OCEOSmp

OCEOSmp

RTOS for RISC-V, ARM, SPARC

(ESA contract 4000141330/23/NL/GLC/adu)

Application Note: When things go wrong

Prof. M. Ryan

OCEOSmp – When Things Go Wrong

Things can go wrong even when software is bug free.

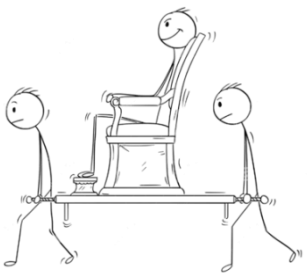
Misunderstandings may have led to an inadequate design, real world circumstances may conspire to cause performance problems, radiation or other factors may cause hardware failures, and in general things can always go wrong.

Murphy's Law then applies, and while the system is in use it is important that system behaviour can be monitored, deviations from expectations noticed, checks made, corrections done, and recovery steps taken if needed.

OCEOSmp has been designed with Murphy's Law in mind and facilitates such actions in a number of ways.

1. Servant not Master
2. White Box not Black Box
3. Directive warning and error status codes
4. Automatic integrity checks
5. System Log
6. System state variable
7. System Policing and Problem Anticipation
8. Problem Detection and Handling
9. Precautions and Correction

Servant not Master:



Many operating systems begin execution directly after the boot sequence and control all subsequent execution of the application software (ASW). Should something go wrong it can be difficult to assess what has occurred and find appropriate corrective action, particularly if the operating system itself has been involved.

With OCEOSmp the ASW begins execution before the operating system and only starts OCEOSmp at some point after entering *main()*.

Before doing so the ASW should check the circumstances that lead to *main()* being entered. If due to a system restart then the data accumulated by OCEOSmp in the previous run is still available in its fixed, dynamic, and log data areas for analysis or download to find the cause (assuming power has been maintained or non-volatile memory is used for those areas).

The ASW can then make appropriate modifications, and continue as usual to set up EDAC, set up memory protection, initialise and check CPU cores and peripherals, and once all seems correct proceed to start or restart OCEOSmp.

Once scheduling begins OCEOSmp returns to *main()* only if a fatal error is detected or if it is instructed to do so by an ASW task. Should this occur all the data accumulated is available for analysis or download by ASW *main()* code, which can restart scheduling by calling *oceos_start()*.

Most problems can be dealt with by tasks set up to do so under the control of OCEOSmp, but sufficiently serious ones may cause a system reset (e.g. a watchdog timer running out) or cause

OCEOSmp to exit (e.g. corruption of critical data), in both cases with the previous state available for analysis if power is maintained or if non-volatile memory is used.

In *main()* the ASW takes five principal steps in starting OCEOSmp:

(i) check whether the entry to *main()* is due to a restart (e.g. by checking if the OCEOSmp data area sentinels are already in place) and if so use the data areas to determine the cause and make whatever modifications to the system are appropriate.

(ii) use *oceos_init()* with the ASW configuration to set up the log area and prepare the fixed data area to hold various addresses, the number of CPU cores, tasks, mutexes, etc.

(iii) use various OCEOSmp create functions to create tasks, mutexs etc.

(iv) use *oceos_init_finish()* to check that the declared numbers of tasks etc. have been created, finish setting up the fixed data area, and set its checksum.

At this point the fixed data area with fixed information about addresses, cores, tasks, mutexes, data queues etc. has been set up, and need not be set up again if not corrupted. The checksum allows the area be checked at any time to ensure it has not been changed. Memory protection if available can make the fixed area read-only and make the stack addresses for each core restricted to being written only by that core.

Scheduling can begin.

(v) use *oceos_start()* to initialise the dynamic data area and begins scheduling. This function usually does not return to *main()* but may do so if a major problem is detected.

As scheduling progresses the ASW can use the system log to record events and monitor performance. The log is also used by OCEOSmp to record problems, and OCEOSmp may also make a system state entry and trigger an ASW problem handling function.

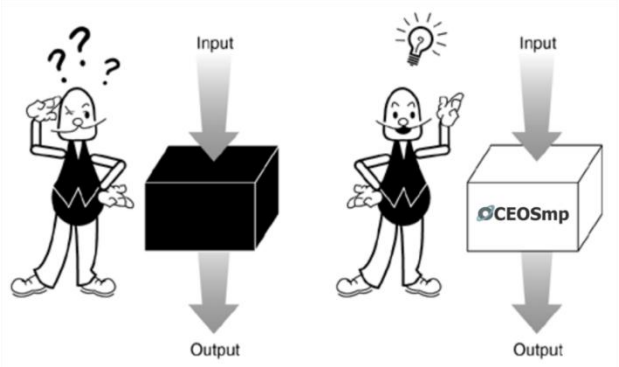
If a problem is sufficiently serious OCEOSmp may exit automatically or may be requested to do so by an ASW task, execution then returns to *main()* at the point after the call to *oceos_start()* with all performance, log, stack and other data available for analysis or download.

The ASW *main()* code can then determine the reason for the unexpected return, take appropriate corrective action, and resume scheduling by calling *oceos_start()* or modify various settings and start over with *oceos_init()*.

The ASW uses OCEOSmp and remains the master, rather than the reverse.

White Box not Black Box:

The data used by OCEOSmp is stored in statically allocated memory in three data areas and in an overall stack area. These usually are specified by the ASW as arrays of 32-bit words and can be examined by the ASW at any time.



The data includes actual and expected task maximum and minimum execution times, activities on CPU cores, and other records of what has been happening. These records can be readily examined by the ASW while scheduling is under way and after OCEOSmp exits should this occur.

OCEOSmp configures each data area to begin with a 32-bit sentinel followed by its size measured in 32-bit words, then the corresponding data, and finally a 32-bit sentinel. In the case of the fixed data area a 32-bit XOR checksum is located before the end sentinel.

The size of each data area and the layout of the data within it depends on the application configuration, but are clearly specified with the location of each data item identified.

Whenever an OCEOSmp directive is used a quick check is made that sentinels have not been breached. The ASW itself can check the fixed area checksum at any time (this is not done automatically for reasons of efficiency).

The Log Area is initialized by *oceos_init()* and contains the system log and system state variable and if used the optional context switch log. Directives are provided to access and update these. The size of the area depends on the number of entries specified for the two logs in the system configuration. Log contents are preserved across system reset and across power failures if stored in non-volatile memory.

The Fixed Data Area is completed by *oceos_init_finish()* and contains all the constants of the system, such as the addresses of data areas and stacks, the identity of the CPU core that OCEOSmp is to use on startup, the number of cores, tasks etc., the priorities and other attributes of tasks etc.

The Dynamic Data area is created and initialized by *oceos_start()* and contains the information collected as oceosmp schedules tasks and as mutexes, counting semaphores, etc. are used. This information is available for inspection by the ASW if OCEOSmp exits, and is only overwritten if the ASW again calls *oceos_start()*.

In OCEOSmp each CPU core has a system stack, a stack for each thread is not required. The stack size per CPU core and the location of the overall top of stack is specified in the application configuration and the overall stack space is statically allocated.

Before scheduling begins OCEOSmp initialises each core's stack with a filler. This is overwritten as the stack is used and by checking how much memory at the top and bottom of each stack still contains the filler it is possible to determine at any point how close the system has come to a stack overrun.

The white box approach of making all records and the system stacks readily accessible while scheduling is taking place and should OCEOSmp exit facilitates checking that the system is performing as expected and determining the causes of any problems that may occur.

Directive warning and error status codes

The ASW uses OCEOSmp by calling its directives, and is responsible for checking the returned directive status code. (OCEOSmp is a statically linked library, only those components related to the directives used by the ASW are linked into the final executable.)

The system time directives *oceos_time_sys_get64()* and *oceos_time_sys_get32()* directly return the requested system time as 64-bit or 32-bit unsigned integers.

Other directives return a 32-bit signed integer status code.

This should always be checked when a directive is used.

The status code indicates ERROR, SUCCESS, or WARNING.

- A negative value indicates that the directive has not executed and gives the reasons
- A zero value indicates successful execution as expected
- A non-zero positive value indicates that the directive executed but with warnings

Certain bits in the status code have the same meaning for all directives.

Other bits have meanings that depend on the directive in use.

These meanings are given in the header files associated with each directive as constants that can be compared with the status code to determine the types of errors or warnings involved.

Errors typically involve an incorrect parameter to a directive, or an attempt to use a resource that is not available such as writing to a data queue that is already full.

Warnings typically indicate that the directive has succeeded but that a danger has been created as when a task acquires multiple mutual exclusion semaphores out of order, or that a resource is now fully in use as when a write makes a data queue full.

The task that used the directive and checked the returned status has full access to the data areas and stack areas in deciding the action to take, which may involve making a system log entry, starting a problem handling task, or in the worst case exiting from OCEOSmp and returning to *main()*.

In many situations the returned status code is the only indicator that is needed of directive success or failure. The ASW can then make a system log entry (automatically timestamped by OCEOSmp) for recording or performance analysis purposes if desired.

In some situations OCEOSmp will make a system log entry automatically and may also update the system state variable, resulting in an ASW defined problem handling function being called if the corresponding flag has been set by the ASW in the system action flags.

OCEOSmp may also automatically make a system log entry or take other actions when problems arise later, as when a task completion deadline is missed or a mutex is not returned.

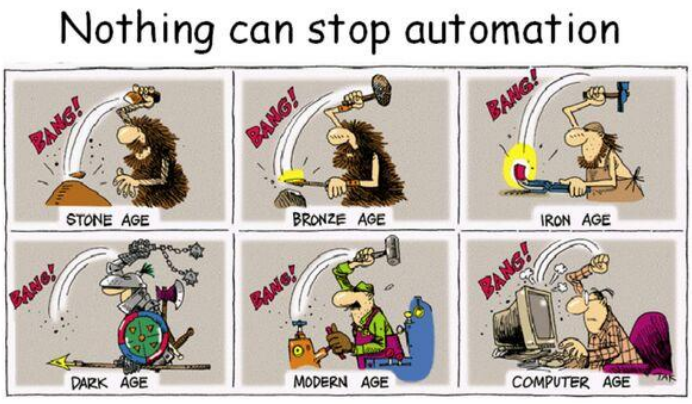
Irrespective of whatever other precautions may be taken, the first line of defence in using a directive is to check the returned status code.

Automatic integrity tests

These are done automatically by OCEOSmp when a directive is used or an event occurs.

They include checking that OCEOSmp data area sentinels are intact, and in the event of corruption may result in an exit from OCEOSmp and return to *main()*.

Directive parameters and the current system phase also are automatically checked when a directive is used, together with the availability of any required resources.



Problems detected immediately when a directive is used result in an appropriate status code being returned to the ASW task that called the directive.

Some problems, such as a task missing a deadline, do not occur immediately when a directive is used but only as scheduling proceeds. Such problems are checked for automatically by OCEOSmp.

Checks include task completion times vs. deadlines, task start requests intervals vs. expected maxima and minima, mutual exclusion semaphore not being released before task completion, time taken to obtain spinlocks longer than expected (an indicator of system loading), and other problems.

If a check fails in most cases a system log entry is made and the system state variable may be updated, possibly resulting in an ASW defined problem handler being called. In extreme cases OCEOSmp may exit and return to *main()*.

Overall the automatic integrity tests carried out by OCEOSmp itself include:

1. System meta pointer intact
2. Data area sentinel/s intact
3. Task deadline not missed
4. Task start request gap too short
5. Task start request gap too long
6. …

It should be noted that these checks detect that a problem has happened. To anticipate problems and perhaps avoid them further checks can and should be done by the ASW as part of system policing.

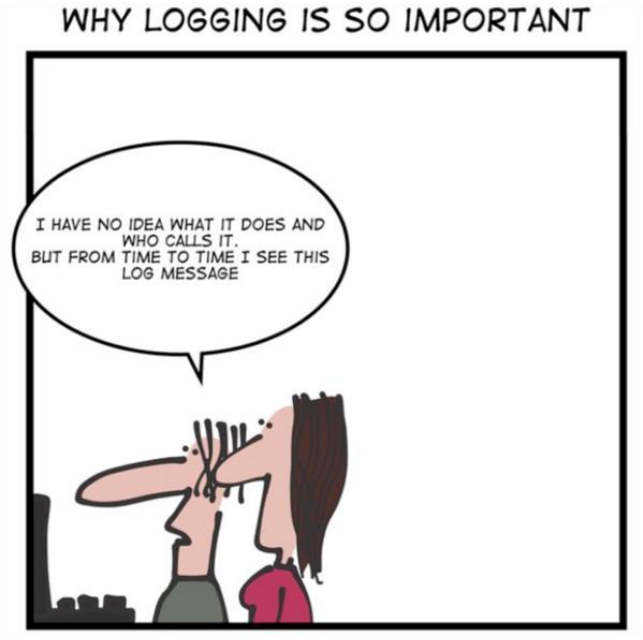
System Log

Purpose:

1. Provide application software with the ability to make time stamped records of events for use in application performance monitoring and in application debugging
2. Provide OCEOSmp with the ability to record anomalies should these be detected.

Structure:

3. A circular buffer with a fixed size that is set in the application configuration
4. Size from 16 to 1024 entries, default 64 if size not specified
5. If the application defines *oceosmp_on_full_log(void *)* this is called automatically when the buffer becomes ¾ full, allowing log entries that might otherwise be overwritten be archived. The automatic call will reoccur only after the buffer becomes ¾ empty.
6. Two indices determine the next log entry position to be read from or



written to. On system reset these are left unchanged if valid so as to postpone recent entries being overwritten.

7. Log entries are left unchanged until a system log write causes the current entry at the write index be overwritten. Reading the log returns the earliest entry not read so far. Directives are provided to allow any log entry be read and to reset the log indices.

Storage:

8. Stored statically in the *oceosmp_log_area*, which also contains the next-read and next-write buffer indices and the system state related variables.
9. The log size and the *oceosmp_on_full()* function declaration are stored in the *oceosmp_fixed_area*, as also are the log area addresses.

Log entry format (from system_log.h):

10. Struct log_entry{
```
        U64_t           time64;
        unsigned int    current_cpu_id      :8;
        unsigned int    entry_type          :8;
        U32_t           entry_comment;
} __attribute__ ((aligned (8));
```
11. When a log entry is made, *time64* and *current_cpu_id* are set automatically by OCEOSmp.
12. The *entry type* is an *enum* defined by the application, with a subset of enum values reserved for use by OCEOSmp. The reserved enum values are specified in *system_log.h*.
13. The *entry_comment* allows noting further information on the current situation

Associated directives (from system_log.h)

14.
```
S32_t   oceos_log_add_entry(
                enum LOG_ENTRY_TYPE type, // 8 bit enum LOG_ENTRY_TYPE
                const U32_t info          // further information
);
```
This adds an entry, overwriting the currently oldest entry.

15.
```
S32_t oceos_log_remove_entry(
                struct log_entry * const outputPtr
);
```
This returns the oldest unread entry to *outputPtr* and updates the buffer indices

16. Other log directives are detailed in *system_log.h*

Usage by OCEOSmp itself

17. OCEOSmp only uses the system log for problems that do not arise directly when a directive is used (problems that arise directly are described in the returned directive status code).
18. Such problems fall into a number of categories
    a. System corruption is detected and/or *oceos_exit* is called
    b. Excessive delays in obtaining system spin locks perhaps due to system overloading
    c. A timed action misses its specified time
    d. A task misses its deadlines
    e. Task start requests occur more frequently than expected
    f. A task exits without returning a mutex or read-write mutex that it holds

g. Nested mutexes or read-write mutexes not returned in correct order
h. Counts already at max or min

19. In many cases when OCEOSmp makes a system log entry it also updates the system state variable. This is updated using 'OR' and so provides a long term record that a problem has occurred even if a log entry is overwritten. In addition, if the update corresponds to an application defined action mask and an application problem handling function is defined this is called automatically.

System State Variable

Purpose:

To record that certain conditions have occurred and to allow the ASW specify which conditions should cause an ASW defined problem handling function be triggered automatically when the state variable is updated due to these conditions.

Structure:

Four 32-bit words each containing 32 flags, some available for use by the ASW, some reserved for use by OCEOSmp

1. system state variable — individual bits are set by ASW or by OCEOSmp
2. accumulated system state — updated when system state flags are set
3. action mask — set by ASW in application configuration
4. action mask previous — previous value of action mask if changed by ASW

Usage:

5. the system state variable is updated using inclusive 'OR' to set the desired flag
6. the accumulated system state flag is set when the corresponding system state flag is set
7. on reset the system state variable is cleared to 0, all flags clear
8. the accumulated system state variable is reset only by the ASW
9. the action mask is set by the ASW, and causes an ASW defined problem handler to be called when a state flag is updated to 1 that corresponds to a set action mask flag.

Whenever a system state variable flag is set during scheduling (using inclusive 'OR') the corresponding accumulated state flag is also set. Accumulated state flags are not affected when a system state flag is cleared to 0, whether by the ASW or on system reset. They can be cleared only directly by the ASW. They keep a record of events across system reset if memory is preserved.

The action mask is specified in the application configuration, which then should also declare a problem handling function. The mask may be altered by the ASW during scheduling.

The ASW problem handling function is called automatically by OCEOSmp as a result of setting a previously clear system state flag that corresponds to a flag that is set in the action mask. The system state flag is set before the function is called. If the system state flag is not reset by the ASW problem handler or otherwise then further settings of that system state flag will not cause the function to be called again.

The ASW can use a system call to modify the action mask at any time, when this is done the previous value of the action mask is stored.

The flags corresponding to the high sixteen bits of the system state variable are reserved for use by OCEOSmp:

These primarily refer to problems that are not immediate consequences of the use of a directive (these are indicated in the returned error or warning status code), but that may occur subsequently during scheduling:

16. reserved
17. reserved
18. reserved
19. reserved
20. reserved
21. Attempt to start pending task that has been disabled while pending.
22. Time from task start request to job completion exceeds task deadline.
23. Time between task start requests less than specified minimum for task
24. Time between task start requests greater than specified maximum for task
25. Missed latest time for transfer of job from timed action queue to scheduler.
26. Missed latest time for performance of timed output.
27. Mutex not returned before job terminates.
28. Read/Write mutex not returned before job termination.
29. System busy warning
30. Data area sentinels corrupted.
31. System meta pointer corrupted

System Policing and Problem Anticipation

All design involves assumptions, and 'System Policing' refers to checking how close these come to being violated as the system operates, and to taking precautions if needed.

The white box design of OCEOSmp facilitates this, with records kept automatically of task timing performances and other parameters that are accessible by the ASW at any time, including after OCEOSmp exits should this occur.

These records can answer questions such as 'what is the longest time this task took to execute', 'what is the shortest time between requests to start this task', 'what is the longest time between requests to start this task', and many others, including stack usage.

By checking such data the ASW can notice that behaviour is not as expected and that something is going wrong, allowing 'Problem Anticipation' and an issue being dealt with before it causes serious difficulties.

The ASW might do this by having an external policing function running on a core not in use by OCEOSmp that periodically checks that things are as expected, or by having a policing task that is scheduled by OCEOSmp.

With all the data stored statically in the 'Log Area', 'Fixed Data Area' and 'Dynamic Data Area' arrays by the ASW it is straightforward to retrieve it.

OCEOSmp only requires one system stack per CPU core rather than one stack per thread. Each stack is initialised to a known value, making it straightforward to determine the greatest extent to which each stack has been used.

If suspicious activity is detected, the ASW can for example disable a task or take a CPU core out of use or disable a peripheral or take other corrective action.

It can also make a historical record of bad behaviour by adding an entry to the System Log and by updating a flag in the System State variable, When the System Log becomes ¾ full an ASW function if defined is called that can archive the System Log, and a shadow copy of the System State variable is automatically maintained that is only reset by the ASW itself.

When multiple mutual exclusion semaphores are used by tasks the OCEOSmp design excludes the possibility of unbounded priority inversion and no check for this is needed.

On multi-core systems deadlocks can occur but only if tasks fail to acquire mutual exclusion semaphores in a consistent order corresponding to their identity numbers. A warning status code is then returned although as yet no deadlock has occurred.

The ASW can include a policing function that checks the execution times of tasks (in OCEOSmp tasks on other CPU cores spin waiting to acquire mutexes), and the time when a task start request was issued, allowing detection of deadlocks, livelocks, and task starvation.

It should be noted that OCEOSmp itself may detect an anomaly and automatically make an entry in the System Log and may also update the System State variable, perhaps resulting in an ASW defined problem handling function being called, but this only happens after a problem has occurred and been detected.

Problem Detection and Handling

This relates to what happens after a problem has occurred and been detected, as distinct from System Policing and Problem Anticipation which aims to forestall problems.

The integrity tests carried out automatically by OCEOSmp check for a number of problems. Some are detected directly when an OCEOSmp directive is used, others when an event occurs such as a job completing execution, others when some internal operation of OCEOSmp does not proceed as expected.

Problems that arise directly from the use or misuse of a directive are identified in the returned status code, and the ASW can then decide the appropriate action.

If appropriate the ASW can disable or terminate tasks and take CPU cores out of use without disrupting scheduling, and can read and update the system log and system state variable. In an extreme case the ASW can end scheduling and return to *main().*

1. Typical directive problems
    a. Current phase doesn't allow this directive
    b. Invalid parameters supplied
    c. Time for timed action already passed
    d. Write fail, data queue full
    e. Write succeeded, data queue now full (warning)
    f. Mutexes or rwmutexes acquired in an incorrect order (warning)

Other problems may be identified automatically be OCEOSmp after a directive has been used or when an event occurs, or during internal OCEOSmp operations

Such problems include:

2. Unexpected patterns of events
    a. Task misses a deadline
    b. Task start requests occur more frequently than expected
    c. Task start requests occur less frequently than expected
    d. Task disabled after being put on a pending queue
    e. Spin locks not being available in a reasonable time
    f. Action time missed
    g. Pending queues being full

These problems typically cause a system log entry to be made, may update the system state variable and may result in an ASW defined problem handling function being called. This has full access to the performance records and can enable or disable tasks and take CPU cores out of use or restore them to use without disrupting scheduling, and can exit from OCEOSmp if appropriate.

Other problems that may be automatically detected by OCEOSmp include

3. Corruption of critical data
    a. Perhaps due to hardware failures
    b. Perhaps by application software malfunctioning and overwriting data
    c. Perhaps by a bug in OCEOSmp

These problems typically result in exit from OCEOSmp and return to *main()* at the point immediately after *oceos_start()* was used. The system state variable and system log may also be updated.

It should be noted that the checks made by OCEOSmp only occur as a result of it being notified of some change in the state of the system, as when a directive is used or a job terminates or an interrupt occurs. It is recommended that these be added to by taking appropriate precautions.

Precautions and Correction

A primary precaution is policing. As described above this typically involves assigning a CPU core to check the activities of the system, or perhaps assigning a task to that purpose. Anticipation and prevention are better than detection.

Further precautions are also possible, depending on the hardware available in the system. These can detect or perhaps even prevent various malfunctions.

Hardware Malfunctions:

These can be caused by radiation or other external factors, or simply by parts of the system wearing out.

They fall into three main categories, memory hardware malfunctions, CPU core malfunctions, and peripheral or connected device malfunctions.

Memory hardware malfunctions

These typically result in one or more bits in a memory location being set to incorrect values.

If the error involve no more than a certain number of bits (BCH 1, RS fixed > 1) then EDAC allows a memory location containing such errors be read correctly despite the error.

While the value read is correct the error remains in the location until the correct value is written back. This should be done before a further bit becomes corrupted, perhaps making the error un-correctable. In some systems this write back is done automatically.

Long intervals can occur between uses of a memory location making 'memory scrubbing' necessary. This cycles through all memory locations, reading and writing back if needed, doing so often enough to make it very unlikely that an un-correctable number of erroneous bits will accumulate in any location.

Usually a hardware 'memory scrubber' is available that can be configured to do this at a certain rate for a certain range of addresses.

Memory read errors can still occur however. A combination of bit errors can defeat EDAC error detection and lead to an incorrect value being read. Usually however such un-correctable errors are detected by EDAC causing it to raise an exception. The corresponding handler must then take appropriate action, which may include terminating OCEOSmp.

Precautions: Before starting OCEOSmp

1. The handler for EDAC detected un-correctable errors should be put in place
2. The memory should be initialized
3. The EDAC system configured and put into operation
4. The EDAC memory scrubber hardware configured and put into operation

Once this has been done and OCEOSmp started

5. The EDAC exception handler can
    a. Update the system log
    b. Update the system state variable
    c. Start a problem handling task
    d. If necessary call *oceos_exit()* to return to *main()*

Computer core hardware malfunctions

Cores may use triple modular redundancy and other techniques to reduce the effects of transient errors, but nevertheless can become faulty. If this is detected in a multi-core system OCEOSmp allows the faulty core be taken out of use without scheduling being disrupted.

Detecting that a core has become faulty can be done using computational tasks for which the results including timings are known. Before such tests are used however a malfunction may have led to an erroneous result being used.

To avoid this when result accuracy is critical and immediately needed it is possible in OCEOSmp to have multiple CPU cores carry out the same computation at the same time (within a few clock cycles). The results can then be compared and the majority agreed value used. A core that was at fault can be tested and taken out of use if the problem is judged not to be transient.

Precaution: Check periodically that cores carry out a computational test correctly.

6. *oceosmp_cpu_disable()* will take the specified CPU core out of use, put it to sleep, and cause the termination functions of any jobs executing on the core be executed on a different core.
7. *oceosmp_cpu_enable()* will make the specified CPU core available for use

These directives can also be used to manage the number of active cores for power saving or other purposes. Scheduling is not disrupted as long as a functioning core remains.

Peripheral Hardware Malfunctions

These can arise due to a fault in the peripheral unit itself, or due to problems in an external system for which the peripheral is an interface.

Before starting OCEOSmp the ASW should carry out appropriate checks of all peripheral units, and should ensure that such checks can be carried out if necessary by tasks after scheduling has begun.

Faults in external systems for which a peripheral unit is an interface tend to either

(i) cause no messages to be sent to the interface
(ii) cause messages to be sent too frequently

In most cases when a peripheral receives an external message it causes an interrupt which then causes a request to start a task.

If the interval between such task start requests is longer than expected this may indicate that the peripheral device has become faulty and is no longer operational.

If the interval between such task start requests is shorter than expected it may indicate the frequency of external events exceeding expectations or some fault in the peripheral device.

In both cases on starting the task OCEOSmp automatically checks the interval since the last task start request and if this is less than the expected minimum or greater than the expected maximum will make a system log entry and set the corresponding flag in the system state variable. If the ASW has set the corresponding system state action flag and defined a problem handling function this will result in the ASW problem handling function being called

Precautions:

In the ASW configuration structure passed to *oceos_init()* the ASW set the desired flags in the problem action mask and define a problem handling function.

Software Malfunctions

Certain types of software errors can cause problems for the system as a whole.

OCEOSmp automatically checks for some of these, others are best checked for by the ASW itself, perhaps as part of system policing.

In addition to policing, various precautions can be taken against these problems, including making use of any memory protection hardware available and using a system watchdog timer.

Invalid memory overwriting

Application software may inadvertently overwrite memory areas used by OCEOSmp.

The three memory areas used by OCEOSmp are each protected at their ends by constant sentinels. Each time a directive is used or job terminates OCEOSmp checks these sentinels to ensure that the areas involved have not been encroached upon. If so OCEOSmp will exit.

In addition to its sentinels the fixed data area has an XOR checksum which allows any alteration be detected. This is not checked automatically by OCEOSmp for reasons of efficiency, but can be checked at any time by ASW policing software.

Prevention is better than detection and if a system has a memory protection unit this should be configured to make the fixed area read-only. Any attempt to write this area will then typically cause the protection unit to raise an exception.

Each CPU core has its own system stack. These form part of a contiguous block of statically allocated memory, and a software error may result in the code running on a core overrunning that core's stack and attempting to use the stack memory of another core.

This can be detected in policing by checking that filler still remains at the top and bottom of each core's stack, and this can be checked at any time by ASW policing software, but prevention is better than detection.

Precautions: Any hardware memory protection present should be used on completion of *oceos_init_finish()* and before *oceos_start()* to provide these features.

(i) Read only (the fixed data area)
(ii) Writeable only by a specific computer core (for each core's stack)

Invalid attempts to use a memory location typically give rise to an exception and an appropriate handler must be in place.

Invalid use of Mutual Exclusion Semaphores (Mutexes) or of Read-Write Mutexes

In general the time for which the ASW holds a mutex or read-write mutex should be as short as possible.

This time can be estimated at compile time and the estimates taken into account in determining whether a set of tasks can be scheduled so that tasks always meet their deadlines.

Such estimates however can be made wrong by events that occur during scheduling if appropriate precautions are not taken.

One possible problem, unbounded priority inversion, arises when a job holding a mutex is pre-empted by a higher priority task that does not use the mutex, causing any even higher priority task that uses the mutex to wait indefinitely for the low priority job to resume and release the mutex. In OCEOSmp unbounded priority inversion cannot occur.

Another problem, deadlock, can arise when two or more jobs each acquire more than one mutex. A job while holding a mutex may then attempt to acquire a different mutex currently held by some other job. If that other job (without releasing the mutex it holds) attempts to acquire the mutex held by the first job neither job can make progress, the jobs are deadlocked.

These problems are addressed by the design of OCEOSmp.

In OCEOSmp each mutex or read-write mutex has a priority ceiling giving the priority of the highest priority task that uses it. This constant is determined at compile time and set for each mutex when it is created.

In scheduling, once a job running on a CPU core holds one or more mutexes or read-write mutexes only tasks with higher priority than the highest ceiling are allowed to pre-empt the job and start running on that core.

As a result unbounded priority inversion cannot occur, and on single core systems deadlocks also are not possible.

On multi-core systems jobs can spin waiting to acquire a mutex held by a job running on a different core and deadlock becomes possible.

In OCEOSmp mutexes and read-write mutexes are assigned a sequential order as they are created. Deadlocks then cannot occur if jobs always respect this order in acquiring mutexes, never attempting to acquire a mutex or read-write mutex if already holding one that is higher in the order.

OCEOSmp does not prevent jobs from acquiring multiple mutexes in an out of order manner but returns a warning directive status code to the job indicating that deadlock is possible and makes a log entry and updates the system state variable, perhaps causing an ASW defined problem handling function to be called.

When a job simultaneously holds multiple mutexes or read-write mutexes it should acquire and return these in a LIFO manner, e.g. waitA, waitB, waitC…..signalC, signalB, signalA. Failure to do so may affect task scheduling temporarily and will result in a warning directive status code, a log entry, and an update of the system state variable.

Other errors in using mutexes such as failure to release a mutex before a job ends or attempting to release a mutex that is not held give rise to an appropriate directive status codes, a log entry, and a system state variable updates.

Precautions:

(i) Ensure at compile time that when a task acquires multiple mutexes or read-write mutexes it always does so in the correct order.

(ii) Ensure at compile time that when a job releases multiple mutexes or read-write mutexes it always does so in the reverse order to that in which they were acquired

Deadlocks, Livelocks and Task Starvation

Deadlocks arise primarily due to misuse of mutual exclusion semaphores or read write mutual exclusion semaphores, as described above.

Livelocks, where tasks interact in a way that results in each task having to return to some earlier point with no or very little overall progress, can arise in a number of ways including from attempts to resolve a deadlock so that the tasks involved can proceed.

Task starvation arises when a task is ready to start, but CPU time is not allocated to it within a reasonable time. In OCEOSmp this will occur for a low priority task if higher priority tasks are always ready to run and require all the available CPU cores.

Task starvation, to at least some degree, can also occur for a high priority task if the resources required to allow it pre-empt a lower priority task are not available on the CPU core involved, as for example when the remaining system stack for that core is below the level needed for pre-emption.

All three of these problems result in a task or tasks taking longer to complete than should be the case. This can be detected in two main ways

(i) By a policing task that periodically checks each task record in the dynamic area to determine the maximum time between a request to start it and task completion.

(ii) By watchdog timer hardware that typically is reset periodically by the lowest priority task and which will cause a hardware system reset if its pre-set time elapses before it is reset. On re-entry to main() the ASW can check to determine if this has occurred before restarting OCEOSmp.

Precautions

(iii) ensure each CPU core has adequate system stack so that a lower priority task running on the core can always be pre-empted by a higher priority task. The amount required can be calculated from the number of different priority levels in use.

(iv) if watchdog timer hardware is available set up a low priority task to periodically reset it. The ASW should check on entry to *main()* whether this was due to a watchdog timeout and if so use the OCEOSmp data to determine the cause and appropriate remedial action before restarting OCEOSmp.

(iii) set up a policing task that checks the execution times of current tasks and detects when this is too long. This task can call an ASW defined problem handling function that can determine how best to deal with the situation, perhaps calling the termination functions of some currently executing tasks, disabling some tasks or changing interrupt intervals.

(iv) If necessary reserve certain CPU cores to use only by higher priority tasks, perhaps ensuring that there is always at least one CPU core ready to be used immediately by such tasks.

Summary

Murphy's Law, 'if things can go wrong they will', has had a significant impact on the design of OCEOSmp.

An older maxim 'the wise man avoids evil by anticipating it' (Publilius Cyrus) has also had an impact. Software can go wrong without prior indication of stress but it may be possible to notice that all is not well before disaster strikes. Avoidance is better than detection.

These ideas have led to OCEOSmp having features that allow fault anticipation, detection, isolation and recovery, with a white box approach that simplifies system policing and checking behaviour against expectations and with memory statically allocated.

When a problem is found OCEOSmp can take CPU cores out of use or kill or disable tasks without disrupting scheduling, or in severe cases exit and return to the ASW *main()* with all data still accessible for analysis or download, and with the ASW able to make changes and restart OCEOSmp if desired.

These features help ensure that OCEOSmp is suitable for use in safety critical or other systems that require the highest reliability.

Even where it is hoped that nothing can possibly go wrong….

For more information please contact sales@ocetechnology.com