CPU Scheduling

CSC 213 – February 8, 2021
Agenda for Today

1. Ngram Solution
2. Lab Q&A
3. Scheduling
4. Wrap Up
Ngram Solution
What exactly are we supposed to do in part E?
   A: In this part, you add support for background commands. When a command ends with an ampersand, start it the same way you start any command, but do not wait for it before accepting additional input from the user.

Is it okay to call getenv("HOME") in our shell to get the user’s home directory?
   A: That’s fine, but make sure getenv doesn’t return NULL. If it does, just print an error.

We’re having trouble getting exit codes from background commands. Any hints?
   A: I recommend calling waitpid in a loop immediately before printing the $ prompt. You’ll call waitpid in a loop with the WNOHANG flag. If there was an exited child, waitpid returns its pid. Otherwise it returns -1.

How do we get the exiting behavior from the example in part E?
   A: To get this output, you need to wait for a specific process ID when you launch a foreground command in that process. You may not have done this yet.
How do we use waitpid to wait for any process?

A: You have to pass in -1 as the pid input. This is a weird quirk described in the manpage.

Do we need special handling to make “cd” with no argument go to the home directory?

A: You don’t have to support “cd” with no arguments, but if you want to, you can use getenv(“HOME”) to get the user’s home directory. Make sure that function doesn’t return NULL.

How can we call waitpid in a loop without blocking the shell?

A: If you call waitpid with options=WNOHANG, it will always return immediately. If there was an exited child process, waitpid returns its pid. Otherwise waitpid returns -1.

Do we need to remember anything about background commands?

A: The examples assume you remember the command name (e.g. “sleep”) but I didn’t mean to create that requirement. I’ll update the lab so you can print either the name or the pid of the background command when it exits.
Is it okay if we check for exited background commands between foreground commands provided on the same input line?

A: Yes, as long as the status prints before the next prompt after the background command has exited, it’s okay.

Can we use a child process to wait for background commands instead of waiting in the parent shell?

A: That’s okay, but you may see messages about exited background commands in the middle of output from some foreground command.
Scheduling
Scheduler Basics

Why do we need a CPU scheduler?

The CPU is an important resource, so using it efficiently is important. We need to make sure each program gets a share of the limited time on the CPU. There are often many more programs running than CPUs, so they have to share.

What characteristics would a good CPU scheduler have?

Efficiency: the scheduler should finish jobs as soon as possible after they arrive. Any work the scheduler has to do should be as quick as possible to maximize programs’ time on the CPU.

Interactive: when users start tasks, we want the scheduler to run those jobs quickly.

The scheduler should produce good schedules, whether we care about response time, turnaround time, or something else.
Scheduling Terminology

Batch Scheduler
   A scheduler that starts jobs and runs them to completion.

Preemption
   When a scheduler interrupts a running job to switch to another job.
   This can happen in response to a variety of events: timers, new jobs, etc.

Quantum
   A short unit of time that jobs can run before being preempted. Not all
   preemptive schedulers use a quantum, but some do.

Context Switch
   When a scheduler switches from one job to another.
Scheduling Algorithms

First In, First Out (FIFO)
Schedule jobs in the order they arrive.

Shortest Job First (SJF)*
Schedule the shortest job first.

Shortest Time to Completion First (STCF)*
Schedule the job that will finish first.

Round Robin
Cycle through jobs executing each for one quantum.
Evaluating Schedulers

How can we measure a scheduler's performance? When is each measurement important?

Turnaround Time: the average time between the moment a job arrives, and when that job is finished. Most important when there is little to no interaction. We just want to finish things quickly.

Response Time: the average time between the moment a job arrives, and when that job is started. More important with user interaction. We don’t want to keep the user waiting for the system to respond.

Fairness: how well is the scheduler dividing time up over all the jobs? Fairness is important for programs that do a lot of input and output where we want all the programs to make some progress. If multiple paying users are sharing a computer, fairness is usually what they paid for.
Preemptive Scheduling

Most systems use preemptive schedulers. Why do you think this is the case?

How do you decide between a long or short scheduler quantum?
Illustrating Schedules

Show schedules for FIFO, SJF, STCF, and RR (quantum = 2s).

A arrives T=0, length=4; B arrives T=1, length=2; C arrives T=3, length=1

---

**FIFO**

- RT: \(0 + 3 + 3 = 6\)
- TT: \(4 + 5 + 4 = 13\)

**SJF (non-preemptive)**

- RT: \(0 + 4 + 1 = 5\)
- TT: \(4 + 6 + 2 = 12\)

**STCF (preemptive)**

- RT: \(0 + 0 + 0 = 0\)
- TT: \(7 + 2 + 1 = 10\)
Illustrating Schedules

Show schedules for FIFO, SJF, STCF, and RR (quantum = 2s).
A arrives T=0, length=4; B arrives T=1, length=2; C arrives T=3, length=1

\[
RT = \frac{0+1+1}{3} = \frac{2}{3} \\
TT = \frac{2+3+2}{3} = 4
\]
Advanced Scheduling *(if there's time)*

What kinds of workloads are particularly bad for these simple schedulers?

How does MLFQ do a better job on these difficult workloads?
Wrap Up
Reminders

**Upcoming Office Hours**
- **Monday** 10am with Quang
- **Tuesday** 11am with Isa, 1pm with Charlie

**Shell Lab**
- Due Tuesday at 11:59pm

**Uniquelist Assignment**
- Due Thursday at 11:59pm