SchedCtl(), SchedCtl_r()

Control the adaptive partitioning scheduler

Synopsis:

```
#include <sys/sched_aps.h>

int SchedCtl( int cmd,
              void *data,
              int length);

int SchedCtl_r( int cmd,
                void *data,
                int length);
```

Arguments:

cmd The control command that you want to execute; one of:

- SCHED_APS_QUERY_PARMS
- SCHED_APS_SET_PARMS
- SCHED_APS_CREATE_PARTITION
- SCHED_APS_LOOKUP
- SCHED_APS_QUERY_PARTITION
- SCHED_APS_JOIN_PARTITION
- SCHED_APS_MODIFY_PARTITION
- SCHED_APS_PARTITION_STATS
- SCHED_APS_OVERALL_STATS
- SCHED_APS_MARK_CRITICAL
- SCHED_APS_CLEAR_CRITICAL
- SCHED_APS_ATTACH_EVENTS
- SCHED_APS_QUERY_THREAD
- SCHED_APS_ADD_SECURITY
- SCHED_APS_QUERY_PROCESS

data A pointer to the specific data structure for the command.

length The size of the structure that *data* points to.

For details about each command and its data, see the sections below.

Library:

`libc`

Use the **`-l c`** option to **`qcc`** to link against this library. This library is usually included automatically.

Description:

The *SchedCtl()* and *SchedCtl_r()* kernel calls control the adaptive partitioning scheduler.

This scheduler is optional and is present only if you install the Adaptive Partitioning Technology Development Kit and add **`[module=aps]`** to your OS image's buildfile. For more information, see the Adaptive Partitioning TDK *User's Guide*. These functions were added in the QNX Neutrino Core OS 6.3.2.

These functions are identical except in the way they indicate errors. See the Returns section for details.

☞ You must initialize all of the fields—including reserved ones—in the structures you pass as the *data* argument, by calling (for example) *memset()*. You can also use the *APS_INIT_DATA()* macro:

```
APS_INIT_DATA( &data );
```

SCHED_APS_QUERY_PARMS

This command fills in a **`sched_aps_info`** structure that describes the overall parameters of the adaptive partitioning scheduler:

```
typedef struct {
    _Uint64t        cycles_per_ms;
    _Uint64t        windowsize_cycles;
    _Uint64t        windowsize2_cycles;
    _Uint64t        windowsize3_cycles;
    _Uint32t        scheduling_policy_flags;
    _Uint32t        sec_flags;
    _Uint32t        bankruptcy_policy;
    _Uint16t        num_partitions;
    _Uint16t        max_partitions;
    _Uint64t        reserved1;
    _Uint64t        reserved2;
} sched_aps_info;
```

The members include:

cycles_per_ms The number of machine cycles in a millisecond. Use this value to convert the output of the SCHED_APS_QUERY_PARTITION command to the time units of your choice.

☞ The value of *cycles_per_ms*:

- might not equal the value of the *cycles_per_sec* member of the system page divided by 1000
- isn't necessarily in the same units as values returned by *ClockCycles()* on all platforms

windowsize_cycles

The length, in CPU cycles, of the averaging window used for scheduling. By default, this corresponds to 100 ms.

☞ If you change the tick size of the system at runtime, do so before defining the adaptive partitioning scheduler's window size. That's because Neutrino converts the window size from milliseconds to clock ticks for internal use.

windowsize2_cycles

The length, in CPU cycles, of window 2, for reporting only. Typically 10 times the window size.

windowsize3_cycles

The length, in CPU cycles, of window 3, for reporting only. Typically 100 times the window size.

scheduling_policy_flags

The set of SCHED_APS_SCHEDPOL_* flags that describe the scheduling policy. For more information, see "Scheduling policies," below.

sec_flags The set of SCHED_APS_SEC_* flags that describe the security options. For more information, see "Security," below.

bankruptcy_policy

What to do if a partition exhausts its critical budget; a combination of SCHED_APS_BNKR_* flags (see "Handling bankruptcy," below).

num_partitions The number of partitions defined.

max_partitions The largest number of partitions that may be created at any time.

Scheduling policies

These flags set options for the adaptive partitioning scheduling algorithm. To set, pass a pointer to an ORed set of these flags with the SCHED_APS_SET_PARMS call to *SchedCtl()*:

SCHED_APS_SCHEDPOL_FREETIME_BY_RATIO

Free time is when at least one partition isn't running. Its time becomes free to other partitions that may then run over their budgets.

By default, the scheduler hands out free time to the partition with the highest-priority running thread. That guarantees realtime scheduling behavior (i.e. scheduling strictly by priority) to partitions any time they aren't being limited by some other partition's right to its guaranteed minimum budget. But it also means that one partition is allowed to grab all the free time.

If you set SCHED_APS_SCHEDPOL_FREETIME_BY_RATIO, the running partitions share the free time in proportion to the ratios of their budgets. So, one partition can no longer grab all the free time. However, when this flag is set, partitions will see strict priority-scheduling between partitions only when they're consuming less than their CPU budgets.

SCHED_APS_SCHEDPOL_BMP_SAFETY

Strict priority scheduling between partitions, with some combinations of partition budgets, and some combinations of runmasks (i.e. bound multiprocessing) can require the adaptive partitioning scheduler to not meet minimum CPU budgets. When SCHED_APS_SCHEDPOL_BMP_SAFETY is set, the scheduler uses a more restrictive algorithm that guarantees minimum CPU budgets, but gives priority-based scheduling between partitions only when when partitions are consuming less than their budgets.

If this flag is set, SCHED_APS_SCHEDPOL_FREETIME_BY_RATIO is also automatically set.

SCHED_APS_SCHEDPOL_DEFAULT

Neither SCHED_APS_SCHEDPOL_FREETIME_BY_RATIO nor SCHED_APS_SCHEDPOL_BMP_SAFETY. Neutrino sets this at startup.

Scheduling within a partition is always strictly by priority, no matter which of these flags are set.

For more information about adaptive partitioning and BMP, see the Adaptive Partitioning Scheduling Details chapter of the Adaptive Partitioning TDK *User's Guide*.

Handling bankruptcy

Bankruptcy is when critical CPU time billed to a partition exceeds its critical budget. Bankruptcy is always considered to be a design error on the part of the application, but you can configure how the system responds to it.

If the system isn't declaring bankruptcy when you expect it, note that bankruptcy can be declared only if critical time is billed to your partition. Critical time is billed on those timeslices when the following conditions are *all* met:

- The running partition has a critical budget greater than zero.
- The top thread in the partition is marked as running critical, or has received the critical state from receiving a SIG_INTR, a **`sigevent`** marked as critical, or has just received a message from a critical thread.
- The running partition must be out of percentage-CPU budget.
- There be at least one other partition that is competing for CPU time.

Only then if the critical time, billed over the current averaging window, exceeds a partition's critical budget will the system declare the partition bankrupt.

When the system detects that a partition has gone bankrupt, it always:

- causes that partition to be out-of-budget for the remainder of the current scheduling window
- delivers any **`sigevent`** that you've specified as a notification of bankruptcy with the SCHED_APS_ATTACH_EVENTS command. This occurs at most once per calling SCHED_APS_ATTACH_EVENTS.

In addition, you can configure the following responses:

SCHED_APS_BNKR_BASIC

Deliver bankruptcy-notification events and make the partition out-of-budget for the rest of the scheduling window (nominally 100 ms). This is the default.

SCHED_APS_BNKR_CANCEL_BUDGET

Set the offending partition's critical budget to zero, which forces the partition to be scheduled by its percentage CPU budget only. This also means that a second bankruptcy can't occur. This persists until a restart occurs, or you call SCHED_APS_MODIFY_PARTITION to set a new critical budget.

SCHED_APS_BNKR_LOG

Not currently implemented.

SCHED_APS_BNKR_REBOOT

Cause the system to crash with a brief message identifying the offending partition. This is the most severe response, suggested for use while testing a product, to make sure bankruptcies are never ignored. You probably shouldn't use this option in your finished product.

SCHED_APS_BNKR_RECOMMENDED

The combination **`SCHED_APS_BNKR_CANCEL_BUDGET | SCHED_APS_BNKR_LOG`**. We recommend this choice.

To set a choice of bankruptcy-handling options, OR the above SCHED_APS_BNKR_* flags and pass a pointer to it as the *bankruptcy_policyp* field of the **`sched_aps_parms`** structure when you call SCHED_APS_SET_PARMS.

Returns:

EOK	Success.
EACCES	The calling thread doesn't meet the security options set (see SCHED_APS_ADD_SECURITY). Usually this means you must be **`root`**.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

SCHED_APS_SET_PARMS

The command sets the parameters for the overall behavior of the adaptive partitioning scheduler. The *data* argument must be a pointer to a **sched_aps_parms** structure:

```
typedef struct {
    _Int16t windowsize_ms;
    _Int16t     reserved1;
    _Uint32t    *scheduling_policy_flagsp;
    _Uint32t    *bankruptcy_policyp;
    _Int32t     reserved2;
    _Int64t     reserved3;
} sched_aps_parms;
```

The members include:

windowsize_ms

The time over which the scheduler is to average CPU cycles and balance the partitions to their percentage budgets as specified by SCHED_APS_CREATE_PARTITION If you don't want to set the window size, set this member to -1.

scheduling_policy_flagsp

A pointer to an ORed set of SCHED_APS_SCHEDPOL_* flags that specify the scheduling policy. For more information, see "Scheduling policies," above. If you don't want to change the scheduling policy, set this member to NULL.

bankruptcy_policyp

A pointer to an ORing of SCHED_APS_BNKR_* flags, as described under "Handling bankruptcy," above. If you don't want to change these flags, set this member to NULL.

Returns:

EOK — Success.

EACCES — SCHED_APS_SEC_PARTITIONS_LOCKED is set, or SCHED_APS_SEC_ROOT0_OVERALL is set and you aren't running as **`root`** in the System partition.

For more information, see "Security," below.

EDOM — A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL — The size of the parameter block doesn't match the size of the expected structure.

ENOSYS — The adaptive partitioning scheduler isn't installed.

SCHED_APS_CREATE_PARTITION

This command creates a new partition which is considered to be a child of the partition that's calling *SchedCtl()*. The system automatically creates a partition called **`System`** (the value of APS_SYSTEM_PARTITION_NAME) with an ID of 0.

The *data* argument for this command must be a pointer to a **`sched_aps_create_parms`** structure:

```
typedef struct {
    /* input parms */
    char        *name;
    _Uint16t    budget_percent;
    _Int16t     critical_budget_ms;
    _Uint32t    reserved1;
    _Uint64t    reserved2;
    /* output parms */
    _Int16t     id;
} sched_aps_create_parms;
```

The input members include:

name — The name of the new partition. If *name* is NULL or points to an empty string, *SchedCtl()* assigns a name, in the form **`Pa`**,

Pb, **Pc**, and so on. The name must be no longer than APS_PARTITION_NAME_LENGTH, not including the trailing null character, and can't include any slashes (**/**).

budget_percent

The percentage CPU budget for the new partition. Budgets given to the new partition are subtracted from the parent partition.

☞ Before creating zero-budget partitions, read the cautions in "Setting budgets for resource managers" in the System Considerations chapter of the Adaptive Partitioning TDK *User's Guide*.

---

critical_budget_ms

The critical budget, in milliseconds, for the partition, or -1 or 0 if you don't want the partition to have a critical budget. Critical budgets don't affect the parent, but are automatically limited to be no bigger than the window size.

The output members include:

id The created partition's ID number, in the range 0 to the maximum number of partitions – 1 (see the *max_partitions* member of the data from a call to SCHED_APS_QUERY_PARMS. The System partition's ID number is APS_SYSTEM_PARTITION_ID.

Returns:

EOK Success.

EACCES SCHED_APS_SEC_PARTITIONS_LOCKED is set, or any of these security conditions are set and not satisfied:

- SCHED_APS_SEC_ROOT_MAKES_PARTITIONS
- SCHED_APS_SEC_SYS_MAKES_PARTITIONS
- SCHED_APS_SEC_NONZERO_BUDGETS

- SCHED_APS_SEC_ROOT_MAKES_CRITICAL
- SCHED_APS_SEC_SYS_MAKES_CRITICAL

For more information, see “Security,” below.

EDOM
: A reserved field isn’t zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EDQUOT
: The parent partition doesn’t have enough budget.

EEXIST
: Another partition is already using the given name.

EINVAL
: The size of the parameter block doesn’t match the size of the expected structure, the partition name is badly formed, or the budget is out of range.

ENAMETOOLONG
: The partition name is longer than APS_PARTITION_NAME_LENGTH characters.

ENOSPC
: The maximum number of partitions already exist.

ENOSYS
: The adaptive partitioning scheduler isn’t installed.

SCHED_APS_QUERY_PARTITION

This command gets information about a given partition. The *data* argument for this command must be a pointer to a **`sched_aps_partition_info`** structure:

```
typedef struct {
    /* out parms */
    _Uint64t    budget_cycles;
    _Uint64t    critical_budget_cycles;
    char        name[APS_PARTITION_NAME_LENGTH+1];
    _Int16t     parent_id;
    _Uint16t    budget_percent;
    _Int32t     notify_pid;
    _Int32t     notify_tid;
    _Uint32t    pinfo_flags;
    _Int32t     pid_at_last_bankruptcy;
    _Int32t     tid_at_last_bankruptcy;
    _Int64t     reserved1;
    _Int64t     reserved2;
    /* input parm */
    _Int16t     id;
} sched_aps_partition_info;
```

The input members include:

id The number of the partition you want to query.

The output members include:

budget_cycles The budget, in cycles. To convert this value to something useful, convert it with the *cycles_per_ms* value from a SCHED_APS_QUERY_PARMS command.

critical_budget_cycles
The critical budget, in cycles.

name The name of the partition.

parent_id The number of the partition that's the parent of the partition being queried. The System partition's ID number is APS_SYSTEM_PARTITION_ID.

budget_percent The partition's budget, expressed as a percentage.

notify_pid, *notify_tid*
The process and thread IDs of the thread to be given overload and bankruptcy notifications, or -1 if not set.

pinfo_flags A set of flag that give extra information about the partition:

- SCHED_APS_PINFO_BANKRUPTCY_NOTIFY_ARMED — see SCHED_APS_ATTACH_EVENTS
- SCHED_APS_PINFO_OVERLOAD_NOTIFY_ARMED — see SCHED_APS_ATTACH_EVENTS

pid_at_last_bankruptcy, *tid_at_last_bankruptcy*

The process and thread IDs at the time of the last bankruptcy, or -1 if there wasn't a previous bankruptcy.

Returns:

EOK	Success.
EDOM	A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.
EINVAL	The size of the parameter block doesn't match the size of the expected structure.
ENOSYS	The adaptive partitioning scheduler isn't installed.

SCHED_APS_LOOKUP

This command finds the partition ID for a given partition name.

The *data* argument for this command must be a **sched_aps_lookup_parms** structure:

```
typedef struct {
        /* input parms */
        char    *name;
        _Int16t reserved1;
        /* output parms */
        _Int16t     id;
} sched_aps_lookup_parms;
```

The input members include:

name The name of the partition

The output members include:

id The ID number of the partition, if found.

Returns:

EOK	Success.
EDOM	A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.
EINVAL	The name wasn't found.

SCHED_APS_JOIN_PARTITION

This command makes the thread specified by the given process and thread IDs becomes a member of the specified partition. This partition also becomes the thread's new home partition, i.e. where it returns after partition inheritance.

The *data* argument for this command must be a pointer to a **`sched_aps_join_parms`** structure:

```
typedef struct {
        _Int16t     id;
        _Int16t     reserved1;
        _Int32t     pid;
        _Int32t     tid;
        _Int32t     reserved2;
} sched_aps_join_parms;
```

The members include:

id The ID number of the partition that the thread is to join.

pid, *tid* The process and thread IDs of the thread that you want to join the specified partition:

- If both *pid* and *tid* are zero, the calling thread joins the specified partition.
- If *tid* is -1, the process with ID *pid* joins the partition. This doesn't change the partitions that the process's threads are in; it just sets the partition that the threads run in when they're handling a pulse.

Returns:

EOK Success.

EACCES The following security options are set but not satisfied:

- SCHED_APS_SEC_ROOT_JOINS
- SCHED_APS_SEC_SYS_JOINS
- SCHED_APS_SEC_PARENT_JOINS
- SCHED_APS_SEC_JOIN_SELF_ONLY

For more information, see "Security," below.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The *pid* and *tid* are invalid.

SCHED_APS_MODIFY_PARTITION

This command changes the parameters of an existing partition. If the new budget's percent value is different from the current, the difference is either taken from, or returned to, the parent partition's budget. The critical time parameter affects only the chosen partition, not its parent. To change just one of new budget or new critical time, set the other to -1.

☞

- You can’t use this command to modify the System partition’s budget. To increase the size of the System partition, reduce the budget of one of its child partitions.
- Reducing the size of a partition may cause it not to run for the time of an averaging window, as you may have caused it to become temporarily over-budget. However, reducing the critical time doesn’t trigger the declaration of bankruptcy.

The *data* argument for this command must be a pointer to a **`sched_aps_modify_parms`** structure:

```
typedef struct {
        _Int16t     id;
        _Int16t     new_budget_percent;
        _Int16t     new_critical_budget_ms;
        _Int16t     reserved1;
        _Int64t     reserved2;
        _Int64t     reserved3;
} sched_aps_modify_parms;
```

The members include:

id The ID number of the partition.

new_budget_percent

The new budget for the partition, expressed as a percentage, or -1 if you don’t want to change it.

new_critical_budget_ms

The new critical budget, in milliseconds, for the partition, or -1 if you don’t want to change it. If the critical budget is greater than the window size, it’s considered to be infinite.

Returns:

EOK
Success.

EACCES
SCHED_APS_SEC_PARTITIONS_LOCKED is set, or the following security options are set and not satisfied:

- SCHED_APS_SEC_PARENT_MODIFIES
- SCHED_APS_SEC_ROOT_MAKES_PARTITIONS
- SCHED_APS_SEC_SYS_MAKES_PARTITIONS
- SCHED_APS_SEC_NONZERO_BUDGETS
- SCHED_APS_SEC_ROOT_MAKES_CRITICAL
- SCHED_APS_SEC_SYS_MAKES_CRITICAL

For more information, see “Security,” below.

EDOM
A reserved field isn’t zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL
The size of the parameter block doesn’t match the size of the expected structure, or the partition ID is invalid.

ENOSYS
The adaptive partitioning scheduler isn’t installed.

SCHED_APS_PARTITION_STATS

This command returns instantaneous values of the CPU time-accounting variables for a set of partitions. It can fill in data for more than one partition. If the *length* argument to *SchedCtl()* indicates that you’ve passed the function an array of **`sched_aps_partition_stats`** structures, *SchedCtl()* fills each element with statistics for a different partition, starting with the partition specified by the *id* field.

☞ To get an accurate picture for the the whole machine it's important to read data for all partitions in one call, since sequential calls to SCHED_APS_PARTITION_STATS may come from separate averaging windows.

To determine the number of partitions, use the SCHED_APS_OVERALL_STATS command.

The command overwrites the *id* field with the partition number for which data is being returned. It stores `-1` into the *id* field of unused elements.

To convert times in cycles into milliseconds, divide them by the *cycles_per_ms* obtained with an SCHED_APS_QUERY_PARMS command.

The *data* argument for this command must be a pointer to a **`sched_aps_partition_stats`** structure, or an array of these structures:

```
typedef struct {
    /* out parms */
    _Uint64t        run_time_cycles;
    _Uint64t        critical_time_cycles;
    _Uint64t        run_time_cycles_w2;
    _Uint64t        critical_time_cycles_w2;
    _Uint64t        run_time_cycles_w3;
    _Uint64t        critical_time_cycles_w3;
    _Uint32t        stats_flags;
    _Uint32t        reserved1;
    _Uint64t        reserved2;
    _Uint64t        reserved3;
    /* in parm */
    _Int16t     id;
} sched_aps_partition_stats;
```

The members include:

run_time_cycles

The CPU execution time during the last scheduling window.

critical_time_cycles

The time spent running critical threads during the last scheduling window.

run_time_cycles_w2

The CPU time spent during the last *windowsize2_cycles*. Window 2 is typically 10 times the length of the averaging window.

critical_time_cycles_w2

The time spent running critical threads during the last *windowsize2_cycles*.

run_time_cycles_w3

The CPU time spent during the last *windowsize3_cycles*. Window 3 is typically 100 times the length of the averaging window.

critical_time_cycles_w3

The time spent running critical threads during the last *windowsize3_cycles*.

stats_flags A set of the following flags:

- SCHED_APS_PSTATS_IS_BANKRUPT_NOW — the critical time used is greater than the critical budget at the time you used the SCHED_APS_PARTITION_STATS command.
- SCHED_APS_PSTATS_WAS_BANKRUPT — the partition was declared bankrupt sometime since the last restart.

id This is both an input and output field. As input, it's the ID number of the first partition you want data for. If you've passed an array of **`sched_aps_partition_stats`** structures, the command fills in the ID number for each partition that it fills in statistics for.

Returns:

EOK
: Success.

EDOM
: A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL
: The size of the parameter block isn't a multiple of **size(sched_aps_partition_stats)**.

ENOSYS
: The adaptive partitioning scheduler isn't installed.

SCHED_APS_OVERALL_STATS

This command returns instantaneous values of overall CPU-usage variables and other dynamic scheduler states. The *data* argument for this command must be a pointer to a **sched_aps_overall_stats** structure:

```
typedef struct {
        _Uint64t    idle_cycles;
        _Uint64t    idle_cycles_w2;
        _Uint64t    idle_cycles_w3;
        _Int16t     id_at_last_bankruptcy;
        _Uint16t    reserved1;
        _Int32t     pid_at_last_bankruptcy;
        _Int32t     tid_at_last_bankruptcy;
        _Uint32t    reserved2;
        _Uint32t    reserved3;
        _Uint64t    reserved4;
} sched_aps_overall_stats;
```

The members include:

idle_cycles
: The time, in cycles, during the last scheduling window where nothing (other than the idle thread) ran. To convert this to the percent idle time, calculate:

  (100 × *idle_cycles*) / *windowsize_cycles*

idle_cycles_w2
: The time spent running idle during the last *windowsize2_cycles*. Window 2 is typically 10 times the length of the averaging window.

idle_cycles_w3
: The time spent running idle during last *windowsize3_cycles*. Window 3 is typically 100 times the length of the averaging window.

id_at_last_bankruptcy
: The ID of last bankrupt partition, or -1 if no bankruptcy has occurred.

pid_at_last_bankruptcy, *tid_at_last_bankruptcy*
: The process and thread IDs at last the bankruptcy, or -1 if no bankruptcy has occurred.

Returns:

EOK
: Success.

EDOM
: A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL
: The size of the parameter block doesn't match the size of the expected structure.

ENOSYS
: The adaptive partitioning scheduler isn't installed.

SCHED_APS_MARK_CRITICAL

This command sets one thread in your process to run as a critical thread whenever it runs. Use a thread ID of zero to set the calling thread to be critical.

☞ In general, it's more useful to send a critical **`sigevent`** structure to a thread to make it run as a critical thread.

The *data* argument for this command must be a pointer to a **`sched_aps_mark_crit_parms`** structure:

```
typedef struct {
        _Int32t     pid;
        _Int32t     tid;
        _Int32t     reserved;
} sched_aps_mark_crit_parms;
```

The members include:

pid The process ID, or 0 for the calling process.

tid The thread ID, or 0 for the calling thread.

☞ You can also set up **`sigevent`** structures that make their receiving threads run as critical.

Returns:

EOK Success.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The specified thread wasn't found.

SCHED_APS_CLEAR_CRITICAL

This command clears the "always run as critical" state set by the SCHED_APS_CLEAR_CRITICAL command. Then the thread will run as critical only when it inherits that state from another thread (on receipt of a message).

The *data* argument for this command must be a pointer to a **`sched_aps_clear_crit_parms`** structure:

```
typedef struct {
        _Int32t     pid;
        _Int32t     tid;
        _Int32t     reserved;
} sched_aps_clear_crit_parms;
```

The members include:

pid The process ID, or 0 for the calling process.

tid The thread ID, or 0 for the calling thread.

Returns:

EOK Success.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The specified thread wasn't found.

SCHED_APS_QUERY_THREAD

This command determines the partition for the given thread and indicates whether or not the thread in your process is marked to run as critical. Use a thread ID of zero to indicate the calling thread.

The *data* argument for this command must be a pointer to a **sched_aps_query_thread_parms** structure:

```
typedef struct {
        _Int32t     pid;
        _Int32t     tid;
        /* out parms: */
        _Int16t     id;
        _Int16t     inherited_id;
        _Uint32t    crit_state_flags;
        _Int32t     reserved1;
        _Int32t     reserved2;
} sched_aps_query_thread_parms;
```

The input members include:

pid The ID of process that the thread belongs to, or 0 to indicate the calling process.

tid The thread ID, or 0 for the calling thread.

The output members include:

id The ID number of the partition that the thread originally joined.

inherited_id The ID number of the partition that the thread currently belongs to. This might not be the same as the *id* member, because the thread might have inherited the partition from a calling process.

crit_state_flags A combination of the following flags:

- APS_QCRIT_PERM_CRITICAL — the thread always runs as critical.
- APS_QCRIT_RUNNING_CRITICAL — the thread is currently running as critical.
- APS_QCRIT_BILL_AS_CRITICAL — the thread's execution time is being billed to the partition's critical budget.

If APS_QCRIT_PERM_CRITICAL isn't set, and APS_QCRIT_RUNNING_CRITICAL is set, it means the thread has temporarily inherited the critical state. If APS_QCRIT_RUNNING_CRITICAL is set, and APS_QCRIT_BILL_AS_CRITICAL isn't set, it means that the thread is running as critical, but isn't depleting its partition's critical-time budget (i.e. it's running for free).

Returns:

EOK Success.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The specified thread wasn't found.

SCHED_APS_ATTACH_EVENTS

This command defines **`sigevent`** structures that the scheduler will return to the calling thread when the scheduler detects that a given partition has become bankrupt, or the whole system has become overloaded.

☞ Overload notification isn't implemented in this release.

Calling SCHED_APS_ATTACH_EVENTS arms the notification once. After you receive the notification, you must call SCHED_APS_ATTACH_EVENTS again to receive a subsequent notification. This is to ensure that the system doesn't send you notifications faster than you can handle them. The *pinfo_flags* field of the **`sched_aps_partition_stats`** structure (see the SCHED_APS_PARTITION_STATS command) indicates if these events are armed.

☞ You can register only one pair of **`sigevent`** structures (bankruptcy and overload) per partition, and the notifications must go to the same thread. The thread notified is the calling thread. Attaching events a second time overwrites the first. Passing NULL pointers means "no changes in notification." To turn off notification, use *SIGEV_NONE_INIT()* to set the appropriate **`sigevent`** to SIGEV_NONE.

If you want to configure additional actions for the system to perform on bankruptcy, see "Handling bankruptcy," below.

The *data* argument for this command must be a pointer to a **`sched_aps_events_parm`** structure:

```
typedef struct {
        const struct sigevent   *bankruptcy_notification;
        const struct sigevent   *overload_notification;
        /* each partition gets a different set of sigevents */
        _Int16t                 id;
        _Int16t                 reserved1;
        _Int32t                 reserved2;
        _Int64t                 reserved3;
} sched_aps_events_parm;
```

The members include:

bankruptcy_notification

A pointer to the **`sigevent`** to send to the calling thread if the partition becomes bankrupt, or NULL if you don't want to change the notification.

overload_notification

Not implemented.

id The ID of the partition that you want to attach events to, or `-1` for the partition of the calling thread. The command updates this member to indicate the partition that it attached the events to.

Returns:

EOK Success.

EACCES You don't have the right to modify the partition, i.e the following security modes are set and not satisfied:

- SCHED_APS_SEC_PARENT_MODIFIES
- SCHED_APS_SEC_ROOT_MAKES_PARTITIONS
- SCHED_APS_SEC_SYS_MAKES_PARTITIONS

For more information, see "Security," below.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The specified thread wasn't found.

SCHED_APS_ADD_SECURITY

This command sets security options. A bit that's set turns the corresponding security option on. Successive calls add to the existing set of security options. Security options can only be cleared by a restart.

☞ You must be **`root`** running in the System partition to use this command, even if all security options are off.

The *data* argument for this command must be a pointer to a **`sched_aps_security_parms`** structure:

```
typedef struct {
        _Uint32t        sec_flags;
        _Uint32t        reserved1;
        _Uint32t        reserved2;
} sched_aps_security_parms;
```

The members include:

sec_flags A set of SCHED_APS_SEC_* flags (see below), as both input and output parameters. Set this member to 0 if you want to get the current security flags.

Security

The adaptive partitioning scheduler lets you dynamically create and modify the partitions in your system.

☞ We recommend that you set up your partition environment at boot time, and then lock all parameters:

- in a program, by using the SCHED_APS_SEC_LOCK_PARTITIONS flag
- from the command line, by using the **`aps modify`** command

However you might need to modify a partition at runtime. In this case, you can use the security options described below.

When Neutrino starts, it sets the security option to APS_SCHED_SEC_OFF. We recommend that you immediately set it to SCHED_APS_SEC_RECOMMENDED. In code, do this:

```
sched_aps_security_parms p;

APS_INIT_DATA( &p );
p.sec_flags = SCHED_APS_SEC_RECOMMENDED;
SchedCtl( SCHED_APS_ADD_SECURITY,&p, sizeof(p) );
```

These are the security options:

SCHED_APS_SEC_RECOMMENDED

> Only **`root`** from the System partition may create partitions or change parameters. This arranges a 2-level hierarchy of partitions: the System partition and its children. Only **`root`**, running in the System partition, may join its own thread to partitions. The percentage budgets must not be zero.

SCHED_APS_SEC_FLEXIBLE

> Only **`root`** in the System partition can change scheduling parameters or change critical budgets. But **`root`** running in any partition can create subpartitions, join threads into its own subpartitions and modify subpartitions. This lets applications create their own local subpartitions out of their own budgets. The percentage budgets must not be zero.

SCHED_APS_SEC_BASIC

Only **root** in the System partition may change overall scheduling parameters and set critical budgets.

Unless you're testing the partitioning and want to change all parameters without needing to restart, you should set at least SCHED_APS_SEC_BASIC.

In general, SCHED_APS_SEC_RECOMMENDED is more secure than SCHED_APS_SEC_FLEXIBLE, which is more secure than SCHED_APS_SEC_BASIC. All three allow partitions to be created and modified. After setting up partitions, use SCHED_APS_SEC_LOCK_PARTITIONS to prevent further unauthorized changes. For example:

```
sched_aps_security_parms p;

APS_INIT_DATA( &p );
p.sec_flags = SCHED_APS_SEC_LOCK_PARTITIONS;
SchedCtl( SCHED_APS_ADD_SECURITY, &p, sizeof(p));
```

SCHED_APS_SEC_RECOMMENDED, SCHED_APS_SEC_FLEXIBLE, and SCHED_APS_SEC_BASIC are composed of the flags defined below (but it's usually more convenient for you to use the compound options):

SCHED_APS_SEC_ROOT0_OVERALL

You must be **root** running in the System partition in order to change the overall scheduling parameters, such as the averaging window size.

SCHED_APS_SEC_ROOT_MAKES_PARTITIONS

You must be **root** in order to create or modify partitions. Applies to the SCHED_APS_CREATE_PARTITION, SCHED_APS_MODIFY_PARTITION, and SCHED_APS_ATTACH_EVENTS commands.

SCHED_APS_SEC_SYS_MAKES_PARTITIONS

You must be running in the System partition in order to create or modify partitions. This applies to same commands as

SCHED_APS_SEC_ROOT_MAKES_PARTITIONS. Attaching events, with SCHED_APS_ATTACH_EVENTS, is considered to be modifying the partition.

SCHED_APS_SEC_PARENT_MODIFIES

Allows partitions to be modified (SCHED_APS_MODIFY_PARTITION), but you must be running in the parent partition of the partition being modified. "Modify" means to change a partition's percentage or critical budget or attach events with the SCHED_APS_ATTACH_EVENTS command.

SCHED_APS_SEC_NONZERO_BUDGETS

A partition may not be created with, or modified to have, a zero budget. Unless you know that all your partitions need to run only in response to client requests, i.e. receipt of messages, you should set this option.

SCHED_APS_SEC_ROOT_MAKES_CRITICAL

You have to be **`root`** in order to create a nonzero critical budget or change an existing critical budget.

SCHED_APS_SEC_SYS_MAKES_CRITICAL

You must be running in the System partition to create a nonzero critical budget or change an existing critical budget.

SCHED_APS_SEC_ROOT_JOINS

You must be **`root`** in order to join a thread to a partition.

SCHED_APS_SEC_SYS_JOINS

You must be running in the System partition in order to join a thread.

SCHED_APS_SEC_PARENT_JOINS

You must be running in the parent partition of the partition you wish to join to.

SCHED_APS_SEC_JOIN_SELF_ONLY

The caller of the SCHED_APS_JOIN_PARTITION command must specify 0 for the *pid* and *tid*. In other words, a process may join only itself to a partition.

SCHED_APS_SEC_PARTITIONS_LOCKED

Prevent further changes to any partition's budget, or overall scheduling parameters, such as the window size. Set this after you've set up your partitions. Once you've locked the partitions, you can still use the SCHED_APS_JOIN_PARTITION and SCHED_APS_ATTACH_EVENTS commands.

Returns:

EOK Success.

EACCES The calling thread doesn't meet the security options set (see SCHED_APS_ADD_SECURITY). Usually this means you must be **root**.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

SCHED_APS_QUERY_PROCESS

This command returns the partition of the given process. The partition of a process is billed while one of the process's threads handles a pulse. The individual threads in a process may all be in different partitions from the process.

The *data* argument for this command must be a pointer to a **sched_aps_query_process_parms** structure:

```
typedef struct {
        _Int32t     pid;
        /* out parms: */
        _Int16t     id;     /* partition of process */
        _Int16t     reserved1;
        _Int32t     reserved2;
        _Int32t     reserved3;
        _Int32t     reserved4;
} sched_aps_query_process_parms;
```

The members include:

pid The process ID, or 0 for the calling process.

id The ID of the process's partition.

Returns:

EOK Success.

EDOM A reserved field isn't zero. You might not have used *APS_INIT_DATA()* to initialize the data parameter.

EINVAL The size of the parameter block doesn't match the size of the expected structure.

ENOSYS The adaptive partitioning scheduler isn't installed.

ESRCH The process wasn't found.

Blocking states

This call doesn't block.

Returns:

The only difference between these functions is the way they indicate errors:

SchedCtl() EOK if the call succeeds. If an error occurs, it returns -1 and sets *errno*.

SchedGet_r() EOK if the call succeeds. This function **doesn't** set *errno*. If an error occurs, the function returns the negative of a error value.

For a list of error values, see the description of each command.

Examples:

```
sched_aps_partition_info part_info;

// You need to initialize the parameter block.
APS_INIT_DATA(&part_info);

// Set the input members of the parameter block.
part_info.id = 2;

// Invoke SchedCtl to query the partition.
ret = SchedCtl( SCHED_APS_QUERY_PARTITION, &part_info,
                sizeof(part_info) );
if (EOK!=ret) some_kind_of_error_handler();

// Use output field
printf( "The budget is %d per cent.\n",
        part_info.budget_percent);
```

Classification:

QNX Neutrino

Safety	
Cancellation point	No
Interrupt handler	No
Signal handler	Yes
Thread	Yes

See also:

SchedGet(), *SchedInfo()*, *SchedSet()*, *SchedYield()*, **`sigevent`**

`aps` in the *Utilities Reference*

Adaptive Partitioning TDK *User's Guide*