1044414: Advanced Operating Systems and Virtualization **Spring 2018**

Lecture 14: April 24

Lecturer: Alessandro Pellegrini *Scribe: Anxhelo Xhebraj*

14.1 Task vs Processes

A task is something that is associated with some management of interrupt services. Processes: execution traces related to user mode or kernel mode programs.

Receiving an interrupt means that someone is asking the attention and this materializes a task. We don't want to block processes since it may make the performance worse. Interrupt management is shifted in time by delaying the execution. With multiple interrupts, multiple tasks are materialized. Multiple tasks can be aggregated into a single process/thread execution trace.

Many to one aggregation: we must ensure that no task is subject to starvation. There should be some priority scheme.

Reconciliation point: a point in time convenient to process the interrupts. In this time no critical section. This scheme has weaker guarantees about the processing of interrupts with respect to usual meaning of interrupts.

All the operating systems handle interrupts in this way. Conventional conciliation points are

- Returning from a systemcall: check if there are tasks pending
- Before executing a context-switch: if the next process is an idle process it might be convenient to process tasks.
- Specific kernel thread points. Points in kernel code such that no critical sections are involved or points with logically completed operations.

The management of interrupts must be split into at least two part: the first part that serves the interrupt in order to let the device continue executing and the second which is really processing the interrupt.

In the second part we can have many to one aggregation to make the processing more lightweight.

top half: tiny amount of work to setup a task (bottom half) and the data structures necessary to finalize and manage the interrupt. Portion of code executed with interrupt flags cleared (requests disabled).

bottom half: similar to process management. As soon that it can be executed it will.

Historical data structures were Task queues up to version 2.5. Then tasklets, soft irqs etc were introduced.

14.2 Task Queues

Some task queues were associated with specific reconciliation points letting the developer choose at what point a task was processed. The three task queues were:

tq immediate: executed when returning from syscall or timer interrupt

tq timer: only on timer interrupt

tq scheduler: checked on processes that can live on userspace but not checked on kernel only threads.

Additional task queues could be defined.

A task is represented by struct tq struct. sync is a member that needs to be initialized to 0 to let the api properly manage the data structure.

A task can be queued through queue task and can be flushed through run task queue (all tasks of the queue are processed).

Sync is internally used to inform whether it is pending (1).

schedule task is used as run task queues but specific for tq schedule queue.

mark bh to inform that run task queue has bottom half code to be executed.

software irq (softirq): as an interrupt request but just generated by software to notify the kernel that something is to be done. Do bottom half is called in schedule and ret from sys call. Execution is done in process context. Blocking had not to be executed in botoom halfs.

Task queues limitations: single thread execution, heavy interrupt load made the performance worse since all tasks had to be executed.

Task queues are no longer used from version 2.6.

14.3 Tasklet

Definition of a task. A Tasklet represent only one task and not a task queue. Tasklet can be declared as enabled or disabled. The former is run whenever possible while the latter only when it is enabled. The latter is used to register the tasks with not so high priority.

tasklet schedule puts a tasklet into a pool from which the kernel takes and executes tasklets. From this pool tasklets can be executed on specific kernel threads. In tasklet you can use blocking services but it is highly discouraged since it blocks the kernel thread that executes tasklets.

Tasklets are run using sofirqs. They can be masked.

Soft irqs are fired after running an hardware interrupt handler. The dispatcher calls the interrupt handler and when it returns the dispatcher checks whether the softirq flag is set and if so tasklets are run.

14.4 Work Queues

Higher latency than tasklets but solves most of the problems of tasklets. More granularity and control on tasklet management. Blocking calls are allowed but discouraged.

work struct: a data structure that can be queued and a reach api to work on them. Can define a delayed work: it can be processed later. Defferrable can be executed even with more delay.

delayed work: task should be delayed and by the timer it is defined a priority.

work queue struct: pool of workers that tells which threads are in charge of executing the tasks of this queue.

14.5 Kernel Timers

A jiffie is a unit of time. A global variable is kept with the number of jiffies that have occurred since the system was booted. The time quantum given to a process is a multiple of jiffies.

hlist description: prev of hlist node has a double pointer to the content of next of the previous node.

timer wheel: is a multi level hash table. 5 levels of time, each one chains the timers according to the scheme shown aboves.

14.6 Final Project Information

Implement fiber support in the linux kernel.

References