Linux Internals
Day 2
The Linux Kernel

Joel Isaacson
Ascender Technologies Ltd

The task_struct

- include/linux/sched.h

```
struct task_struct {
       /* -1 unrunnable, 0 runnable, >0 stopped */
        volatile long state;
        struct thread_info *thread_info;
        atomic_t usage;
       /* per process flags, defined below */
        unsigned long flags;
        unsigned long ptrace;

        int lock_depth;          /* BKL lock depth */
```

The task_struct

- include/linux/sched.h

```
#ifdef CONFIG_SMP
#ifdef __ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif
/* for niceness load balancing purposes */
        int load_weight;
        int prio, static_prio, normal_prio;
        struct list_head run_list;
        struct prio_array *array;

        unsigned short ioprio;
#ifdef CONFIG_BLK_DEV_IO_TRACE
        unsigned int btrace_seq;
#endif
```

Process Scheduling

- The process scheduler is the component of the kernel that which process to run next.
- It is the subsystem that defines how a finite resource (CPU time) is divided between processes

Process Scheduling

- The Linux scheduler is a clear derivative of the first Unix scheduler of over 30 years ago.
- The scheduler concept and algorithm is clearly very successful
- The scheduler is designed to give the apparent impression that multiple processes are running simultaneously.

Process Scheduling

- The design criterion of the scheduler are:
  - Keep the CPU busy.
  - Don't perform too many context switches.
  - Give apparent priority to interactive processes.
  - Be fair.
  - Avoid starvation.
  - Provide per-process processor affinity.

Process Scheduling

- On uniprocessor machines give an illusion of multiple processes running simultaneously.
- On multiprocessor machines actually run processes in parallel.

Process Scheduling

- At any one time there might be hundreds of processes that are present.
- Normally on a desktop machine only very few of these processes will actually want to run (be runnable).
- The rest of the processes are blocked, waiting for something to happen.

Process Scheduling

- Large servers (e.g. web servers servicing hundreds of users) can have a large number of runnable processes.
- In the Linux 2.6 kernel the scheduler was adapted to this challenge. This change is called the O(1) scheduler.
- The external behavior of the scheduler remains the same.

Process Scheduling

- Multitasking operating systems come in two varieties:
  - Cooperative multitasking (e.g. Windows 3.1, RTOS's, Mac OS 9)
  - Preemptive multitasking (e.g. Windows XT, Unix, Linux
- In preemptive multitasking processes are typically given a maximum timeslice.

Process Scheduling

- The timeslice (or quantum) is the slice of processor time that the process can use before it is involuntary preempted.
- The Linux scheduler calculates this timeslice dynamically.

Process Scheduling

The fundamental task of the scheduler is to choose, at any one time, from the set of runnable processes, ***m***, only ***n***, where ***n*** is the number of CPU's present, processes to run.

If ***m*** is less than ***n*** this task is trivial.

Process Scheduling

- There are two conflicting goals to the scheduler:
  - Provide low latency response.
  - Maximize CPU utilization.
- In order to try to satisfy both goals Linux classifies processes as either:
  - I/O bound
  - Processor bound

Process Scheduling

- The scheduler has a very simple algorithm. It provides a strict priority-based scheduler:
  - Processes with higher priority run before processes of lower priority.
  - Processes with higher priority are given larger timeslices.

Process Scheduling

- Where Linux(Unix) is sophisticated is in calculating the priority/timeslice.
- Linux dynamically calculates the priority from a number of inputs:
  - The past behavior of the process (I/O vs processor bound)
  - The nice value. Which has a small effect.
  - The real time priority. Which has an overriding effect.

The Timeslice

- The timeslice is the amount of time a process has to run until it is preempted.
  - I/O bound processes need smaller timeslices
  - Processor bound process need larger timeslices.
- Actually the Linux does exactly the opposite.

The Timeslice

- I/O bound processes get higher priorities and timeslices. The timeslice is not reset everyime the process is rescheduled.
- Processor bound processes get lower priority and smaller timeslices.
- This enables higher priority process to run longer and more often.

An Example

- Consider an example of two processes:
  - One process is compute bound
  - The other process is I/O bound.

The sched.c file

- We will look at:
  - struct prio_array
  - struct rq
  - sched_find_first_bit()
  - schedule()
  -