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

Lecture 15: April 27

Lecturer: Alessandro Pellegrini *Scribe: Anxhelo Xhebraj*

15.1 Generic Lists in Linux

The prev pointer in `list_head` usually points to the entry of the struct that points to the node instead of the start of the struct. But from this fact how can we retrieve the start of the struct given the pointer to one of its entries? `offsetof` is able to return, given a member of a struct, its offset from the beginning of the struct. `container_of` uses `offsetof` to get the struct containing a given entry pointed by `ptr`.

15.2 Timer Interrupts Management

15.2.1 Up to 2.4

Upon receiving the timer interrupt the jiffies variable is incremented and it is checked whether the time quantum for a running thread has expired. In that case a bottom half is scheduled and `need_resched` is set to inform upon a reconciliation point preempt this thread. The reconciliation point is when returning to the dispatcher and therefore the scheduler is called (bottom half).

The top half of the handler is implemented in `do_timer` which increments the `jiffies`, updates the time of the running process and marks that the bottom half needs to be checked.

Upon receiving the timer interrupt (IRQ0) first of all the context of the registers since it is a precise interrupt (can happen at any moment in time). Afterward the handler is called (`do_timer_interrupt`) which will eventually call `do_timer` that on `update_process_times` will set `need_resched` if the time quantum has expired.

15.2.2 From 2.6 to later

With the introduction of the LAPIC the timer interrupt management is reimplemented relying on this new technology.

`set_irq_regs` returns a pointer to a `pt_regs` which is a cpu snapshot of the previous cpu context.

The interrupt handler is `local_apic_timer_interrupt` that takes some cpu variables, computes some stats and finally calls the event handlers.

15.2.3 High-resolution timers

Allow to specify higher granularity timers mainly used in real time operating systems.

Clock id tells which clock the initialization should be done for. This is a posix specifications that tell the kinds of codes but some of them cannot be implemented in constrained architectures.

`need_resched` is replaced with a call to a function that checks one bit of a per cpu variable.

15.3 Process Control Block

One for each thread. `volatile long state` is tells the current state of the thread (running, wait etc). The two `mm_struct` are used to manage the user space view of memory. The PCB identifies the thread but it keeps also information to which process the thread is related. The `pid` is what is exposed at user space as `tid` which is the thread id. While `tgid` keeps the id of the process to which the thread belongs and stand for *Thread Group Id*. `ppid` stores the `tgid` from which the process was forked. `thread_struct` thread is an architecture specific data structure which keeps various information about the state of the cpu such as TSS, FPU, CR2 etc. `int prio` is used to implement the `nice()` system call to specify the priority of the thread. `policy` which will see later. Finally two entries for the *affinity* of the thread to the various cpus.

In `mm_struct` there is the virtual address of the page table (`pgd`). Upon scheduling the scheduler will look into the PCB, check the `mm_struct` and sets `cr3` with the translation of the `pgd` entry. To describe the process virtual address space in a more generic way there is `vm_area_struct` that describes how the memory is mapped in the address space of the process.

`active_mm` is used to steal an mm when running in an anonymous process. Kernel level threads are threads that do not have a user space address. For these threads we either setup these data structures also for this kind of threads or steal it from a user space process. For kernel threads mm is set to null and active mm is set to the virtual address space of the user space thread. For user space threads `active_mm == mm`. A memory view is necessary for kernel threads since some facilities require in read mode mm data structures just to perform some corner case tasks.

15.3.1 VM area struct

When `mm`apping memory the kernel is lazy meaning that until it is really accessed it is not allocated really. Once it is accessed a minor page fault occurs and memory is allocated to the process requesting.

15.4 PCB Allocation

Up to kernel 2.6 the PCB was placed at the base of the kernel level stack. The kernel level stack was just 2 frames (8KB) which was also a constrain on the maximum number of calls that could be performed in kernel mode by the process.

From 2.6 the PCB was moved to another place in memory and at the base of the stack there was `thread_info` with a member called `task` that pointed to the PCB. Also the kernel level stack could be 2 or 4 memory frames. There is a problem with kernel level stacks which is that they must be contiguous and are taken from zone normal which has high contention.

The kernel level stack was defined as `thread_union`. The linux kernel level stack has always been the weak point in terms of security and it required some time to be re implemented. `thread_info` contains the pointer to the PCB, a set of flags, times that the thread has been preempted etc. `addr_limit` tells at what point there is a switch from kernel level addresses to user level addresses. `access_ok()` was just looking to this member to know whether it was safe to access memory instead of looking at all the memory management data structures.

`sysenter_return` is used for systemcalls related to timing events. For example if a thread executes `sleep(10)` but the system call fails for some reason with `EINTR` it would be useful

to know how much time has slept.

Kernel level stacks are contiguous and taken from physical memory. Overflowing the stack means that we're touching physical memory assigned to data structures of the kernel.

`thread_info` was moved to the PCB and the stack is just kernel level stack. Also the stack is not contiguous but `vmalloc`ed. This introduced great latency.

`current` returns memory address of the PCB. Previously it used the thread info and then it became a per cpu variable that is set any time a thread is scheduled.

References