|  |  |  |
| --- | --- | --- |
| Name | Input Parameters | Output Parameters |
| rn\_creatern\_identrn\_deletern\_getsegrn\_retseg | name paddr length pagesise flagsnamernidrnid sise flagsrnid segaddr timeout | &rnid &bytes&rnid&segaddr |
| pt\_creatept\_identpt\_deletept\_getbufpt\_retbuf | name paddr length bsise flagsname nodeptidptidptid bufaddr | &ptid &bnum&ptid&bufaddr |
| mm\_l2pmm\_p2lmm\_pmapmm\_unmapmm\_preadmm\_pwritemm\_ptcreate | tid laddrtid paddrtid laddr paddr length flagstid laddrpaddr laddr lengthpaddr laddr lengthname paddr length bsise laddr flags | &paddr &length&laddr &length&ptid &bnum |
| m\_ext2intm\_int2ext | externalinternal | &internal&external |

TABLE 2. Directive Usage

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Remote | ISR | ISR to Remote |
| t\_createt\_identt\_startt\_restartt\_deletet\_suspendt\_resumet\_setprit\_modet\_getregt\_setreg | noyesnononoyesyesyesnoyesyes | noyesnonononoyesnonoyesyes | -yes----no--nono |
| q\_createq\_identq\_deleteq\_sendq\_urgentq\_broadcastq\_receive | noyesnoyesyesyesyes | noyesnoyesyesyesyes | -yes-nononono |
| ev\_sendev\_receive | yesno | yesno | no- |
| as\_catchas\_sendas\_return | noyesno | noyesno | -no- |
| sm\_createsm\_identsm\_deletesm\_psm\_v | noyesnoyesyes | noyesnoyesyes | -yes-nono |
| tm\_settm\_gettm\_wkaftertm\_wkwhentm\_evaftertm\_evwhentm\_canceltm\_tick | nononononononono | yesyesnononononoyes | nono-----no |
| i\_return | no | yes | - |
| k\_fatal | no | yes | - |

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Remote | ISR | ISR to Remote |
| rn\_creatern\_identrn\_deletern\_getsegrn\_retseg | nonononono | noyesnonono | -yes--- |
| pt\_creatept\_identpt\_deletept\_getbufpt\_retbuf | noyesnoyesyes | noyesnoyesyes | -yes-yesyes |
| mm\_l2pmm\_p2lmm\_pmapmm\_unmapmm\_preadmm\_pwritemm\_ptcreate | nonononononono | yesnoyesyesnonono | no-nono--- |
| m\_ext2intm\_int2ext | nono | yesyes | nono |

3.1 Task Management

A task is a function that can execute concurrently with other functions within a multitasking environment. A task typically accepts one or more inputs, performs some processing function based on the input, and responds with one or more outputs.

A task is created using the *t\_create* directive. Once a task is created, other tasks can refer to it and act on its behalf in allocating resources to it. A task is started with the *t\_start* directive. Once a task has been restarted, it can execute its function and vie with other tasks for processor time according to its relative priority.

A task may be deleted with the *t\_delete* directive. All knowledge of the task is removed from the system, and other tasks referring to it will be returned an error.

All tasks have a task identifier (tid). The *tid* is assigned to the task at creation time, and must be used in all subsequent calls to the executive to identify that task. The *t\_ident* directive may be used to obtain the *tid* of another task when the task name is known.

All tasks have a priority. A task’s priority is a measure of the task’s importance relative to all other tasks within the system and indicate its “need to run” in a multitasking environment where many tasks may be ready to run at any moment. A task is given a priority at creation time. A task’s priority may be changes with the *t\_setpri* directive.

A task’s mode of execution is set up initially with the *t\_start* directive, and may be changed using the *t\_mode* directive. The mode of a task specifies its ability to be preempted, timesliced, to execute in user mode, to execute in supervisor mode at an optional interrupt level, and to disable/enable its asynchronous signal routine.

The task manager provide the pair of directives, *t\_suspend* and *t\_resume*, to control execution of another task.

A task is provided with a set of eight user and eight system defined software registers which may be set with the *t\_setreg* directive, and read with the *t\_getreg* directive.

The directives provided by the task manager are:

|  |  |
| --- | --- |
| Directive | Function |
| t\_createt\_identt\_deletet\_startt\_restartt\_suspendt\_resumet\_setprit\_modet\_getregt\_setreg | Create a taskObtain id of a taskDelete a taskStart a taskRestart a taskSuspend a taskResume a taskSet task priorityChange task modeGet task registerSet task register |

3.1.1 T\_CREATE

NAME

t\_create – “Create a Task”

SYNOPSIS

uint t\_create ( name, superstk, userstk, priority, flags, &tid )

uint name; /\* user defined 4-byte task name \*/

uint superstk; /\* supervisor stack sise in bytes \*/

uint userstk; /\* user stack sise in bytes \*/

uint priority; /\* task priority \*/

uint flags; /\* task attributes \*/

uint tid; /\* task id – return by the call \*/

*Flags* is defined as follows:

CMASK Coprocessor mask

0 = no coprocessor

CLOBAL set to indicate the task is a multiprocessor global resource.

 clear to indicate the task is local

DESCRIPTION

The *t\_create* directive creates a task by allocating and initialising a task data structure. A task is created by name. A task id is returned to the caller in the *tid* field. The *tid* must be used in all calls to the executive requiring a *tid*.

The task is allocated a user stack and supervisor stack as determined by the values in the *userstk* and *superstk* fields. A minimum supervisor stack is required, and an error will be returned if the *superstk* value is too small. There is no minimum user stack required.

By setting the GLOBAL value in the flags field, the *tid* will be sent to all processors in the sys-tem, to be entered into a global resource table. The system is defined as the collection of inter-connected processors. The task is always created on the local node.

The newly created task will be placed in the dormant state. The *t\_start* directive will make the task ready, in priority order. The executive will support a minimum of 32 priorities.

The maximum number of tasks is a configuration parameter.

RETURN VALUE

If *t\_create* successfully created a task, the *tid* is filled in, and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Too many tasks.

No more memory for stack(s) segment.

*Supertik* too small.

NOTES

Not callable from ISR.

Will not cause a preempt.

3.1.2 T\_IDENT

NAME

t\_ident = “Obtain id of task”

SYNOPSIS

uint t\_ident ( name, node, &tid )

uint name; /\* user defined 4-byte task name \*/

 /\* 0 indicates requesting task \*/

uint node; /\* node identifier \*/
/\* 0 indicates any node \*/

uint tid; /\* task id – returned by this call \*/

DESCRIPTION

This directive allows a task to obtain the *tid* of itself or another task in the system. The *tid* must then be used in all calls to the executive requiring a *tid*.

If the task name is not unique, the *tid* returned may not correspond to the task named in this call.

The task identifies by its name may exist on the local processor or any remote processor in a multiprocessor configuration, as long as the task was created with the GLOBAL flags value set (see *t\_create*). If the task names is not unique within the multiprocessor configuration, a non-sero node identifier must be specified in the node field.

RETURN VALUE

If *t\_ident* succeeded, the *tid* is filled in, and 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Task with this name does not exist

Invalid node identifier.

NOTES

Can be called from within an ISR.

Will not cause a preempt.

3.1.3 T\_START

NAME

t\_start – “Start a Task”

SYNOPSIS

uint t\_start ( tid, saddr, mode, argp )

uint tid; /\* task id as returned from t\_create or t\_ident \*/

ptf saddr; /\* start execution address of task \*/

uint mode; /\* initial mode value of task \*/

long (\*argp)[4]; /\* pointer to argument list \*/

The *mode* value is defined as follows:

NOPREEMPT set to disable preempting
clear to enable preempting

TSLICE set to enable timeslicing
clear to disable timeslicing

NOASR set to disable asynchronous signal processing
clear to enable asynchronous signal processing

SUPV set to execute in supervisor mode
clear to execute in user mode

LEVEL interrupt level when SUPV is set

DESCRIPTION

The task identified by the *tid* is made ready, based on its current priority, to await execution. A task can be started only from the dormant state.

*Saddr* is the logical address where the task wants to start execution. *Mode* contains the flag values to enable/disable preempting, timeslicing, asynchronous processing, supervisor mode and an optional interrupt level when the task starts execution.

*Argp* is a pointer to a list of four arguments. These arguments are pushed onto the stack of the task being started. A fifth argument, the executive’s fatal error handler, is also pushed onto the task’s stack. Should the task attempt to exit the procedure (which normally causes unpredictable behaviour), the executive’s fatal error handler will be executed. The user must take this frame into consideration when calculating the sise of a task’s stack(s).

|  |
| --- |
| fatal |
| argp[0] |
| argp[1] |
| argp[2] |
| argp[3] |

The task identified by the *tid* must exist on the local processor, even if the task was created with the GLOBAL flags value set (see *t\_create)*.

RETURN VALUE

If *t\_start* successfully started the task, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tid*.

Task not in dormant state.

Task not created from local node.

NOTES

Not callable from ISR.

May cause a preempt if the task being started has higher priority than the running task, and the preempt mode is in effect.

3.1.4 T\_START

NAME

t\_restart – “Restart a Task”

SYNOPSIS

uint t\_restart ( tid, argp )

uint tid; /\* task id as returned from t\_create ot t\_ident \*/

long argp[4]; /\* pointer to argument list \*/

DESCRIPTION

The task identified by the *tid* is made ready. If the task was blocked, the executive unblocks it. The task’s *superstk, userstk*, and priority are set to their original values established when the task was created using *t\_create*. The task’s start address *saddr* and *mode* are set to their original values established when the task was started using *t\_start*. A task can be restarted from any state.

*Argp* is a pointer to a list of four arguments. These arguments are pushed onto the stack of the task be restarted. This argument list may be different from the original argument list. A fifth argument, the executive’s fatal error handler, is also pushed onto the task’s stack. Should the task attempt to exit the procedure (which normally causes unpredictable behaviour), the executive’s fatal error handler will be executed.

Tasks which anticipate being restarted can use the argument to distinguish between initial startup and a restart.

Due to the capability of this call to unblock a task, this call is useful to delete a task in the sys-tem. Tasks which anticipate being deleted can use the argument to distinguish between initial startup and deletion.

|  |
| --- |
| fatal |
| argp[0] |
| argp[1] |
| argp[2] |
| argp[3] |

The task identified by the *tid* must exist on the local processor, even if the task was created with the GLOBAL flags value set (see *t\_create*).

RETURN VALUE

If *t\_restart* successfully restarted the task, then 0 is returned.

If the call was not successful, an error code is returned.