CS162 Discussion #6: Scheduling

Announcements

- Phase 2 started on Tuesday, 10/05
  - Initial design due Friday, 10/15; design reviews Monday-Tuesday (10/18-10/19)
- Midterm 1 on Monday, 10/18 @ 6-9pm in 155 Dwinelle (review session TBA)
  - You get a one-page cheat sheet, handwritten, front and back

Maybe my OS can use TeleBEARS too…

So far, we've talked about how to get threads to cooperate under the assumption that they'll pretty much all get to run at some point. But how does the OS actually decide how to pick which threads get to run at what time? It does this through scheduling. However, as usual, there are tradeoffs between different scheduling policies – depending on what our goals are for the scheduler, we'll want to use different scheduling algorithms. Here are some different criteria we can consider:

- *Response time*. The amount of time it takes to perform an operation (for example, showing a character after a key is pressed, etc.).
- *Throughput*. The number of operations performed per second.
- *Fairness*. A measure of how equally the CPU is shared among users/processes.

We have to make tradeoffs between these criteria; for example, minimizing response time might mean more context switching, but this will increase overhead and consequently lower throughput. Therefore, it's important to decide what scheduling criteria we want to follow and choose a scheduling policy that will achieve them. For each policy, we can compute the following parameters to determine how they differ (these can be computed per process, or as averages across all processes):

- *Waiting time*. The sum of all the chunks of time where the process is waiting on the ready queue, not running.
- *Completion time*. The amount of time it takes from the time the process arrives to the time it finishes, including both waiting and running time.

These are some policies we could use:

- *First Come First Served (FCFS, FIFO)*. Runs each process until completion, in the order they arrive.
- *Round Robin (RR)*. Gives each process a quantum of CPU time. If the process does not finish within that time quantum, it gets preempted and put back onto the end of the queue.
- *Shortest Job First (SJF, STCF)*. Runs the process which requires the least amount of computation first.

- *Shortest Remaining Time First (SRTF, SRTCF)*. Like SJF, but allows for preemption – switches to another process if that process has less time remaining to complete.

Note that SJF and SRTF are more difficult to implement than the others because they require some knowledge about the future to correctly determine which job to run. We can also try to condition our choices based off of previously observed behavior, for example, with an exponential averaging function or multi-level feedback scheduling.

Why you might want a virtual address

As you might remember from the beginning of the semester, running multiple processes means we have to prevent processes from messing with each other's memory. We do this by giving each process a virtual address space, which to the process looks like a chunk of contiguous memory that it has all to itself. While this looks nice and clean from the program's point of view, all of this memory could be broken up into various pieces scattered across physical memory. We use address translation to translate the virtual addresses the program uses into physical addresses that they are associated with on physical memory. This allows each program to refer to memory as if it is the only program running; not only does this provide protection between programs and make things easier for programmers, it also makes programs more modular because physical addresses aren't hardcoded into the program.

One method of memory management is to establish segments contiguous memory that start at a certain base location in physical memory. Virtual addresses in a segment start at 0 and can go up to a given address limit. To find the physical address, we add the base, specified by a segment base pointer, to the virtual address. A program could have multiple segments (i.e., for code, data, stack, etc.), each in a different part of memory; these segments are specified by segment ID, base, and limit, stored as entries in a translation table.

We'll continue onto some other tools for memory management during the coming weeks.

Midterm!

Since the midterm is coming up soon, you might be wondering how you should study for it. Personally, I find that these resources, listed in order from most important to least important, are very useful for doing well on exams in this class:

1. Past midterms
2. Lecture notes
3. Nachos
4. Textbook

Past midterms are extremely important for getting a feel on how your midterm will look, and you might notice that very similar questions appear on many of them across semesters…that's a good sign you'll want to make sure you know how to do them. Also, don't neglect Nachos – it's also fair game for test questions. Good luck!