OS 3/24

Reagan Shirk

March 24, 2020

Project 1

- CPU Scheduling
- We should be able to read a text file that looks like:

```
P
p N
<pid> <burst> <priority>
<pid> <burst> <priority>
```

- PID is an integer, burst is how long it happens, priority is, well, priority
- The first P is the number of processes, the second p is the number of unique processes
- N is the number of instructions to follow
- PID can appear once or multiple times
- We are expected to output seven different statistics based on the run, printed to STDOUT

```
<voluntary context switch>
<non-voluntary context switch>
<CPU Utilization>
<throughput>
<turnaround time>
<waiting time>
<response time>
```

- A context switch is said to be non-voluntary when it switches and then comes back
  - Having PID 1, PID 2000, PID 1 means that PID 1 is a non-voluntary switch because it comes back
- Section 5.2, pg.204-205 goes over the statistics
  - They're also discussed in the project description
- First-Come, First-Served Scheduling is on pg. 206-207, section 5.3.1
- Exiting a PID is a voluntary context switch
- We *just* have to compute the stats, we don't have to simulate anything (although simulation might help us later)

Recap from Last Time

Statistics (useful for the project)

- CPU Utilization: how busy is the CPU
- Throughput: work done per the unit (60 instructions/milisecond)
- Turnaround Time: Sum of the time spent waiting on the queue, I/O, etc
- Waiting Time: How long is a process on the ready queue
- Response Time: Time the scheduler takes to start responding to a task

Round Robin Scheduling

- Round Robin "is just a strange term"
- Comes from a butchered french term
- You have tasks and you schedule them such that each task has a set time (a quantum) and it'll run for that quantum, then the scheduler will take the text one
- Pros:
  - Simple, we can design our own
- Large quantum: we're going to have a low number of context switches and a high throughput of each task
  - Context swithces *are expensive*, we need to remove stuff from the registers and put new stuff on and it can take a while
    - Because they're expensive, we don't want a lot of them
- Small quantum: high number of context switches and decreased total throughput of the work which increases response time
- The trick is to tune in your quantum to what is going to be best for the task at hand
  - If you don't have a specific task in mind when using round robin, you're going to have some tradeoffs

New Stuff: Scheduling

Priority Scheduling

- Says I know that my users know best, and as an OS I have some priority to all of my actions, so I will prioritize based on the priority for each task
- The highest priority/lowest value is scheduled first
- Priority is in the process control block
- nice/renice:`nice -20` means that that value will be scheduled the most fequently...? Not entirely sure what he was saying with this
- Cons:
  - starvation, aka indefinite blocking
- Aging was created to prevent starvation
  - While a process is waiting, the priority value is steadily increased so that, if you wait long enough, the process will eventually have a high enough priority to execute
- Several ways to implement: priority queue

Multilevel Queue Scheduling

- Instead of a single priority queue, it'll create a queue for each priority
  - You can prioritize visits to the most important queues, modified round robin algorithm
- Imagine you have a video game, your priority queues might be:
  - graphics processors
  - controller
  - sprite generation
  - And under each of these queues, you could have:
    - real time
    - system processes
    - interactive things
    - batch

Multilevel Feedback Queue Scheduling

- You can separate processes by their CPU Bursts
  - If a process uses too much burst, reschedule them to a lower priority

Thread Scheduling

- In thread scheduling, we have two different levels:
  - kernel
  - user
  - only a system call can move between the two
- Each thread(s) on the user side has a corresponding thread(s) on the kernel side
  - You can have multiple user threads per one kernel tread
  - Multiple threads on both that correspond to each other
  - One user thread that corresponds to one kernel thread
  - One user thread that corresponds to multiple kernel threads
- Contention
  - Process contention (PCS)
  - System contention (SCS)