed hwloc\_obj::os\_level**

OS-provided physical level, -1 if unknown or meaningless.

**14.3.2.23 struct hwloc\_obj\* hwloc\_obj::parent**

Parent, NULL if root (system object)

**14.3.2.24 struct hwloc\_obj\* hwloc\_obj::prev\_cousin**

Previous object of same type.

**14.3.2.25 struct hwloc\_obj\* hwloc\_obj::prev\_sibling**

Previous object below the same parent.

**14.3.2.26 unsigned hwloc\_obj::sibling\_rank**

Index in parent's `children[]` array.

**14.3.2.27 hwloc\_obj\_type\_t hwloc\_obj::type**

Type of object.

**14.3.2.28 void\* hwloc\_obj::userdata**

Application-given private data pointer, initialized to NULL, use it as you wish.

The documentation for this struct was generated from the following file:

- `hwloc.h`

## 14.4 hwloc\_obj\_attr\_u Union Reference

Object type-specific Attributes.

```
#include <hwloc.h>
```

### Data Structures

- struct [hwloc\\_cache\\_attr\\_s](#)  
*Cache-specific Object Attributes.*
- struct [hwloc\\_group\\_attr\\_s](#)  
*Group-specific Object Attributes.*

### Data Fields

- struct [hwloc\\_obj\\_attr\\_u::hwloc\\_cache\\_attr\\_s](#) cache
- struct [hwloc\\_obj\\_attr\\_u::hwloc\\_group\\_attr\\_s](#) group

### 14.4.1 Detailed Description

Object type-specific Attributes.

### 14.4.2 Field Documentation

**14.4.2.1** struct [hwloc\\_obj\\_attr\\_u::hwloc\\_cache\\_attr\\_s](#) [hwloc\\_obj\\_attr\\_u::cache](#)

**14.4.2.2** struct [hwloc\\_obj\\_attr\\_u::hwloc\\_group\\_attr\\_s](#) [hwloc\\_obj\\_attr\\_u::group](#)

The documentation for this union was generated from the following file:

- [hwloc.h](#)

## 14.5 hwloc\_obj\_info\_s Struct Reference

Object info.

```
#include <hwloc.h>
```

### Data Fields

- char \* [name](#)
- char \* [value](#)



### 14.5.1 Detailed Description

Object info.

### 14.5.2 Field Documentation

#### 14.5.2.1 char\* hwloc\_obj\_info\_s::name

Info name.

#### 14.5.2.2 char\* hwloc\_obj\_info\_s::value

Info value.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.6 hwloc\_obj\_memory\_s::hwloc\_obj\_memory\_page\_type\_s Struct Reference

Array of local memory page types, NULL if no local memory and `page_types` is 0.

```
#include <hwloc.h>
```

### Data Fields

- uint64\_t [size](#)
- uint64\_t [count](#)

### 14.6.1 Detailed Description

Array of local memory page types, NULL if no local memory and `page_types` is 0. The array is sorted by increasing `size` fields. It contains `page_types_len` slots.

### 14.6.2 Field Documentation

#### 14.6.2.1 uint64\_t hwloc\_obj\_memory\_s::hwloc\_obj\_memory\_page\_type\_s::count

Number of pages of this size.

#### 14.6.2.2 `uint64_t hwloc_obj_memory_s::hwloc_obj_memory_page_type_s::size`

Size of pages.

The documentation for this struct was generated from the following file:

- `hwloc.h`

## 14.7 `hwloc_obj_memory_s` Struct Reference

Object memory.

```
#include <hwloc.h>
```

### Data Structures

- struct [hwloc\\_obj\\_memory\\_page\\_type\\_s](#)  
*Array of local memory page types, NULL if no local memory and page\_types is 0.*

### Data Fields

- `uint64_t total_memory`
- `uint64_t local_memory`
- `unsigned page_types_len`
- struct [hwloc\\_obj\\_memory\\_s::hwloc\\_obj\\_memory\\_page\\_type\\_s](#) \* `page_types`

#### 14.7.1 Detailed Description

Object memory.

#### 14.7.2 Field Documentation

##### 14.7.2.1 `uint64_t hwloc_obj_memory_s::local_memory`

Local memory (in bytes)

##### 14.7.2.2 `struct hwloc_obj_memory_s::hwloc_obj_memory_page_type_s * hwloc_obj_memory_s::page_types`

##### 14.7.2.3 `unsigned hwloc_obj_memory_s::page_types_len`

Size of array `page_types`.

#### 14.7.2.4 uint64\_t hwloc\_obj\_memory\_s::total\_memory

Total memory (in bytes) in this object and its children.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.8 hwloc\_topology\_cpupbind\_support Struct Reference

Flags describing actual PU binding support for this topology.

```
#include <hwloc.h>
```

### Data Fields

- unsigned char [set\\_thisproc\\_cpupbind](#)
- unsigned char [get\\_thisproc\\_cpupbind](#)
- unsigned char [set\\_proc\\_cpupbind](#)
- unsigned char [get\\_proc\\_cpupbind](#)
- unsigned char [set\\_thisthread\\_cpupbind](#)
- unsigned char [get\\_thisthread\\_cpupbind](#)
- unsigned char [set\\_thread\\_cpupbind](#)
- unsigned char [get\\_thread\\_cpupbind](#)

#### 14.8.1 Detailed Description

Flags describing actual PU binding support for this topology.

#### 14.8.2 Field Documentation

##### 14.8.2.1 unsigned char hwloc\_topology\_cpupbind\_support::get\_proc\_cpupbind

Getting the binding of a whole given process is supported.

##### 14.8.2.2 unsigned char hwloc\_topology\_cpupbind\_support::get\_thisproc\_cpupbind

Getting the binding of the whole current process is supported.

##### 14.8.2.3 unsigned char hwloc\_topology\_cpupbind\_support::get\_thisthread\_cpupbind

Getting the binding of the current thread only is supported.

**14.8.2.4 unsigned char hwloc\_topology\_cpubind\_support::get\_thread\_cpubind**

Getting the binding of a given thread only is supported.

**14.8.2.5 unsigned char hwloc\_topology\_cpubind\_support::set\_proc\_cpubind**

Binding a whole given process is supported.

**14.8.2.6 unsigned char hwloc\_topology\_cpubind\_support::set\_thisproc\_cpubind**

Binding the whole current process is supported.

**14.8.2.7 unsigned char hwloc\_topology\_cpubind\_support::set\_thisthread\_cpubind**

Binding the current thread only is supported.

**14.8.2.8 unsigned char hwloc\_topology\_cpubind\_support::set\_thread\_cpubind**

Binding a given thread only is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.9 hwloc\_topology\_discovery\_support Struct Reference

Flags describing actual discovery support for this topology.

```
#include <hwloc.h>
```

### Data Fields

- unsigned char [pu](#)

#### 14.9.1 Detailed Description

Flags describing actual discovery support for this topology.

#### 14.9.2 Field Documentation

**14.9.2.1 unsigned char hwloc\_topology\_discovery\_support::pu**

Detecting the number of PU objects is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

## 14.10 hwloc\_topology\_membind\_support Struct Reference

Flags describing actual memory binding support for this topology.

```
#include <hwloc.h>
```

### Data Fields

- unsigned char [set\\_thisproc\\_membind](#)
- unsigned char [get\\_thisproc\\_membind](#)
- unsigned char [set\\_proc\\_membind](#)
- unsigned char [get\\_proc\\_membind](#)
- unsigned char [set\\_thisthread\\_membind](#)
- unsigned char [get\\_thisthread\\_membind](#)
- unsigned char [set\\_area\\_membind](#)
- unsigned char [get\\_area\\_membind](#)
- unsigned char [alloc\\_membind](#)
- unsigned char [firsttouch\\_membind](#)
- unsigned char [bind\\_membind](#)
- unsigned char [interleave\\_membind](#)
- unsigned char [replicate\\_membind](#)
- unsigned char [nexttouch\\_membind](#)
- unsigned char [migrate\\_membind](#)

### 14.10.1 Detailed Description

Flags describing actual memory binding support for this topology.

### 14.10.2 Field Documentation

#### 14.10.2.1 unsigned char hwloc\_topology\_membind\_support::alloc\_membind

Allocating a bound memory area is supported.

#### 14.10.2.2 unsigned char hwloc\_topology\_membind\_support::bind\_membind

Bind policy is supported.

#### 14.10.2.3 unsigned char hwloc\_topology\_membind\_support::firsttouch\_membind

First-touch policy is supported.

**14.10.2.4 unsigned char hwloc\_topology\_membed\_support::get\_area\_membed**

Getting the binding of a given memory area is supported.

**14.10.2.5 unsigned char hwloc\_topology\_membed\_support::get\_proc\_membed**

Getting the binding of a whole given process is supported.

**14.10.2.6 unsigned char hwloc\_topology\_membed\_support::get\_thisproc\_membed**

Getting the binding of the whole current process is supported.

**14.10.2.7 unsigned char hwloc\_topology\_membed\_support::get\_thisthread\_membed**

Getting the binding of the current thread only is supported.

**14.10.2.8 unsigned char hwloc\_topology\_membed\_support::interleave\_membed**

Interleave policy is supported.

**14.10.2.9 unsigned char hwloc\_topology\_membed\_support::migrate\_membed**

Migration flags is supported.

**14.10.2.10 unsigned char hwloc\_topology\_membed\_support::nexttouch\_membed**

Next-touch migration policy is supported.

**14.10.2.11 unsigned char hwloc\_topology\_membed\_support::replicate\_membed**

Replication policy is supported.

**14.10.2.12 unsigned char hwloc\_topology\_membed\_support::set\_area\_membed**

Binding a given memory area is supported.

**14.10.2.13 unsigned char hwloc\_topology\_membed\_support::set\_proc\_membed**

Binding a whole given process is supported.

**14.10.2.14 unsigned char hwloc\_topology\_membed\_support::set\_thisproc\_membed**

Binding the whole current process is supported.

#### 14.10.2.15 unsigned char hwloc\_topology\_membind\_support::set\_thisthread\_membind

Binding the current thread only is supported.

The documentation for this struct was generated from the following file:

- hwloc.h

### 14.11 hwloc\_topology\_support Struct Reference

Set of flags describing actual support for this topology.

```
#include <hwloc.h>
```

#### Data Fields

- struct [hwloc\\_topology\\_discovery\\_support](#) \* [discovery](#)
- struct [hwloc\\_topology\\_cpubind\\_support](#) \* [cpubind](#)
- struct [hwloc\\_topology\\_membind\\_support](#) \* [membind](#)

#### 14.11.1 Detailed Description

Set of flags describing actual support for this topology. This is retrieved with [hwloc\\_topology\\_get\\_support\(\)](#) and will be valid until the topology object is destroyed. Note: the values are correct only after discovery.

#### 14.11.2 Field Documentation

**14.11.2.1** **struct** [hwloc\\_topology\\_cpubind\\_support](#)\* [hwloc\\_topology\\_support::cpubind](#)

**14.11.2.2** **struct** [hwloc\\_topology\\_discovery\\_support](#)\* [hwloc\\_topology\\_support::discovery](#)

**14.11.2.3** **struct** [hwloc\\_topology\\_membind\\_support](#)\* [hwloc\\_topology\\_support::membind](#)

The documentation for this struct was generated from the following file:

- hwloc.h

# Index

- Advanced Traversal Helpers, [78](#)
- alloc\_membind
  - hwloc\_topology\_membind\_support, [113](#)
- allowed\_cpuset
  - hwloc\_obj, [103](#)
- allowed\_nodeset
  - hwloc\_obj, [103](#)
- API version, [43](#)
- arity
  - hwloc\_obj, [104](#)
- attr
  - hwloc\_obj, [104](#)
- Basic Traversal Helpers, [72](#)
- bind\_membind
  - hwloc\_topology\_membind\_support, [113](#)
- Binding Helpers, [79](#)
- cache
  - hwloc\_obj\_attr\_u, [108](#)
- Cache-specific Finding Helpers, [77](#)
- children
  - hwloc\_obj, [104](#)
- complete\_cpuset
  - hwloc\_obj, [104](#)
- complete\_nodeset
  - hwloc\_obj, [104](#)
- Configure Topology Detection, [48](#)
- Conversion between cpuset and nodeset, [83](#)
- count
  - hwloc\_obj\_memory\_s::hwloc\_obj\_memory\_s, [109](#)
- CPU binding, [58](#)
- cpubind
  - hwloc\_topology\_support, [115](#)
- cpuset
  - hwloc\_obj, [105](#)
- Cpuset Helpers, [81](#)
- Create and Destroy Topologies, [47](#)
- CUDA Driver API Specific Functions, [98](#)
- CUDA Runtime API Specific Functions, [99](#)
- depth
  - hwloc\_obj, [105](#)
  - hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s, [101](#)
  - hwloc\_obj\_attr\_u::hwloc\_group\_attr\_s, [102](#)
- discovery
  - hwloc\_topology\_support, [115](#)
- Finding a set of similar Objects covering at least a CPU set, [77](#)
- Finding a single Object covering at least CPU set, [76](#)
- Finding Objects Inside a CPU set, [74](#)
- first\_child
  - hwloc\_obj, [105](#)
- firsttouch\_membind
  - hwloc\_topology\_membind\_support, [113](#)
- Get some Topology Information, [54](#)
- get\_area\_membind
  - hwloc\_topology\_membind\_support, [113](#)
- get\_proc\_cpubind
  - hwloc\_topology\_cpubind\_support, [111](#)
- get\_proc\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- get\_thisproc\_cpubind
  - hwloc\_topology\_cpubind\_support, [111](#)
- get\_thisproc\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- get\_thisthread\_cpubind
  - hwloc\_topology\_cpubind\_support, [111](#)
- get\_thisthread\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- get\_thread\_cpubind
  - hwloc\_topology\_cpubind\_support, [111](#)
- group
  - hwloc\_obj\_attr\_u, [108](#)



- Helpers for manipulating glibc sched affinity, [93](#)
- Helpers for manipulating Linux libnuma bitmask, [96](#)
- Helpers for manipulating Linux libnuma nodemask\_t, [97](#)
- Helpers for manipulating Linux libnuma unsigned long masks, [95](#)
- HWLOC\_CPUBIND\_NOMEMBIND
  - hwlocality\_cpubinding, [60](#)
- HWLOC\_CPUBIND\_PROCESS
  - hwlocality\_cpubinding, [60](#)
- HWLOC\_CPUBIND\_STRICT
  - hwlocality\_cpubinding, [60](#)
- HWLOC\_CPUBIND\_THREAD
  - hwlocality\_cpubinding, [60](#)
- HWLOC\_MEMBIND\_BIND
  - hwlocality\_membinding, [65](#)
- HWLOC\_MEMBIND\_DEFAULT
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_FIRSTTOUCH
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_INTERLEAVE
  - hwlocality\_membinding, [65](#)
- HWLOC\_MEMBIND\_MIGRATE
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_MIXED
  - hwlocality\_membinding, [65](#)
- HWLOC\_MEMBIND\_NEXTTOUCH
  - hwlocality\_membinding, [65](#)
- HWLOC\_MEMBIND\_NOCPUBIND
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_PROCESS
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_REPLICATE
  - hwlocality\_membinding, [65](#)
- HWLOC\_MEMBIND\_STRICT
  - hwlocality\_membinding, [64](#)
- HWLOC\_MEMBIND\_THREAD
  - hwlocality\_membinding, [64](#)
- HWLOC\_OBJ\_CACHE
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_CORE
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_GROUP
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_MACHINE
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_MISC
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_NODE
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_PU
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_SOCKET
  - hwlocality\_types, [46](#)
- HWLOC\_OBJ\_SYSTEM
  - hwlocality\_types, [46](#)
- HWLOC\_TOPOLOGY\_FLAG\_IS\_THISSYSTEM
  - hwlocality\_configuration, [50](#)
- HWLOC\_TOPOLOGY\_FLAG\_WHOLE\_SYSTEM
  - hwlocality\_configuration, [50](#)
- HWLOC\_TYPE\_DEPTH\_MULTIPLE
  - hwlocality\_information, [54](#)
- HWLOC\_TYPE\_DEPTH\_UNKNOWN
  - hwlocality\_information, [54](#)
- HWLOC\_TYPE\_UNORDERED
  - hwlocality\_types, [45](#)
- hwloc\_alloc
  - hwlocality\_membinding, [65](#)
- hwloc\_alloc\_membind
  - hwlocality\_membinding, [65](#)
- hwloc\_alloc\_membind\_nodeset
  - hwlocality\_membinding, [66](#)
- hwloc\_alloc\_membind\_policy
  - hwlocality\_helper\_binding, [80](#)
- hwloc\_alloc\_membind\_policy\_nodeset
  - hwlocality\_helper\_binding, [80](#)
- HWLOC\_API\_VERSION
  - hwlocality\_api\_version, [43](#)
- hwloc\_bitmap\_allbut
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_alloc
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_alloc\_full
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_and
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_andnot
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_asprintf
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_clr
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_clr\_range
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_compare
  - hwlocality\_bitmap, [88](#)
- hwloc\_bitmap\_compare\_first

- hwlocality\_bitmap, 89
- hwloc\_bitmap\_copy
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_dup
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_fill
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_first
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_foreach\_begin
  - hwlocality\_bitmap, 87
- hwloc\_bitmap\_foreach\_end
  - hwlocality\_bitmap, 87
- hwloc\_bitmap\_free
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_from\_ith\_ulong
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_from\_ulong
  - hwlocality\_bitmap, 89
- hwloc\_bitmap\_intersects
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_isequal
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_isfull
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_isincluded
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_isset
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_iszero
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_last
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_next
  - hwlocality\_bitmap, 90
- hwloc\_bitmap\_not
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_only
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_or
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_set
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_set\_ith\_ulong
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_set\_range
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_singlify
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_snprintf
  - hwlocality\_bitmap, 91
- hwloc\_bitmap\_sscanf
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_t
  - hwlocality\_bitmap, 87
- hwloc\_bitmap\_taskset\_asprintf
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_taskset\_snprintf
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_taskset\_sscanf
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_to\_ith\_ulong
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_to\_ulong
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_weight
  - hwlocality\_bitmap, 92
- hwloc\_bitmap\_xor
  - hwlocality\_bitmap, 93
- hwloc\_bitmap\_zero
  - hwlocality\_bitmap, 93
- hwloc\_compare\_types
  - hwlocality\_types, 46
- hwloc\_compare\_types\_e
  - hwlocality\_types, 45
- hwloc\_const\_bitmap\_t
  - hwlocality\_bitmap, 87
- hwloc\_const\_cpuset\_t
  - hwlocality\_sets, 44
- hwloc\_const\_nodese\_t
  - hwlocality\_sets, 44
- hwloc\_cpubind\_flags\_t
  - hwlocality\_cpubinding, 59
- hwloc\_cpuset\_from\_glibc\_sched\_affinity
  - hwlocality\_glibc\_sched, 93
- hwloc\_cpuset\_from\_linux\_libnuma\_bitmask
  - hwlocality\_linux\_libnuma\_bitmask, 96
- hwloc\_cpuset\_from\_linux\_libnuma\_nodemask
  - hwlocality\_linux\_libnuma\_nodemask, 98
- hwloc\_cpuset\_from\_linux\_libnuma\_ulong
  - hwlocality\_linux\_libnuma\_ulong, 95
- hwloc\_cpuset\_from\_nodese\_t
  - hwlocality\_helper\_nodese\_convert, 84
- hwloc\_cpuset\_from\_nodese\_t\_strict
  - hwlocality\_helper\_nodese\_convert, 84
- hwloc\_cpuset\_t
  - hwlocality\_sets, 44
- hwloc\_cpuset\_to\_glibc\_sched\_affinity
  - hwlocality\_glibc\_sched, 93

- hwloc\_cpuset\_to\_linux\_libnuma\_bitmask
  - hwlocality\_linux\_libnuma\_bitmask, 96
- hwloc\_cpuset\_to\_linux\_libnuma\_nodemask
  - hwlocality\_linux\_libnuma\_nodemask, 98
- hwloc\_cpuset\_to\_linux\_libnuma\_ulongs
  - hwlocality\_linux\_libnuma\_ulongs, 95
- hwloc\_cpuset\_to\_nodese
  - hwlocality\_helper\_nodese\_convert, 84
- hwloc\_cpuset\_to\_nodese\_strict
  - hwlocality\_helper\_nodese\_convert, 85
- hwloc\_cuda\_get\_device\_cpuset
  - hwlocality\_cuda, 99
- hwloc\_cudart\_get\_device\_cpuset
  - hwlocality\_cudart, 99
- hwloc\_distribute
  - hwlocality\_helper\_binding, 80
- hwloc\_distributev
  - hwlocality\_helper\_binding, 80
- hwloc\_free
  - hwlocality\_membinding, 66
- hwloc\_get\_ancestor\_obj\_by\_depth
  - hwlocality\_helper\_traversal\_basic, 72
- hwloc\_get\_ancestor\_obj\_by\_type
  - hwlocality\_helper\_traversal\_basic, 72
- hwloc\_get\_api\_version
  - hwlocality\_api\_version, 43
- hwloc\_get\_area\_membind
  - hwlocality\_membinding, 66
- hwloc\_get\_area\_membind\_nodese
  - hwlocality\_membinding, 66
- hwloc\_get\_cache\_covering\_cpuset
  - hwlocality\_helper\_find\_cache, 78
- hwloc\_get\_child\_covering\_cpuset
  - hwlocality\_helper\_find\_covering, 76
- hwloc\_get\_closest\_objs
  - hwlocality\_helper\_traversal, 78
- hwloc\_get\_common\_ancestor\_obj
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_cpupbind
  - hwlocality\_cpupbinding, 60
- hwloc\_get\_depth\_type
  - hwlocality\_information, 54
- hwloc\_get\_first\_largest\_obj\_inside\_cpuset
  - hwlocality\_helper\_find\_inside, 74
- hwloc\_get\_largest\_objs\_inside\_cpuset
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_membind
  - hwlocality\_membinding, 67
- hwloc\_get\_membind\_nodese
  - hwlocality\_membinding, 68
- hwloc\_get\_nbobjs\_by\_depth
  - hwlocality\_information, 54
- hwloc\_get\_nbobjs\_by\_type
  - hwlocality\_information, 55
- hwloc\_get\_nbobjs\_inside\_cpuset\_by\_depth
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_nbobjs\_inside\_cpuset\_by\_type
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_next\_child
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_next\_obj\_by\_depth
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_next\_obj\_by\_type
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_next\_obj\_covering\_cpuset\_by\_depth
  - hwlocality\_helper\_find\_coverings, 77
- hwloc\_get\_next\_obj\_covering\_cpuset\_by\_type
  - hwlocality\_helper\_find\_coverings, 77
- hwloc\_get\_next\_obj\_inside\_cpuset\_by\_depth
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_next\_obj\_inside\_cpuset\_by\_type
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_obj\_below\_array\_by\_type
  - hwlocality\_helper\_traversal, 78
- hwloc\_get\_obj\_below\_by\_type
  - hwlocality\_helper\_traversal, 79
- hwloc\_get\_obj\_by\_depth
  - hwlocality\_traversal, 56
- hwloc\_get\_obj\_by\_type
  - hwlocality\_traversal, 56
- hwloc\_get\_obj\_covering\_cpuset
  - hwlocality\_helper\_find\_covering, 76
- hwloc\_get\_obj\_inside\_cpuset\_by\_depth
  - hwlocality\_helper\_find\_inside, 75
- hwloc\_get\_obj\_inside\_cpuset\_by\_type
  - hwlocality\_helper\_find\_inside, 76
- hwloc\_get\_proc\_cpupbind
  - hwlocality\_cpupbinding, 60
- hwloc\_get\_proc\_membind
  - hwlocality\_membinding, 68
- hwloc\_get\_proc\_membind\_nodese
  - hwlocality\_membinding, 69
- hwloc\_get\_pu\_obj\_by\_os\_index
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_root\_obj
  - hwlocality\_helper\_traversal\_basic, 73
- hwloc\_get\_shared\_cache\_covering\_obj

- hwlocality\_helper\_find\_cache, 78
- hwloc\_get\_thread\_cpubind
  - hwlocality\_cpubinding, 60
- hwloc\_get\_type\_depth
  - hwlocality\_information, 55
- hwloc\_get\_type\_depth\_e
  - hwlocality\_information, 54
- hwloc\_get\_type\_or\_above\_depth
  - hwlocality\_helper\_types, 72
- hwloc\_get\_type\_or\_below\_depth
  - hwlocality\_helper\_types, 72
- hwloc\_ibv\_get\_device\_cpuset
  - hwlocality\_openfabrics, 99
- hwloc\_linux\_get\_tid\_cpubind
  - hwlocality\_linux, 94
- hwloc\_linux\_parse\_cpumap\_file
  - hwlocality\_linux, 94
- hwloc\_linux\_set\_tid\_cpubind
  - hwlocality\_linux, 94
- hwloc\_membind\_flags\_t
  - hwlocality\_membinding, 63
- hwloc\_membind\_policy\_t
  - hwlocality\_membinding, 64
- hwloc\_mx\_board\_get\_device\_cpuset
  - hwlocality\_myriexpress, 100
- hwloc\_mx\_endpoint\_get\_device\_cpuset
  - hwlocality\_myriexpress, 100
- hwloc\_nodest\_from\_linux\_libnuma\_bitmask
  - hwlocality\_linux\_libnuma\_bitmask, 97
- hwloc\_nodest\_from\_linux\_libnuma\_nodemask
  - hwlocality\_linux\_libnuma\_nodemask, 98
- hwloc\_nodest\_from\_linux\_libnuma\_ulongs
  - hwlocality\_linux\_libnuma\_ulongs, 95
- hwloc\_nodest\_t
  - hwlocality\_sets, 44
- hwloc\_nodest\_to\_linux\_libnuma\_bitmask
  - hwlocality\_linux\_libnuma\_bitmask, 97
- hwloc\_nodest\_to\_linux\_libnuma\_nodemask
  - hwlocality\_linux\_libnuma\_nodemask, 98
- hwloc\_nodest\_to\_linux\_libnuma\_ulongs
  - hwlocality\_linux\_libnuma\_ulongs, 96
- hwloc\_obj, 102
  - allowed\_cpuset, 103
  - allowed\_nodest, 103
  - arity, 104
  - attr, 104
  - children, 104
  - complete\_cpuset, 104
  - complete\_nodest, 104
  - cpuset, 105
  - depth, 105
  - first\_child, 105
  - infos, 105
  - infos\_count, 105
  - last\_child, 105
  - logical\_index, 105
  - memory, 106
  - name, 106
  - next\_cousin, 106
  - next\_sibling, 106
  - nodeset, 106
  - online\_cpuset, 106
  - os\_index, 107
  - os\_level, 107
  - parent, 107
  - prev\_cousin, 107
  - prev\_sibling, 107
  - sibling\_rank, 107
  - type, 107
  - userdata, 107
- hwloc\_obj\_attr\_snprintf
  - hwlocality\_conversion, 56
- hwloc\_obj\_attr\_u, 108
  - cache, 108
  - group, 108
- hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_s, 101
  - depth, 101
  - linesize, 101
  - size, 101
- hwloc\_obj\_attr\_u::hwloc\_group\_attr\_s, 102
  - depth, 102
- hwloc\_obj\_cpuset\_snprintf
  - hwlocality\_conversion, 57
- hwloc\_obj\_get\_info\_by\_name
  - hwlocality\_conversion, 57
- hwloc\_obj\_info\_s, 108
  - name, 109
  - value, 109
- hwloc\_obj\_is\_in\_subtree
  - hwlocality\_helper\_traversal\_basic, 74
- hwloc\_obj\_memory\_s, 110
  - local\_memory, 110
  - page\_types, 110
  - page\_types\_len, 110
  - total\_memory, 110
- hwloc\_obj\_memory\_s::hwloc\_obj\_memory\_
  - page\_type\_s, 109
  - count, 109

- size, [109](#)
- hwloc\_obj\_snprintf
  - hwlocality\_conversion, [57](#)
- hwloc\_obj\_t
  - hwlocality\_objects, [47](#)
- hwloc\_obj\_type\_of\_string
  - hwlocality\_conversion, [57](#)
- hwloc\_obj\_type\_snprintf
  - hwlocality\_conversion, [58](#)
- hwloc\_obj\_type\_string
  - hwlocality\_conversion, [58](#)
- hwloc\_obj\_type\_t
  - hwlocality\_types, [45](#)
- hwloc\_set\_area\_membind
  - hwlocality\_membinding, [69](#)
- hwloc\_set\_area\_membind\_nodest
  - hwlocality\_membinding, [70](#)
- hwloc\_set\_cpubind
  - hwlocality\_cpubinding, [61](#)
- hwloc\_set\_membind
  - hwlocality\_membinding, [70](#)
- hwloc\_set\_membind\_nodest
  - hwlocality\_membinding, [70](#)
- hwloc\_set\_proc\_cpubind
  - hwlocality\_cpubinding, [61](#)
- hwloc\_set\_proc\_membind
  - hwlocality\_membinding, [71](#)
- hwloc\_set\_proc\_membind\_nodest
  - hwlocality\_membinding, [71](#)
- hwloc\_set\_thread\_cpubind
  - hwlocality\_cpubinding, [61](#)
- hwloc\_topology\_check
  - hwlocality\_creation, [47](#)
- hwloc\_topology\_cpubind\_support, [111](#)
  - get\_proc\_cpubind, [111](#)
  - get\_thisproc\_cpubind, [111](#)
  - get\_thisthread\_cpubind, [111](#)
  - get\_thread\_cpubind, [111](#)
  - set\_proc\_cpubind, [112](#)
  - set\_thisproc\_cpubind, [112](#)
  - set\_thisthread\_cpubind, [112](#)
  - set\_thread\_cpubind, [112](#)
- hwloc\_topology\_destroy
  - hwlocality\_creation, [48](#)
- hwloc\_topology\_discovery\_support, [112](#)
  - pu, [112](#)
- hwloc\_topology\_export\_xml
  - hwlocality\_tinker, [53](#)
- hwloc\_topology\_export\_xmlbuffer
  - hwlocality\_tinker, [53](#)
- hwloc\_topology\_flags\_e
  - hwlocality\_configuration, [50](#)
- hwloc\_topology\_get\_allowed\_cpuset
  - hwlocality\_helper\_cpuset, [81](#)
- hwloc\_topology\_get\_allowed\_nodest
  - hwlocality\_helper\_nodest, [82](#)
- hwloc\_topology\_get\_complete\_cpuset
  - hwlocality\_helper\_cpuset, [81](#)
- hwloc\_topology\_get\_complete\_nodest
  - hwlocality\_helper\_nodest, [83](#)
- hwloc\_topology\_get\_depth
  - hwlocality\_information, [55](#)
- hwloc\_topology\_get\_online\_cpuset
  - hwlocality\_helper\_cpuset, [81](#)
- hwloc\_topology\_get\_support
  - hwlocality\_configuration, [50](#)
- hwloc\_topology\_get\_topology\_cpuset
  - hwlocality\_helper\_cpuset, [82](#)
- hwloc\_topology\_get\_topology\_nodest
  - hwlocality\_helper\_nodest, [83](#)
- hwloc\_topology\_ignore\_all\_keep\_structure
  - hwlocality\_configuration, [50](#)
- hwloc\_topology\_ignore\_type
  - hwlocality\_configuration, [50](#)
- hwloc\_topology\_ignore\_type\_keep\_structure
  - hwlocality\_configuration, [51](#)
- hwloc\_topology\_init
  - hwlocality\_creation, [48](#)
- hwloc\_topology\_insert\_misc\_object\_by\_  
cpuset
  - hwlocality\_tinker, [53](#)
- hwloc\_topology\_insert\_misc\_object\_by\_  
parent
  - hwlocality\_tinker, [53](#)
- hwloc\_topology\_is\_thissystem
  - hwlocality\_information, [55](#)
- hwloc\_topology\_load
  - hwlocality\_creation, [48](#)
- hwloc\_topology\_membind\_support, [113](#)
  - alloc\_membind, [113](#)
  - bind\_membind, [113](#)
  - firsttouch\_membind, [113](#)
  - get\_area\_membind, [113](#)
  - get\_proc\_membind, [114](#)
  - get\_thisproc\_membind, [114](#)
  - get\_thisthread\_membind, [114](#)
  - interleave\_membind, [114](#)
  - migrate\_membind, [114](#)
  - nexttouch\_membind, [114](#)
  - replicate\_membind, [114](#)

- set\_area\_membind, [114](#)
  - set\_proc\_membind, [114](#)
  - set\_thisproc\_membind, [114](#)
  - set\_thisthread\_membind, [114](#)
- hwloc\_topology\_set\_flags
  - hwlocality\_configuration, [51](#)
- hwloc\_topology\_set\_fsroot
  - hwlocality\_configuration, [51](#)
- hwloc\_topology\_set\_pid
  - hwlocality\_configuration, [51](#)
- hwloc\_topology\_set\_synthetic
  - hwlocality\_configuration, [52](#)
- hwloc\_topology\_set\_xml
  - hwlocality\_configuration, [52](#)
- hwloc\_topology\_set\_xmlbuffer
  - hwlocality\_configuration, [52](#)
- hwloc\_topology\_support, [115](#)
  - cpubind, [115](#)
  - discovery, [115](#)
  - membind, [115](#)
- hwloc\_topology\_t
  - hwlocality\_topology, [44](#)
- hwlocality\_configuration
  - HWLOC\_TOPOLOGY\_FLAG\_IS\_THISSYSTEM, [50](#)
  - HWLOC\_TOPOLOGY\_FLAG\_WHOLE\_SYSTEM, [50](#)
- hwlocality\_cpubinding
  - HWLOC\_CPUBIND\_NOMEMBIND, [60](#)
  - HWLOC\_CPUBIND\_PROCESS, [60](#)
  - HWLOC\_CPUBIND\_STRICT, [60](#)
  - HWLOC\_CPUBIND\_THREAD, [60](#)
- hwlocality\_information
  - HWLOC\_TYPE\_DEPTH\_MULTIPLE, [54](#)
  - HWLOC\_TYPE\_DEPTH\_UNKNOWN, [54](#)
- hwlocality\_membinding
  - HWLOC\_MEMBIND\_BIND, [65](#)
  - HWLOC\_MEMBIND\_DEFAULT, [64](#)
  - HWLOC\_MEMBIND\_FIRSTTOUCH, [64](#)
  - HWLOC\_MEMBIND\_INTERLEAVE, [65](#)
  - HWLOC\_MEMBIND\_MIGRATE, [64](#)
  - HWLOC\_MEMBIND\_MIXED, [65](#)
  - HWLOC\_MEMBIND\_NEXTTOUCH, [65](#)
  - HWLOC\_MEMBIND\_NOCPUBIND, [64](#)
  - HWLOC\_MEMBIND\_PROCESS, [64](#)
  - HWLOC\_MEMBIND\_REPLICATE, [65](#)
  - HWLOC\_MEMBIND\_STRICT, [64](#)
  - HWLOC\_MEMBIND\_THREAD, [64](#)
- hwlocality\_types
  - HWLOC\_OBJ\_CACHE, [46](#)
  - HWLOC\_OBJ\_CORE, [46](#)
  - HWLOC\_OBJ\_GROUP, [46](#)
  - HWLOC\_OBJ\_MACHINE, [46](#)
  - HWLOC\_OBJ\_MISC, [46](#)
  - HWLOC\_OBJ\_NODE, [46](#)
  - HWLOC\_OBJ\_PU, [46](#)
  - HWLOC\_OBJ\_SOCKET, [46](#)
  - HWLOC\_OBJ\_SYSTEM, [46](#)
  - HWLOC\_TYPE\_UNORDERED, [45](#)
- hwlocality\_api\_version
  - HWLOC\_API\_VERSION, [43](#)
  - hwloc\_get\_api\_version, [43](#)
- hwlocality\_bitmap
  - hwloc\_bitmap\_allbut, [88](#)
  - hwloc\_bitmap\_alloc, [88](#)
  - hwloc\_bitmap\_alloc\_full, [88](#)
  - hwloc\_bitmap\_and, [88](#)
  - hwloc\_bitmap\_andnot, [88](#)
  - hwloc\_bitmap\_asprintf, [88](#)
  - hwloc\_bitmap\_clr, [88](#)
  - hwloc\_bitmap\_clr\_range, [88](#)
  - hwloc\_bitmap\_compare, [88](#)
  - hwloc\_bitmap\_compare\_first, [89](#)
  - hwloc\_bitmap\_copy, [89](#)
  - hwloc\_bitmap\_dup, [89](#)
  - hwloc\_bitmap\_fill, [89](#)
  - hwloc\_bitmap\_first, [89](#)
  - hwloc\_bitmap\_foreach\_begin, [87](#)
  - hwloc\_bitmap\_foreach\_end, [87](#)
  - hwloc\_bitmap\_free, [89](#)
  - hwloc\_bitmap\_from\_ith\_ulong, [89](#)
  - hwloc\_bitmap\_from\_ulong, [89](#)
  - hwloc\_bitmap\_intersects, [90](#)
  - hwloc\_bitmap\_isequal, [90](#)
  - hwloc\_bitmap\_isfull, [90](#)
  - hwloc\_bitmap\_isincluded, [90](#)
  - hwloc\_bitmap\_isset, [90](#)
  - hwloc\_bitmap\_iszero, [90](#)
  - hwloc\_bitmap\_last, [90](#)
  - hwloc\_bitmap\_next, [90](#)
  - hwloc\_bitmap\_not, [91](#)

- hwloc\_bitmap\_only, 91
- hwloc\_bitmap\_or, 91
- hwloc\_bitmap\_set, 91
- hwloc\_bitmap\_set\_ith\_ulong, 91
- hwloc\_bitmap\_set\_range, 91
- hwloc\_bitmap\_singlify, 91
- hwloc\_bitmap\_snprintf, 91
- hwloc\_bitmap\_sscanf, 92
- hwloc\_bitmap\_t, 87
- hwloc\_bitmap\_taskset\_asprintf, 92
- hwloc\_bitmap\_taskset\_snprintf, 92
- hwloc\_bitmap\_taskset\_sscanf, 92
- hwloc\_bitmap\_to\_ith\_ulong, 92
- hwloc\_bitmap\_to\_ulong, 92
- hwloc\_bitmap\_weight, 92
- hwloc\_bitmap\_xor, 93
- hwloc\_bitmap\_zero, 93
- hwloc\_const\_bitmap\_t, 87
- hwlocality\_configuration
  - hwloc\_topology\_flags\_e, 50
  - hwloc\_topology\_get\_support, 50
  - hwloc\_topology\_ignore\_all\_keep\_structure, 50
  - hwloc\_topology\_ignore\_type, 50
  - hwloc\_topology\_ignore\_type\_keep\_-structure, 51
  - hwloc\_topology\_set\_flags, 51
  - hwloc\_topology\_set\_fsroot, 51
  - hwloc\_topology\_set\_pid, 51
  - hwloc\_topology\_set\_synthetic, 52
  - hwloc\_topology\_set\_xml, 52
  - hwloc\_topology\_set\_xmlbuffer, 52
- hwlocality\_conversion
  - hwloc\_obj\_attr\_snprintf, 56
  - hwloc\_obj\_cpuset\_snprintf, 57
  - hwloc\_obj\_get\_info\_by\_name, 57
  - hwloc\_obj\_snprintf, 57
  - hwloc\_obj\_type\_of\_string, 57
  - hwloc\_obj\_type\_snprintf, 58
  - hwloc\_obj\_type\_string, 58
- hwlocality\_cpubinding
  - hwloc\_cpubind\_flags\_t, 59
  - hwloc\_get\_cpubind, 60
  - hwloc\_get\_proc\_cpubind, 60
  - hwloc\_get\_thread\_cpubind, 60
  - hwloc\_set\_cpubind, 61
  - hwloc\_set\_proc\_cpubind, 61
  - hwloc\_set\_thread\_cpubind, 61
- hwlocality\_creation
  - hwloc\_topology\_check, 47
  - hwloc\_topology\_destroy, 48
  - hwloc\_topology\_init, 48
  - hwloc\_topology\_load, 48
- hwlocality\_cuda
  - hwloc\_cuda\_get\_device\_cpuset, 99
- hwlocality\_cudart
  - hwloc\_cudart\_get\_device\_cpuset, 99
- hwlocality\_glibc\_sched
  - hwloc\_cpuset\_from\_glibc\_sched\_affinity, 93
  - hwloc\_cpuset\_to\_glibc\_sched\_affinity, 93
- hwlocality\_helper\_binding
  - hwloc\_alloc\_membind\_policy, 80
  - hwloc\_alloc\_membind\_policy\_nodeset, 80
  - hwloc\_distribute, 80
  - hwloc\_distributev, 80
- hwlocality\_helper\_cpuset
  - hwloc\_topology\_get\_allowed\_cpuset, 81
  - hwloc\_topology\_get\_complete\_cpuset, 81
  - hwloc\_topology\_get\_online\_cpuset, 81
  - hwloc\_topology\_get\_topology\_cpuset, 82
- hwlocality\_helper\_find\_cache
  - hwloc\_get\_cache\_covering\_cpuset, 78
  - hwloc\_get\_shared\_cache\_covering\_-obj, 78
- hwlocality\_helper\_find\_covering
  - hwloc\_get\_child\_covering\_cpuset, 76
  - hwloc\_get\_obj\_covering\_cpuset, 76
- hwlocality\_helper\_find\_coverings
  - hwloc\_get\_next\_obj\_covering\_cpuset\_-by\_depth, 77
  - hwloc\_get\_next\_obj\_covering\_cpuset\_-by\_type, 77
- hwlocality\_helper\_find\_inside
  - hwloc\_get\_first\_largest\_obj\_inside\_-cpuset, 74
  - hwloc\_get\_largest\_objs\_inside\_cpuset, 75
  - hwloc\_get\_nbobjs\_inside\_cpuset\_by\_-depth, 75
  - hwloc\_get\_nbobjs\_inside\_cpuset\_by\_-type, 75
  - hwloc\_get\_next\_obj\_inside\_cpuset\_-by\_depth, 75



- hwloc\_get\_next\_obj\_inside\_cpuset\_by\_type, 75
- hwloc\_get\_obj\_inside\_cpuset\_by\_depth, 75
- hwloc\_get\_obj\_inside\_cpuset\_by\_type, 76
- hwlocality\_helper\_nodeset
  - hwloc\_topology\_get\_allowed\_nodeset, 82
  - hwloc\_topology\_get\_complete\_nodeset, 83
  - hwloc\_topology\_get\_topology\_nodeset, 83
- hwlocality\_helper\_nodeset\_convert
  - hwloc\_cpuset\_from\_nodeset, 84
  - hwloc\_cpuset\_from\_nodeset\_strict, 84
  - hwloc\_cpuset\_to\_nodeset, 84
  - hwloc\_cpuset\_to\_nodeset\_strict, 85
- hwlocality\_helper\_traversal
  - hwloc\_get\_closest\_objs, 78
  - hwloc\_get\_obj\_below\_array\_by\_type, 78
  - hwloc\_get\_obj\_below\_by\_type, 79
- hwlocality\_helper\_traversal\_basic
  - hwloc\_get\_ancestor\_obj\_by\_depth, 72
  - hwloc\_get\_ancestor\_obj\_by\_type, 72
  - hwloc\_get\_common\_ancestor\_obj, 73
  - hwloc\_get\_next\_child, 73
  - hwloc\_get\_next\_obj\_by\_depth, 73
  - hwloc\_get\_next\_obj\_by\_type, 73
  - hwloc\_get\_pu\_obj\_by\_os\_index, 73
  - hwloc\_get\_root\_obj, 73
  - hwloc\_obj\_is\_in\_subtree, 74
- hwlocality\_helper\_types
  - hwloc\_get\_type\_or\_above\_depth, 72
  - hwloc\_get\_type\_or\_below\_depth, 72
- hwlocality\_information
  - hwloc\_get\_depth\_type, 54
  - hwloc\_get\_nbobjs\_by\_depth, 54
  - hwloc\_get\_nbobjs\_by\_type, 55
  - hwloc\_get\_type\_depth, 55
  - hwloc\_get\_type\_depth\_e, 54
  - hwloc\_topology\_get\_depth, 55
  - hwloc\_topology\_is\_thissystem, 55
- hwlocality\_linux
  - hwloc\_linux\_get\_tid\_cpubind, 94
  - hwloc\_linux\_parse\_cpumap\_file, 94
  - hwloc\_linux\_set\_tid\_cpubind, 94
- hwlocality\_linux\_libnuma\_bitmask
  - hwloc\_cpuset\_from\_linux\_libnuma\_bitmask, 96
  - hwloc\_cpuset\_to\_linux\_libnuma\_bitmask, 96
  - hwloc\_nodeset\_from\_linux\_libnuma\_bitmask, 97
  - hwloc\_nodeset\_to\_linux\_libnuma\_bitmask, 97
- hwlocality\_linux\_libnuma\_nodemask
  - hwloc\_cpuset\_from\_linux\_libnuma\_nodemask, 98
  - hwloc\_cpuset\_to\_linux\_libnuma\_nodemask, 98
  - hwloc\_nodeset\_from\_linux\_libnuma\_nodemask, 98
  - hwloc\_nodeset\_to\_linux\_libnuma\_nodemask, 98
- hwlocality\_linux\_libnuma\_ulongs
  - hwloc\_cpuset\_from\_linux\_libnuma\_ulongs, 95
  - hwloc\_cpuset\_to\_linux\_libnuma\_ulongs, 95
  - hwloc\_nodeset\_from\_linux\_libnuma\_ulongs, 95
  - hwloc\_nodeset\_to\_linux\_libnuma\_ulongs, 96
- hwlocality\_membinding
  - hwloc\_alloc, 65
  - hwloc\_alloc\_membind, 65
  - hwloc\_alloc\_membind\_nodeset, 66
  - hwloc\_free, 66
  - hwloc\_get\_area\_membind, 66
  - hwloc\_get\_area\_membind\_nodeset, 66
  - hwloc\_get\_membind, 67
  - hwloc\_get\_membind\_nodeset, 68
  - hwloc\_get\_proc\_membind, 68
  - hwloc\_get\_proc\_membind\_nodeset, 69
  - hwloc\_membind\_flags\_t, 63
  - hwloc\_membind\_policy\_t, 64
  - hwloc\_set\_area\_membind, 69
  - hwloc\_set\_area\_membind\_nodeset, 70
  - hwloc\_set\_membind, 70
  - hwloc\_set\_membind\_nodeset, 70
  - hwloc\_set\_proc\_membind, 71
  - hwloc\_set\_proc\_membind\_nodeset, 71
- hwlocality\_myriexpress
  - hwloc\_mx\_board\_get\_device\_cpuset, 100
  - hwloc\_mx\_endpoint\_get\_device\_cpuset, 100



- hwlocality\_objects
  - hwloc\_obj\_t, [47](#)
- hwlocality\_openfabrics
  - hwloc\_ibv\_get\_device\_cpuset, [99](#)
- hwlocality\_sets
  - hwloc\_const\_cpuset\_t, [44](#)
  - hwloc\_const\_nodeset\_t, [44](#)
  - hwloc\_cpuset\_t, [44](#)
  - hwloc\_nodeset\_t, [44](#)
- hwlocality\_tinker
  - hwloc\_topology\_export\_xml, [53](#)
  - hwloc\_topology\_export\_xmlbuffer, [53](#)
  - hwloc\_topology\_insert\_misc\_object\_-by\_cpuset, [53](#)
  - hwloc\_topology\_insert\_misc\_object\_-by\_parent, [53](#)
- hwlocality\_topology
  - hwloc\_topology\_t, [44](#)
- hwlocality\_traversal
  - hwloc\_get\_obj\_by\_depth, [56](#)
  - hwloc\_get\_obj\_by\_type, [56](#)
- hwlocality\_types
  - hwloc\_compare\_types, [46](#)
  - hwloc\_compare\_types\_e, [45](#)
  - hwloc\_obj\_type\_t, [45](#)
- infos
  - hwloc\_obj, [105](#)
- infos\_count
  - hwloc\_obj, [105](#)
- interleave\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- last\_child
  - hwloc\_obj, [105](#)
- linesize
  - hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_-s, [101](#)
- Linux-only helpers, [94](#)
- local\_memory
  - hwloc\_obj\_memory\_s, [110](#)
- logical\_index
  - hwloc\_obj, [105](#)
- membind
  - hwloc\_topology\_support, [115](#)
- memory
  - hwloc\_obj, [106](#)
- Memory binding, [62](#)
- migrate\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- name
  - hwloc\_obj, [106](#)
  - hwloc\_obj\_info\_s, [109](#)
- next\_cousin
  - hwloc\_obj, [106](#)
- next\_sibling
  - hwloc\_obj, [106](#)
- nexttouch\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- nodeset
  - hwloc\_obj, [106](#)
- Nodeset Helpers, [82](#)
- Object sets (hwloc\_cpuset\_t and hwloc\_-nodeset\_t), [44](#)
- Object Type Helpers, [71](#)
- Object/String Conversion, [56](#)
- online\_cpuset
  - hwloc\_obj, [106](#)
- OpenFabrics-Specific Functions, [99](#)
- os\_index
  - hwloc\_obj, [107](#)
- os\_level
  - hwloc\_obj, [107](#)
- page\_types
  - hwloc\_obj\_memory\_s, [110](#)
- page\_types\_len
  - hwloc\_obj\_memory\_s, [110](#)
- parent
  - hwloc\_obj, [107](#)
- prev\_cousin
  - hwloc\_obj, [107](#)
- prev\_sibling
  - hwloc\_obj, [107](#)
- pu
  - hwloc\_topology\_discovery\_support, [112](#)
- replicate\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- Retrieve Objects, [55](#)
- set\_area\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- set\_proc\_cpupbind
  - hwloc\_topology\_cpupbind\_support, [112](#)
- set\_proc\_membind
  - hwloc\_topology\_membind\_support, [114](#)

- set\_thisproc\_cpubind
  - hwloc\_topology\_cpubind\_support, [112](#)
- set\_thisproc\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- set\_thisthread\_cpubind
  - hwloc\_topology\_cpubind\_support, [112](#)
- set\_thisthread\_membind
  - hwloc\_topology\_membind\_support, [114](#)
- set\_thread\_cpubind
  - hwloc\_topology\_cpubind\_support, [112](#)
- sibling\_rank
  - hwloc\_obj, [107](#)
- size
  - hwloc\_obj\_attr\_u::hwloc\_cache\_attr\_  
s, [101](#)
  - hwloc\_obj\_memory\_s::hwloc\_obj\_memory\_  
page\_type\_s, [109](#)
- The bitmap API, [85](#)
- Tinker with topologies., [53](#)
- Topology context, [43](#)
- Topology Object Types, [45](#)
- Topology Objects, [47](#)
- total\_memory
  - hwloc\_obj\_memory\_s, [110](#)
- type
  - hwloc\_obj, [107](#)
- userdata
  - hwloc\_obj, [107](#)
- value
  - hwloc\_obj\_info\_s, [109](#)