CS 111: Operating System Principles

Midterm Exam Summer '21

Instructor: Jon Eyolfson

Overview.

“All problems in computer science can be solved by another level of indirection.”

Describe one operating system abstraction of your choice, and what specifically it abstracts.

Interfaces.

You wrote a kernel module in Lab 0, which must have run in kernel mode. You also accessed it in user mode (in your terminal) by using `cat /proc/count`. What is the interface between the two CPU modes? Describe how `cat /proc/count` works using this interface, and be specific about the calls.

(Hint: you do not need to describe how your kernel module works).

Libraries.

Assume you're writing a library you intend for lots of people to use it (and they will, because you know good library principles!). Would you deploy your library as a static or dynamic library? Describe the benefits of your choice over the other.

As your library evolves, describe what you'd have to do to ensure a pain-free experience for your users. Also, describe what would users would have to do to use your new version of the library.

Processes.

Processes virtualize both CPU and memory, and saves state in a process control block (PCB). For both resources please describe what the PCB would need to store to allow context switching, and why. Please use a separate paragraph for each resource.

Basic IPC.

You're running a program in your terminal. You type the command and hit enter. Jon wrote the program you're running, so you're 100% certain it's executing in an infinite loop. You press Ctrl+C, and the process ends, giving you back control of your terminal. Describe how the process was able to exit, even though it was in an infinite loop. Be as specific as possible.

Basic Scheduling.

Between FCFS, SRTF, and RR, which scheduling algorithm(s) do NOT suffer from starvation. Explain how the algorithm(s) avoid this problem.

Advanced Scheduling.

Assume you have a critical user process on your Linux system that must run before any other process, what would you do to ensure it gets scheduled before other processes?

(Hint: you can’t modify your kernel).

Page Replacement.

Using FIFO page replacement has several drawbacks, including suffering from Bélády’s anomaly. Why might you still choose to use FIFO page replacement?

Process API.

Consider the following code:

```
#include <unistd.h>

int main() {
  pid_t pid = fork();
  if (pid > 0) {
    sleep(10);
    // (1)
    sleep(10);
  }
  return 0;
}
```

Assume we execute this program, and it gets assigned PID 100. When PID 100 forks, it creates a child with PID 101.

Question 1.

What did we forget to do in the parent?

(Hint: the answer isn't error checking, assume that I checked for errors. The code is like that to be more readable).

Question 2.

At point (1), what is the state of our child process? Why is it in this state?

Question 3.

After PID 100 exits, what happens to PID 101? Explain.

Page Tables.

Consider a system with 8192 byte pages, a page table entry (PTE) size of 8 bytes, 40-bit virtual addresses, and 56-bit physical addresses.

Question 1.

How many offset bits are there in the virtual address?

Question 2.

How many bits of the virtual address are left over for index bits?

Question 3.

For a single page table (not a multi-level page table), how large would the table have to be (in bytes, KiB, MiB, or GiB)? Explain how you arrived at that size.

Question 4.

Considering the size you calculated in the last question, is it realistic to use a single page table in the real world? Explain why or why not.

Question 5.

How many PTEs could fit on a single page?

Question 6.

How many levels of page tables would you need, if you ensure that each (smaller) page table fits on a page.

Question 7.

Explain how you arrived at your answer for the previous question.