Operating Systems
COMS W4118
Lecture 7

Alexander Roth

2015 – 02 – 12

1 Homework

- Build http server. Has 5 parts.
- Understand the mechanics and understand why they work.

1.1 Part 0 Basic Single-Process/Single-Threaded Web Server

- Same HTTP server from AP
- Only serves one user at a time
- Read through the code and understand everything

2 POSIX Semaphores

- Designed so shared processes don't interfere with each other and avoids a race condition.
- The keyword `synchronized` in Java causes the method to run atomically. No other method running on a thread can occur during this method's call.
- When this method is called, the hidden `lock` on a Java object will be set. On exit, the `lock` will be unset.
- This is a version of mutual exclusion between locking methods.
- A `semaphore` is similar to a lock, except you can lock multiple times.
- Semaphore is initialized with a number, such as 4.
- Different processes can decrement the number.

- `sem_post` means that the process is done and increment the number of the semaphore.
- `sem_wait` means that the process is entering the semaphore and that the number of the semaphore should be decremented.
- A binary semaphore is equivalent to a `mutex` lock. Only one process can hold onto the semaphore now.
- Locking with a semaphore is a completely voluntary process. There is nothing that will necessary stop another process from entering the pool unless it calls `sem_wait`.
- There is no requirement that the process that grabbed the semaphore number has to call `sem_post`. Any process can do that.
- `mutex` forces that there is a mutual exclusions between processes. So if one process locks a part of the program, it must be the process to unlock that part.
- There are two kinds of semaphores:
    1. Named semaphores - Can be used between unrelated processes.
    2. Unnamed semaphores - Semaphore lives on a mapped area of shared memory.
- Typical convention for the name of a semaphore is a slash followed by something relted to the process.
- The named semaphore function `sem_open` returns a pointer to the semaphore.
- Flag value lets the system know if you are creating the semaphore or opening an existing one.
- Semaphores are not deallocated until you call `sem_unlink`.
- Often times, you create a temporary file while the program is running, when you finish the task, you remove the temp file.
- To delete this file, you open it and get a file descriptor of it, then you immediately call link.
- Once a file loses all of its links, then the file will be removed.
- The semaphore API looks like the file API because it is backed up by a file.

3 Signals

- One of the most complicated topics in a UNIX system.
- Concurrency is the main concept behind operating systems.
- Asynchronous calls are the worst part of concurrency.

3.1 Process Groups, Sessions, Controlling Terminal

- Controlling terminal is the terminal program that started the session.
- A single process, the session leader, interacts with the controlling terminal in order to ensure that all programs are terminated when a user "hangs up" the terminal connection.

3.2 Sending Signals

- `kill` command delivers a signal into a process ID.
- `raise` is the same thing as `kill` but it commits suicide.
- `raise` is a simple wrapper on top of the `kill` function.
- `pid` is not part of C; it is UNIX-specific.

4 SIGALRM

4.1 alarm() and pause() functions

- Set up a signal handler for `SIGALRM`, give it a time, and run it.
- Programs will block at `pause()` command.
- This is a good way to implement `sleep`. Create an alarm signal handler and set pause.When the timer runs out, alarm will be called and break out the program.
- You always call `alarm(0)` to cancel an alarm.

4.2 Using setjmp & longjmp to solve the two problems

- Part of the standard C language
- Across function goto statements
- `setjmp` on the first call will return 0, on the next call it will return whatever value you gave to `longjmp`.
- It's not easy to deal with race conditions.