# OS 3/12

*Reagan Shirk*

*March 12, 2020*

## Pthreads/threads

- You can make each thread do something in one part of a big array (not literally but that's kind of what it's like I think)
  - This is called: data parallelism (splitting up the data between the threads)
- Task parallelism: you put each thread in the same array index and they all do different tasks? I kind of missed the explination

## Makefiles and Flags

```
main.c:
    gcc main c -03 -lpthread //may not compile without this flag
```

- Useful compile flags
- `march`
- `mnative`

## CPU Scheduling

- Four decisions:
  - Process terminates
  - Process from waiting state to the ready state
    * I/O arrives
  - Process switches from running to ready state
    * Timer Interrupt
  - Process switches from running to the waiting state
    * I/O request
  - There is preemptive and nonpreemptive scheduling
    * Nonpreemptive: a process continues to run until it either terminates or is waiting
    * Preemptive: a scheduler can swtich a running process as necessary
  - What is the negative of nonpreemptive scheduling?
    * Some processes may not get to run if one process decides to hog all of the time
    * Fine for one main process, maybe not the best for lots of processes though
  - What is the negative for preemptive scheduling?
    * Less predicatble, more edge cases
    * You could have two processes trying to edit data at the same time, somehow turns into a race where they both want to finish first
    * Need to worry about shared data structures
- Dispatcher: main component involved in all of these decisions
  - Looks at all of the processes and determines which ones will and won't run
  - Tasks are:
    * switching context from one process to another
    * switching from user mode to kernel mode

        * understands how to pause and resume running processes
    - Why do you want your dispatcher to be super fast?
        * Every time you swtich a process, the dispather runs code
            · If the swtich takes 1 second and the dispatcher takes 10 seconds, it's unoptimized and slow
        * Dispatcher speed = dispatcher latency, won't talk much about this but it's important in the real world
    - run `cat /proc/cprocess id > /status` to see the process switches

# Measuring Scheduling Performance

- CPU Utilization: how busy is the CPU?
- Throughput: the amount of work done per time unit (60 instructions per microsecond or something like that, could be seen as operation/second)
- Turnaround time: sum of the time spend waiting on the queue, I/O, etc. Everything to completion. When you schedule a task, how long will it take to complete that task?
- Waiting time: how long is a process on the ready queue?
- Response time: the time the scheduler takes to start responding to a task

# Scheduling Algorithms

- Think of this from the perspective of one CPU, when you have more than one shit gets weird
- FIFO algorithms - make a queue
    - In this situation the term is first come first served scheduling
    - You have three processes:
        * P1 takes 24 seconds
        * P2 takes 3 seconds
        * P3 takes 3 seconds
    - Average waiting times:
        * If p1 executes first: (0 + 24 + 27) / 3 = 17 ms
            · This is because p1 doesn't have to wait to execute if it goes first
            · If p2 follows p1, then p2 has to wait the full 24 seconds before it can execute
            · If p3 is last, it has to wait for p2 and p1 to execute which is 27 seconds
        * If it's p2 $\rightarrow$ p3 $\rightarrow$ p1: (0 + 3 + 6) / 3 = 9 ms
            · This is because p2 doesn't have to wait to execute if it goes first
            · p3 has to wait the 3 seconds for p2 to execute
            · p1 has to wait the 6 seconds to execute, 3 for p3 and 3 for p2
    - Pros to FCFS: simple
    - Cons to FCFS: average waiting time could be high
- Shortest job first scheduling
    - Sort the possible jobs and select the smallest task first
    - Pros: optimal method for single core scheduling
    - Cons:
        * A bit of overhead the the sort
        * We don't know which task is shortest by default
- Round Robin Scheduling
    - Each task split up into a time quantum