Tasks and KUDOS

CompSys 2016

Oleks & Nicklas

DIKU

November 2, 2016

Overview

- Recap: system calls, processes, threads
- `clone(2)`: a common ancestor
- Scheduling
- `KUDOS`

KUDOS (1/2): What Is KUDOS?

KUDOS is a skeleton operating system
for exploring operating systems concepts.

It is intended for teaching operating system concepts, and to serve as a baseline for open-ended student projects.

- Based on BUENOS (see also Aalto University, Finland)
- KUDOS is a continuation of the BUENOS effort at DIKU
- BUENOS targeted the MIPS32 architecture
- KUDOS can (now, also) run on x86_64

`KUDOS` (2/2): More `KUDOS` After Lunch

Nicklas will give a hands-on introduction after lunch

What to eat for lunch:

- `http://kudos.readthedocs.org/`
- `https://github.com/DIKU-EDU/kudos`
- NB! On our VirtualBox, you first need to do this:

  ```
  sudo apt-get install xorriso
  ```

System Calls (1/3)

An operating system mediates the users' access
to underlying physical devices.

User programs issue system calls
to get things done.

System Calls (2/3): What's To a System Call?

A system call is:

- A system call number (i.e., handler has a switch case).
- A handful of register-resident arguments.
- A register-resident result (more outcomes on next slide).

Examples:

```c
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

System Calls (3/3): More Examples

What could possibly happen?

- Opening, reading, writing, or closing a file.
- Mounting a file system.
- Spawning a subprocess.
- Killing another process.
- System halts.
- Anything!

Userland and Kernel Mode

Exception handlers (e.g., for system calls) run in kernel mode.

- Processor enters kernel mode when an exception occurs.
- Careful. Anything can be done in kernel mode.
- The `iret-` family of instructions will *atomically* return the processor to userland and continue with the user program.

C Standard Libraries

Convenient userland wrappers around common system calls.

Examples:

- glibc (GNU C Library, standard on Linux).
- musl libc (aims to be lightweight and secure).
- BSD libc.
- Bionic (standard on Android).
- `userland/libc` (the one in `KUDOS`).

`userland/libc` for `KUDOS`

A *very* conservative subset of a real-world C Standard Library.

- Basic I/O (e.g., `getc`, `putc`)
- Formatted output (e.g., `printf`)
- String handling (e.g., `strlen`, `memcpy`)
- Static heap allocation (e.g., `malloc`, `free`)

OBS! Despite the familiar names, these are not full-fledged.

`man(1)`

- Documentation of local system and utilities.
- "man" is short for "manual".
- man-pages are the goto documentation
- Mainly on Unix-like systems

- `man(1)` uses `less(1)` by default.
- Also available online:

  `https://www.kernel.org/doc/man-pages/`

- Beware that details vary from system to system.

strace(1)

clone(2)

Scheduling

Scheduling Policies

In the context of one processing core,
if more than one task is ready to run,
which should get to go first?

Scheduling Metrics

We need measures for comparing scheduling policies.

Average Turnaround Time

$$\frac{\sum_{i=1}^{n} T^{i}_{\text{turnaround}}}{n}$$

where,

$$T^{i}_{\text{turnaround}} = T^{i}_{\text{completion}} - T^{i}_{\text{arrival}}$$

- This is a *performance* metric, not a *fairness* metric.
  - We can manipulate averages at the cost of fairness.

Average Response Time

$$\frac{\sum_{i=1}^{n} T^{i}_{\text{response}}}{n}$$

where,

$$T^{i}_{\text{response}} = T^{i}_{\text{firstrun}} - T^{i}_{\text{arrival}}$$

- This is *still* not a fairness metric.
  - We can manipulate averages at the cost of fairness.
- This is an *interactive* performance metric.

Summing Up: Points of Measure

Task parameters indpendent of the scheduler:

1. The task arrival time, T_{arrival}.
2. The task execution time, $T_{\text{execution}}$.

The following depends on the scheduler:

3. The time a task is first run, T_{firstrun}.
4. The time a task completes, $T_{\text{completion}}$.

(1) and (2), can play a varying role in (3) and (4), depending on the scheduling policy.

Simplifying Assumptions

Let us begin with some unrealistic assumptions:

1. Each task runs for the same, fixed amount of time.
2. All tasks arrive at the same time.
3. Once started, a task runs to completion.
4. All tasks only use the CPU (i.e., perform no I/O)

Some formal consequences (more along the way):

a. There is a fixed number of n tasks in the system.

First-In, First-Out (FIFO)

- Sometimes also called First-Come, First-Serve (FCFS).
- If processing core is available, run the task.
- Else, queue the task after the last queued task.
- Best if tasks run for an equal amount of time.

Shortest Job First (SJF)

- Arriving tasks are ordered by their execution time.
- The task that takes the least time gets to run first.
- Best if all tasks arrive at the same time.

Preemptive Scheduling

A non-preemptive scheduler runs a task to completion.

A preemptive scheduler interrupts a task
whenever there are better things to do.

Shortest Time to Completion First (STCF)

- A preemptive spinoff of SJF.
- If a job arrives that will complete before the currently running one, switch to that job.
- Starvation?

(Demo left as an exercise.)

Round Robin (RR)

- Run a task for at most a fixed timeslice.
- If it doesn't complete, put it back in the queue.
- Back where?

I/O Scheduling

- I/O operations may again take a varying amount of time.
- Some systems have separate I/O, CPU time schedulers.

Multi-Level Feedback Queues (MLFQ): Rules

- Maintain multiple RR queues with varying time slices.
- So there are multiple “levels” where a task may be.
- Lower levels have shorter timeslices and higher levels.

Multi-Level Feedback Queues (MLFQ): Rules

Rule 1 If Level$(A) <$ Level(B), A runs (B doesn't).

Rule 2 If Level$(A) =$ Level(B), A & B run in RR.

Rule 3 When a job enters the system, it is placed in the lowest queue.

Rule 4 Once a job uses up its time-slice in the RR-scheme (see Rule 2), it is moved one level up, unless the job finished, or it is already on the highest queue.

Rule 5 After a time period of x ticks, move all the jobs in the system to the lowest queue.

(Demo left as an exercise.)