NAME

slurm_allocate_pack_job_blocking, slurm_allocate_resources, slurm_allocate_resources_blocking, slurm_allocation_msg_thr_create, slurm_allocation_msg_thr_destroy, slurm_allocation_lookup, slurm_pack_job_lookup, slurm_confirm_allocation, slurm_free_submit_response_response_msg, slurm_init_job_desc_msg, slurm_job_will_run, slurm_pack_job_will_run, slurm_job_will_run2, slurm_read_hostfile, slurm_submit_batch_job, slurm_submit_batch_pack_job − Slurm job initiation functions

SYNTAX

#include <slurm/slurm.h>

int **slurm_allocate_resources** (
job_desc_msg_t **job_desc_msg_ptr*,
resource_allocation_response_msg_t ***slurm_alloc_msg_pptr*
);

resource_allocation_response_msg_t ***slurm_allocate_resources_blocking** (
job_desc_msg_t **job_desc_msg_ptr*,
time_t *timeout*, void *(*pending_callback)(uint32_t job_id)*
);

List ***slurm_allocate_pack_job_blocking** (
List *job_desc_msg_list*,
time_t *timeout*, void *(*pending_callback)(uint32_t job_id)*
);

allocation_msg_thread_t ***slurm_allocation_msg_thr_create** (
uint16_t **port*,
slurm_allocation_callbacks_t **callbacks*
);

void ***slurm_allocation_msg_thr_destroy** (
allocation_msg_thread_t **slurm_alloc_msg_thr_ptr*
);

int **slurm_allocation_lookup** {
uint32_t *jobid*,
resource_allocation_response_msg_t ***slurm_alloc_msg_pptr*
);

int **slurm_pack_job_lookup** {
uint32_t *jobid*,
List **slurm_alloc_msg_list*
);

int **slurm_confirm_allocation** (
old_job_alloc_msg_t **old_job_desc_msg_ptr*,
resource_allocation_response_msg_t ***slurm_alloc_msg_pptr*
);

void **slurm_free_resource_allocation_response_msg** (
resource_allocation_response_msg_t **slurm_alloc_msg_ptr*
);

void **slurm_free_submit_response_response_msg** (
submit_response_msg_t **slurm_submit_msg_ptr*
);

void **slurm_init_job_desc_msg** (
job_desc_msg_t **job_desc_msg_ptr*
);

int **slurm_job_will_run** (
job_desc_msg_t **job_desc_msg_ptr*
);

int **slurm_pack_job_will_run** (

List *job_desc_msg_list*
);

int **slurm_job_will_run2** (
job_desc_msg_t **job_desc_msg_ptr*,
will_run_response_msg_t ***will_run_resp*
);

int **slurm_read_hostfile** (
const char **filename*, int *n*
);

int **slurm_submit_batch_job** (
job_desc_msg_t **job_desc_msg_ptr*,
submit_response_msg_t ***slurm_submit_msg_pptr*
);

int **slurm_submit_batch_pack_job** (
List *job_desc_msg_list*,
submit_response_msg_t ***slurm_submit_msg_pptr*
);

ARGUMENTS

job_desc_msg_list
List of job request specifications (of type job_desc_msg_t) for a heterogeneous job in an ordered list. See slurm.h for full details on the data structure's contents.

job_desc_msg_ptr
Specifies the pointer to a job request specification. See slurm.h for full details on the data structure's contents.

callbacks
Specifies the pointer to a allocation callbacks structure. See slurm.h for full details on the data structure's contents.

old_job_desc_msg_ptr
Specifies the pointer to a description of an existing job. See slurm.h for full details on the data structure's contents.

slurm_alloc_msg_list
Specifies a pointer to a List structure to be created and filled with a list of pointers to resource allocation data (of type resource_allocation_response_msg_t).

slurm_alloc_msg_pptr
Specifies the double pointer to the structure to be created and filled with a description of the created resource allocation (job): job ID, list of allocated nodes, processor count per allocated node, etc. See slurm.h for full details on the data structure's contents.

slurm_alloc_msg_ptr
Specifies the pointer to the structure to be created and filled in by the function *slurm_allocate_resources*, *slurm_allocate_resources_blocking*, *slurm_allocation_lookup*, *slurm_confirm_allocation*, *slurm_job_will_run* or *slurm_job_will_run*.

slurm_alloc_msg_thr_ptr
Specifies the pointer to the structure created and returned by the function *slurm_allocation_msg_thr_create*. Must be destroyed with function *slurm_allocation_msg_thr_destroy*.

slurm_submit_msg_pptr
Specifies the double pointer to the structure to be created and filled with a description of the created job: job ID, etc. See slurm.h for full details on the data structure's contents.

slurm_submit_msg_ptr
Specifies the pointer to the structure to be created and filled in by the function *slurm_submit_batch_job*.

will_run_resp
Specifies when and where the specified job descriptor could be started.

DESCRIPTION

slurm_allocate_resources Request a resource allocation for a job. If successful, a job entry is created. Note that if the job's requested node count or time allocation are outside of the partition's limits then a job entry will be created, a warning indication will be placed in the *error_code* field of the response message, and the job will be left queued until the partition's limits are changed. Always release the response message when no longer required using the function **slurm_free_resource_allocation_response_msg**. This function only makes the request once. If the allocation is not available immediately the node_cnt variable in the resp will be 0. If you want a function that will block until either an error is received or an allocation is granted you can use the *slurm_allocate_resources_blocking* function described below.

slurm_allocate_resources_blocking Request a resource allocation for a job. This call will block until the allocation is granted, an error occurs, or the specified timeout limit is reached. The *pending_callback* parameter will be called if the allocation is not available immediately and the immediate flag is not set in the request. This can be used to get the jobid of the job while waiting for the allocation to become available. On failure NULL is returned and errno is set.

slurm_allocate_pack_job_blocking Request a set of resource allocations for a heterogeneous job. This call will block until the allocation is granted, an error occurs, or the specified timeout limit is reached. The *pending_callback* parameter will be called if the allocation is not available immediately and the immediate flag is not set in the request. This can be used to get the jobid of the job while waiting for the allocation to become available. On failure NULL is returned and errno is set. The returned list should be freed using the **list_destroy** function.

slurm_allocation_msg_thr_create Startup a message handler talking with the controller dealing with messages from the controller during an allocation. Callback functions are declared in the *callbacks* parameter and will be called when a corresponding message is received from the controller. This message thread is needed to receive messages from the controller about node failure in an allocation and other important messages. Although technically not required, it could be very helpful to inform about problems with the allocation.

slurm_allocation_msg_thr_destroy Shutdown the message handler
talking with the controller dealing with messages from the controller during
an allocation.

slurm_confirm_allocation Return detailed information on a specific existing job allocation. **OBSOLETE FUNCTION: Use slurm_allocation_lookup instead.** This function may only be successfully executed by the job's owner or user root.

slurm_allocation_lookup Returns detailed information about an existing job allocation.

slurm_pack_job_lookup Returns detailed information about an existing heterogeneous job allocation. Each element in the list represents a component of the job in sequential order. The returned list should be freed using the **list_destroy** function.

slurm_free_resource_allocation_response_msg Release the storage generated in response to a call of the function **slurm_allocate_resources** or **slurm_allocation_lookup**.

slurm_free_submit_response_msg Release the storage generated in response to a call of the function **slurm_submit_batch_job**.

slurm_init_job_desc_msg Initialize the contents of a job descriptor with default values. Execute this function before issuing a request to submit or modify a job.

slurm_job_will_run Report when and where the supplied job description can be executed.

slurm_pack_job_will_run Report when and where the supplied heterogeneous job description can be executed.

slurm_job_will_run2 Determine when and where the supplied job description can be executed.

slurm_read_hostfile Read a Slurm hostfile specified by "filename". "filename" must contain a list of Slurm NodeNames, one per line. Reads up to "n" number of hostnames from the file. Returns a string representing a hostlist ranged string of the contents of the file. This is a helper function, it does not contact any Slurm daemons.

slurm_submit_batch_job Submit a job for later execution. Note that if the job's requested node count or time allocation are outside of the partition's limits then a job entry will be created, a warning

indication will be placed in the *error_code* field of the response message, and the job will be left queued until the partition's limits are changed and resources are available. Always release the response message when no longer required using the function **slurm_free_submit_response_msg**.

slurm_submit_batch_pack_job Submit a heterogeneous job for later execution. Note that if the job's requested node count or time allocation are outside of the partition's limits then a job entry will be created, a warning indication will be placed in the *error_code* field of the response message, and the job will be left queued until the partition's limits are changed and resources are available. Always release the response message when no longer required using the function **slurm_free_submit_response_msg**.

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and Slurm error code is set appropriately.

ERRORS

SLURM_PROTOCOL_VERSION_ERROR Protocol version has changed, re−link your code.

ESLURM_CAN_NOT_START_IMMEDIATELY the job can not be started immediately as requested.

ESLURM_DEFAULT_PARTITION_NOT_SET the system lacks a valid default partition.

ESLURM_JOB_MISSING_PARTITION_KEY use of this partition is restricted through a credential provided only to user root. This job lacks such a valid credential.

ESLURM_JOB_MISSING_REQUIRED_PARTITION_GROUP use of this partition is restricted to certain groups. This user is not a member of an authorized group.

ESLURM_REQUESTED_NODES_NOT_IN_PARTITION the job requested use of specific nodes which are not in the requested (or default) partition.

ESLURM_TOO_MANY_REQUESTED_CPUS the job requested use of more processors than can be made available to in the requested (or default) partition.

ESLURM_TOO_MANY_REQUESTED_NODES the job requested use of more nodes than can be made available to in the requested (or default) partition.

ESLURM_ERROR_ON_DESC_TO_RECORD_COPY unable to create the job due to internal resources being exhausted. Try again later.

ESLURM_JOB_MISSING_SIZE_SPECIFICATION the job failed to specify some size specification. At least one of the following must be supplied: required processor count, required node count, or required node list.

ESLURM_JOB_SCRIPT_MISSING failed to identify executable program to be queued.

ESLURM_USER_ID_MISSING identification of the job's owner was not provided.

ESLURM_DUPLICATE_JOB_ID the requested job id is already in use.

ESLURM_NOT_TOP_PRIORITY job can not be started immediately because higher priority jobs are waiting to use this partition.

ESLURM_NOT_PACK_JOB_LEADER the job ID does not represent a heterogeneous job leader as required by the function.

ESLURM_REQUESTED_NODE_CONFIG_UNAVAILABLE the requested node configuration is not available (at least not in sufficient quantity) to satisfy the request.

ESLURM_REQUESTED_PART_CONFIG_UNAVAILABLE the requested partition configuration is not available to satisfy the request. This is not a fatal error, but indicates that the job will be left queued until the partition's configuration is changed. This typically indicates that the job's requested node count is outside of the node count range its partition is configured to support (e.g. the job wants 64 nodes and the partition will only schedule jobs using between 1 and 32 nodes). Alternately, the job's time limit exceeds the partition's time limit.

ESLURM_NODES_BUSY the requested nodes are already in use.

ESLURM_INVALID_FEATURE the requested feature(s) does not exist.

ESLURM_INVALID_JOB_ID the requested job id does not exist.

ESLURM_INVALID_NODE_COUNT the requested node count is not valid.

ESLURM_INVALID_NODE_NAME the requested node name(s) is/are not valid.

ESLURM_INVALID_PARTITION_NAME the requested partition name is not valid.

ESLURM_TRANSITION_STATE_NO_UPDATE the requested job configuration change can not take place at this time. Try again later.

ESLURM_ALREADY_DONE the specified job has already completed and can not be modified.

ESLURM_ACCESS_DENIED the requesting user lacks authorization for the requested action (e.g. trying to delete or modify another user's job).

ESLURM_INTERCONNECT_FAILURE failed to configure the node interconnect.

ESLURM_BAD_DIST task distribution specification is invalid.

SLURM_PROTOCOL_SOCKET_IMPL_TIMEOUT Timeout in communicating with Slurm controller.

NON-BLOCKING EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <slurm/slurm.h>
#include <slurm/slurm_errno.h>

int main (int argc, char *argv[])
{
        job_desc_msg_t job_desc_msg;
        resource_allocation_response_msg_t* slurm_alloc_msg_ptr ;

        slurm_init_job_desc_msg( &job_desc_msg );
        job_desc_msg. name = ("job01  ");
        job_desc_msg. job_min_memory = 1024;
        job_desc_msg. time_limit = 200;
        job_desc_msg. min_nodes = 400;
        job_desc_msg. user_id = getuid();
        job_desc_msg. group_id = getgid();
        if (slurm_allocate_resources(&job_desc_msg,
                        &slurm_alloc_msg_ptr)) {
                slurm_perror ("slurm_allocate_resources error");
                exit (1);
        }
        printf ("Allocated nodes %s to job_id %u\n",
              slurm_alloc_msg_ptr−>node_list,
              slurm_alloc_msg_ptr−>job_id );
        if (slurm_kill_job(slurm_alloc_msg_ptr−>job_id, SIGKILL, 0)) {
                printf ("kill errno %d\n", slurm_get_errno());
                exit (1);
        }
        printf ("canceled job_id %u\n",
              slurm_alloc_msg_ptr−>job_id );
        slurm_free_resource_allocation_response_msg(
                        slurm_alloc_msg_ptr);
        exit (0);
}
```

BLOCKING EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <slurm/slurm.h>
#include <slurm/slurm_errno.h>

int main (int argc, char *argv[])
```

```
{
        job_desc_msg_t job_desc_msg;
        resource_allocation_response_msg_t* slurm_alloc_msg_ptr ;

        slurm_init_job_desc_msg( &job_desc_msg );
        job_desc_msg. name = ("job01  ");
        job_desc_msg. job_min_memory = 1024;
        job_desc_msg. time_limit = 200;
        job_desc_msg. min_nodes = 400;
        job_desc_msg. user_id = getuid();
        job_desc_msg. group_id = getgid();
        if (!(slurm_alloc_msg_ptr =
            slurm_allocate_resources_blocking(&job_desc_msg, 0, NULL))) {
                slurm_perror ("slurm_allocate_resources_blocking error");
                exit (1);
        }
        printf ("Allocated nodes %s to job_id %u\n",
               slurm_alloc_msg_ptr−>node_list,
               slurm_alloc_msg_ptr−>job_id );
        if (slurm_kill_job(slurm_alloc_msg_ptr−>job_id, SIGKILL, 0)) {
                printf ("kill errno %d\n", slurm_get_errno());
                exit (1);
        }
        printf ("canceled job_id %u\n",
               slurm_alloc_msg_ptr−>job_id );
        slurm_free_resource_allocation_response_msg(
                                slurm_alloc_msg_ptr);
        exit (0);
}
```

NOTE

These functions are included in the libslurm library, which must be linked to your process for use (e.g. "cc −lslurm myprog.c").

COPYING

SEE ALSO

hostlist_create(3), **hostlist_shift**(3), **hostlist_destroy**(3), **scancel**(1), **srun**(1), **slurm_free_job_info_msg**(3), **slurm_get_errno**(3), **slurm_load_jobs**(3), **slurm_perror**(3), **slurm_strerror**(3)