Operating Systems
COMS W4118
Lecture 12

Alexander Roth

2015 – 03 – 03

1 UNIX Domain Sockets

- Reliable across datatype modes.

1.1 File Sharing

- When you for `fork` a process, you `dup` all the file descriptors into the child as well as the parent.
- Sometimes it is useful to pass an open file between processes, but you can't just pass the values to another process with pipes, etc.
- Domain socket is the only facility that allows you to pass an open file descriptor to another process.
- Threads share the same memory, so they are able to access the same file.

2 Single and Multithreaded Processes

2.1 Why Use Threads?

- Express concurrency
- Efficient communication
- Leverage multiple cores (depends)
- Having multithreaded programs allows for easier implementation.
- Multithreading simplifies single threading logic by having separate threads that handle tasks.
- A good example is a multiplayer game server.

- There is no easy way to control scheduling in a single thread.
- The problem with shared memory is that use my setup the region, designate the reason, and tear down the region.

2.2 Multithreading Models

- User threads: thread management done by user-level threads library; kernel knows nothing
- Kernel threads: threads directly supported by the kernel.
- The kernel knows about these threads.
- Kernel level threads are aware of the threads and there is a threaded table that is a link list of `task structs`.
- If you create a thread in `pthread_create`, linux creates a clone of the thread and schedules it accordingly.
- Another way to implement threading is to cheat and not let the kernel know about the thread and do everything in the user level process. Do everything in the user space.
- Kernel threads are just like processes, they are scheduled by the kernel.
- User threads cannot be preemptively scheduled.
- In a program, you must call `pthread_yield()`, which will save the thread in some position in memory and jump to a different routine.

2.3 User Thread Blocking

- One block thread can block all the threads because the kernel just sees one read and does not know that there are other threads in the program.
- We can manage this by using `select()` to intercept every system call beforehand.
- This is a possible solution; however, it's messy and hard to implement.

2.4 Scheduler Activations

- User-level threads are not really used anymore.
- However, the kernel is somewhat aware of the threading.
- When a blocking method is call, the call will go to the kernel
- Since the kernel is away of the threads, it will block all the threads or none at all.

- The kernel will alert the thread scheduler and let's it know it can run other threads in the mean time.
- When the blocking call completes, it will send an interrupt to interrupt the current thread that is active.
- The kernel then alerts the thread scheduler that the blocking call is complete.
- The kernel just knows about the thread scheduler.
- Slightly violates the encapsulation and separation between the kernel and the user space because the kernel needs to know about user threads.