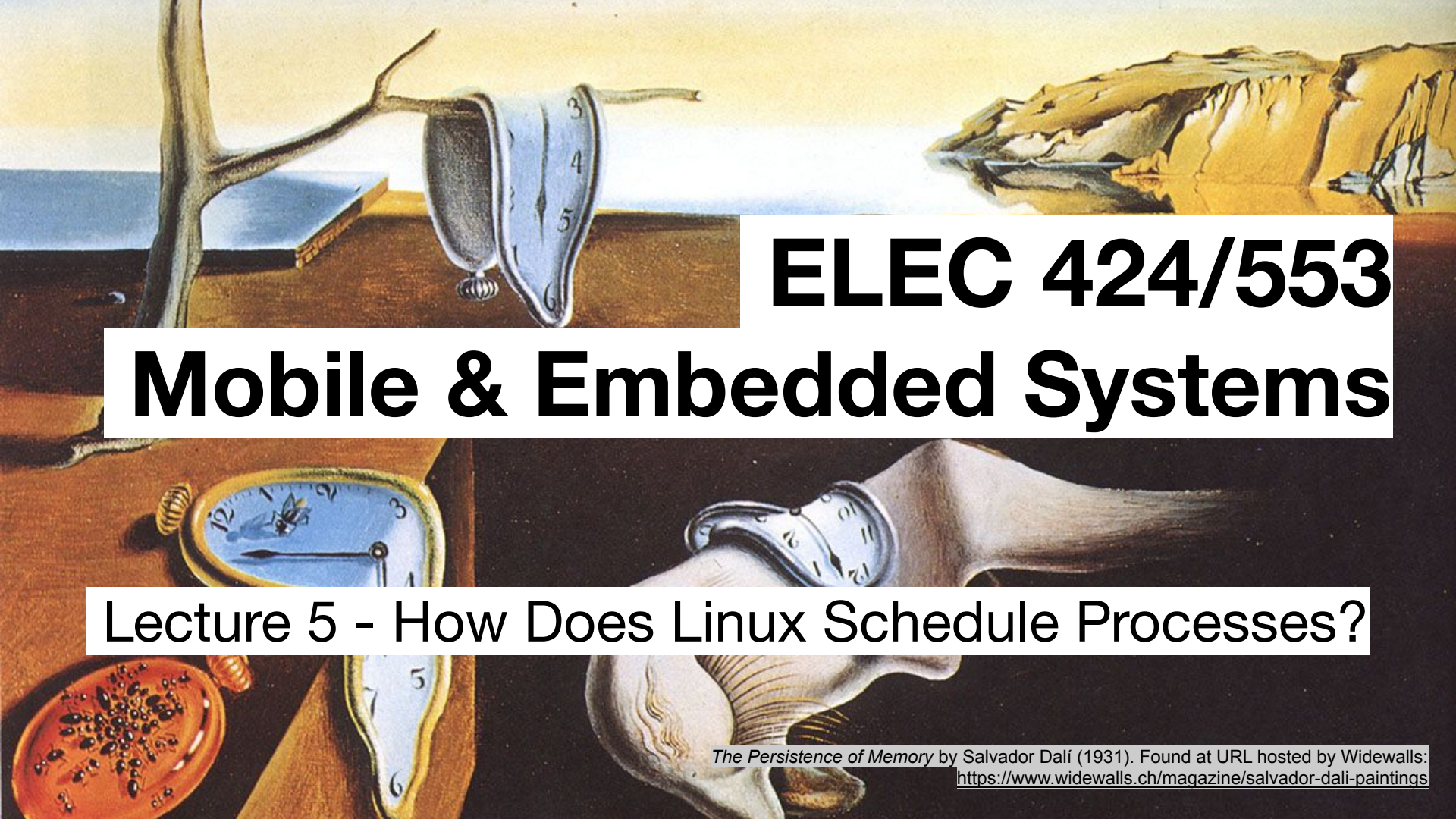


ELEC 424/553
Mobile & Embedded Systems

Lecture 5 - How Does Linux Schedule Processes?

The Persistence of Memory by Salvador Dalí (1931). Found at URL hosted by Widewalls: https://www.widewalls.ch/magazine/salvador-dali-paintings

Housekeeping

- Office Hours - Tuesdays 3-4pm - Duncan Hall 2098
    - Raise your hand if this time works for you
- Or by appointment

Housekeeping

- Project 1 posted
- Setting up your Raspberry Pi Zero W
- Due Mon 9/18 11:59pm

fork() demo

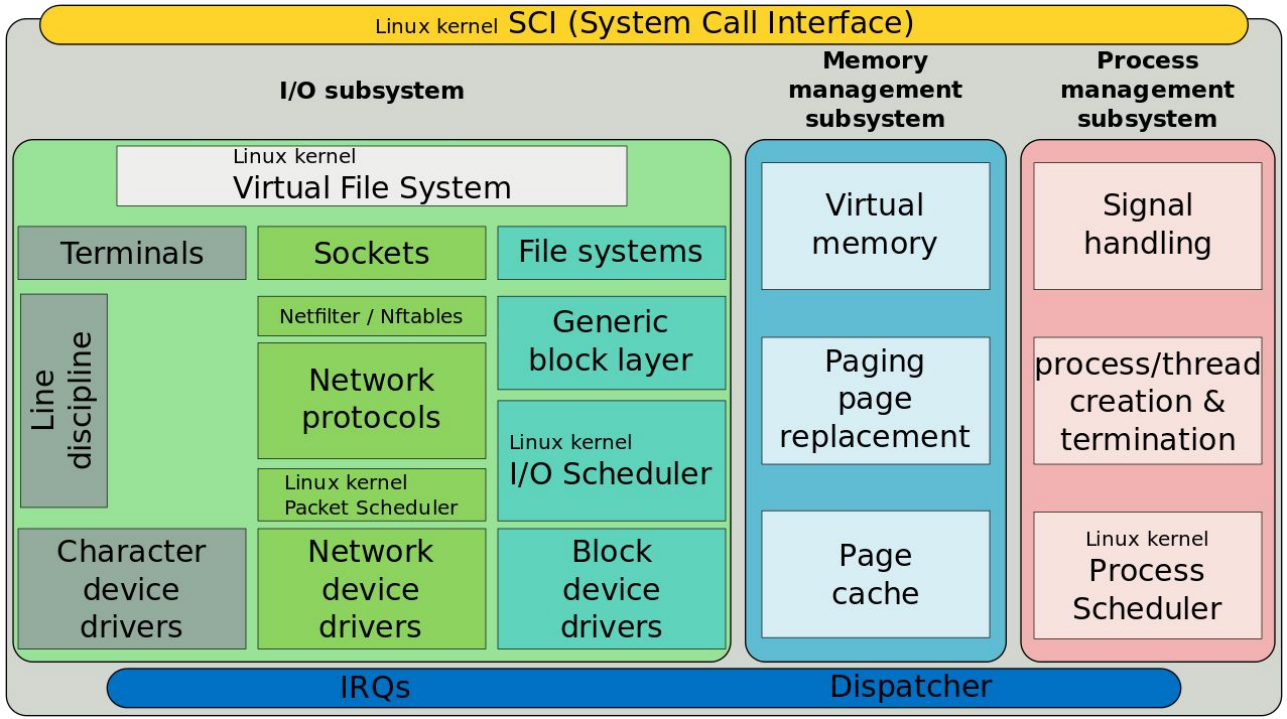
Exercise 4

Access Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
```

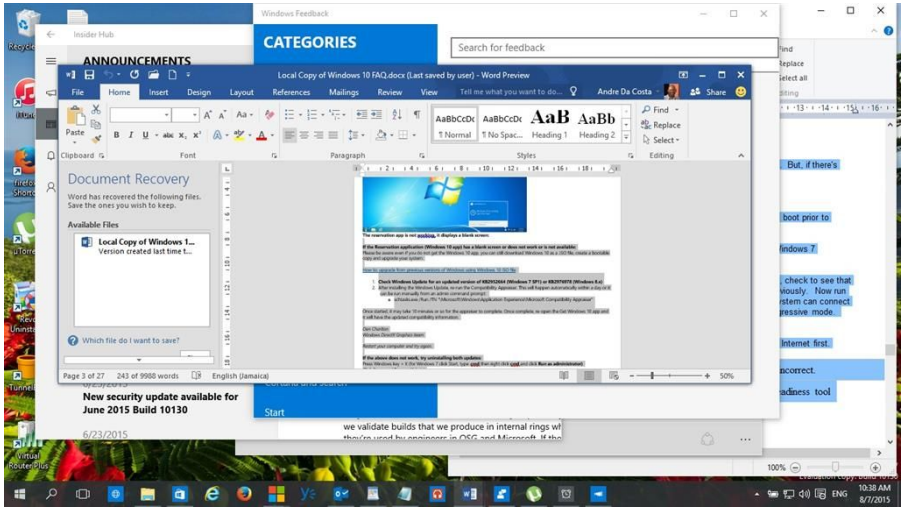
Use waitpid(pid, 0, 0);

Where Are We In The Linux Kernel?



Scheduling Processes

- Running a bunch of applications
- Multitasking makes it look like OS is simultaneously executing more than 1 application/process
    - Multicore: Simultaneous execution possible
- **Multitasking OS**: Capable of alternating between processes



Andre Da Costa, “How to: manage running programs and virtual desktops using Task View in Windows 10”. URL: https://answers.microsoft.com/en-us/insider/forum/all/how-to-manage-running-programs-and-virtual/17d068b7-5e4a-4351-a019-afa528a81538

Robert Love, *Linux Kernel Development*, 3rd Edition

Processes Don’t Run All The Time

- **Block/sleep**:
  - Waiting on something to do before running
  - Still taking up memory
  - Not **runnable**
  - Kernel will react when, for example, a key is finally pressed

Robert Love, *Linux Kernel Development*, 3rd Edition

Two Types of Multitasking

Cooperative

From People.com. Credit: KEVIN MAZUR/WIREIMAGE. URL: https://imagesvc.meredithcorp.io/v3/mm/image?url=https%3A%2F%2Fstatic.onecms.io%2Fwp-content%2Fuploads%2Fsites%2F20%2F2019%2F08%2Fswift-2009-5.jpg

Preemptive

https://assets.teenvogue.com/photos/570cf9284005974b596c4391/master/pass/Kanye.jpg

Robert Love, *Linux Kernel Development*, 3rd Edition

Two Types of Multitasking

Cooperative

Process has the power

Process can give up (yield) execution whenever it wants

Preemptive

Kernel has the power

Scheduler can come in and swap out which process is executing

Timeslice
Non-preemptable time for an executing process

Robert Love, *Linux Kernel Development*, 3rd Edition

Original Linux Scheduler

- Linux up to 2.4
    - Higher number of processes/processors would overwhelm scheduler

Robert Love, *Linux Kernel Development*, 3rd Edition

O(1) Scheduler

- Linux 2.5
- O(1): Independent of # of processes, constant-time scheduling
- Pro: Scaled much better than prior scheduler
- Con: Latency issues
    - Interactive processes
- **Completely Fair Scheduler (CFS)** introduced in Linux 2.6.23 to take spot of O(1)

Robert Love, *Linux Kernel Development*, 3rd Edition

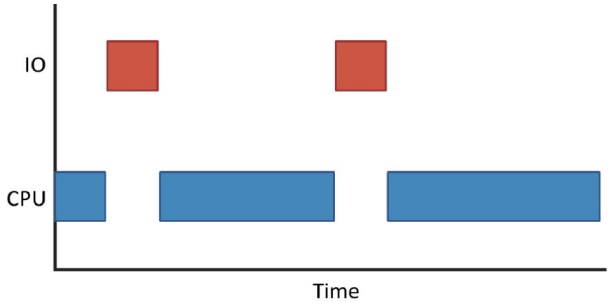
Policy

- What is the decision process for when to execute tasks?
- Processes can be:
  - I/O-bound
  - Processor-bound
- Processes have a **priority**
- Timeslices

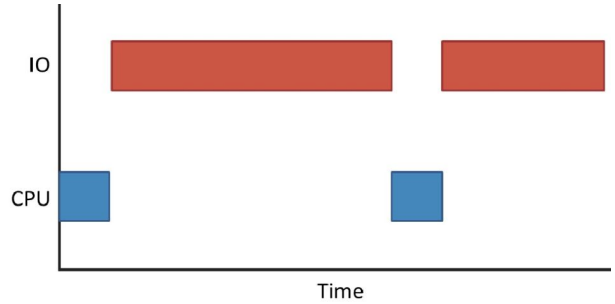
Robert Love, *Linux Kernel Development*, 3rd Edition

Process: I/O-Bound or Processor-Bound?

- How is the process investing its time?
  - Running code?
  - Handling I/O requests?
- Processor (CPU)-bound example
  - Infinite loop
  - MATLAB
- I/O examples
  - Solid state drive
  - Keyboard + mouse
  - Network



(a) CPU-bound application



(b) IO-bound application

Robert Love, *Linux Kernel Development*, 3rd Edition

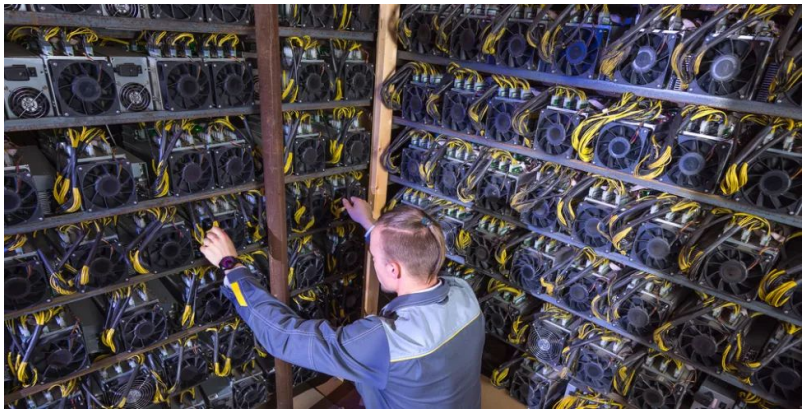
Modified from: Fernandes et al. *Computers & Electrical Engineering*. (2016)

A Fundamental Tradeoff

Low latency vs. High throughput

Responsiveness vs. Utilization

From VentureBeat. Image Credit: 20th Century Fox. URL: https://venturebeat.com/2016/11/27/minority-report-science-advisor-builds-the-most-awesome-conference-room/



From: Tom's Hardware. Image credit: Shutterstock. URL: https://www.tomshardware.com/news/china-clampdown-crypto-mining-gpu-prices-plunging

Robert Love, *Linux Kernel Development*, 3rd Edition

Process Priority

- Consider **priority-based** scheduling
- Ranking
- Round-robin for equal priority
- Priorities in Linux:
  - **nice** (view using `ps -el`)
    - (higher priority) [-20, -19, …, 18, 19] (lower priority)
  - **Real-time (RT) priority** (view using `ps -eo state,uid,pid,ppid,`**`rtprio`**`,time,comm`)
    - (lower priority) [0, 1, …, 98, 99] (higher priority)
    - RT processes > normal processes in terms of priority

Robert Love, *Linux Kernel Development*, 3rd Edition

Timeslice

- Task can execute for a timeslice
- How long is a good standard timeslice?
- A number of OS's shy away from long timeslices
  - E.g. 10 ms
- Processes get dynamic timeslices from CFS
  - Priority will increase or decrease timeslice (relatively)

Robert Love, *Linux Kernel Development*, 3rd Edition

Scheduling Case Study

- Text editor (I/O-bound)
- Video encoder (processor-bound)
- Text editor barely needs CPU time
  - But you expect it to respond quickly
- Video encoder needs a ton of CPU time
  - But a difference of a second would be unnoticed

Robert Love, *Linux Kernel Development*, 3rd Edition

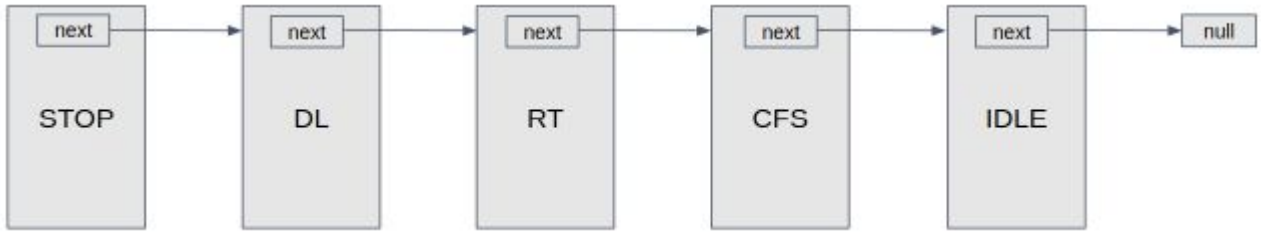
Scheduling Case Study (Continued)

- Encoder will get full CPU time when no key pressed
- Key pressed! What happens?
- Editor awakens (blocked->runnable)
- CFS sees editor has used barely any time
- CFS preempts encoder
- CFS runs editor
- Cycle repeats until the end of history

Robert Love, *Linux Kernel Development*, 3rd Edition

Scheduler Classes

- Modularity of Linux scheduler
- Can use particular scheduler for particular process types
- Priority ranking for process types
- Each class gets a sub run queue (rq) which contains runnable tasks
- Relevant files in `/kernel/sched/`
  - `core.c`
  - `fair.c`



Viresh Kumar, “Fixing SCHED_IDLE”. URL: https://lwn.net/Articles/805317/

Robert Love, *Linux Kernel Development*, 3rd Edition

Scheduler Classes

/ kernel / sched / core.c All

```
5902    * WARNING: must be called with preemption disabled!
5903    */
5904   static void __sched notrace __schedule(bool preempt)
5905   {
5906           struct task_struct *prev, *next;
5907           unsigned long *switch_count;
5908           unsigned long prev_state;
5909           struct rq_flags rf;
5910           struct rq *rq;
5911           int cpu;
5912
5913           cpu = smp_processor_id();
5914           rq = cpu_rq(cpu);
5915           prev = rq->curr;
```

/ kernel / sched / sched.h All sym Search

```
905   /*
906    * This is the main, per-CPU runqueue data structure.
907    *
908    * Locking rule: those places that want to lock multiple runqueues
909    * (such as the load balancing or the thread migration code), lock
910    * acquire operations must be ordered by ascending &runqueue.
911    */
912   struct rq {
913           /* runqueue lock: */
914           raw_spinlock_t          __lock;
915
916           /*
917            * nr_running and cpu_load should be in the same cacheline be
918            * remote CPUs use both these fields when doing load calculat
944           unsigned int            uclamp_flags;
945   #define UCLAMP_FLAG_IDLE 0x01
946   #endif
947
948           struct cfs_rq           cfs;
949           struct rt_rq            rt;
950           struct dl_rq            dl;
```

Linux 5.14.2 Source via Bootlin Elixir Cross Referencer
Left: https://elixir.bootlin.com/linux/v5.14.2/source/kernel/sched/core.c#L5904
Right: https://elixir.bootlin.com/linux/v5.14.2/source/kernel/sched/sched.h#L912

Next Lecture:

Scheduling Continued