Mark Gameng

Homework 1 – CS 450 Spring 2022

1. OS is a sleeping beauty. It can be woken up by four kinds of events. What are these events? Give an example for each kind of events

   The four kinds of events are booting up, and interrupts which can be faults, system calls and device interrupts. Booting up is the initialization of devices, like turning on a computer. Faults, or also called exceptions, is an unexpected event and is like an error, for example, dividing by zero. System calls is a way for a user program to request a service from the kernel. There are different types of system calls, from process control, file/device management etc. Some common system calls are write, read, fork, exit. Device interrupts are hardware signals from a device, such as a mouse moving.

2. What is a process? More specifically, what does a process contain?

   A process is basically an instance of a program or an executable running in a computer. For example, running the same program simultaneously N times is essentially 1 program and N processes. A process basically contains all the needs for a program in execution. It has its own address space containing the static code and input, as well as the call stack and memory allocation. It also has its own registers and program counter. It has a unique ID called process ID or PID in which you can then use to kill a specific process and many other things.

3. What is a thread? What does a thread contain?

   A thread is similar to a process and can be said to be a lightweight process. Threads within the same process share virtual memory and have to be mutually trusting. So, they contain shared address space (code, input, and data in heap), current working directory, and user and group id. Each thread then has their own thread id, or TID, set of registers with program counter and stack pointer as well as a stack for local variables and return addresses.

4. In 02/03 lecture slide 29, in release(), why not set lock = false when unpark()?

   When unparking, a thread will be woken up and we want to just pass on the lock, basically holding the lock for the next thread. This way, the next thread is responsible for determining whether to set lock to false or not. When unpark and a particular thread wakes up, it doesn't hold the guard instantly and thus, can't set the lock. That's why we don't set lock = false when unpark().

5. In 02/03 lecture slide 31, why does the code fix the race condition described in the previous slide?

   First, the problem is that if a switch happens before park() and the other thread holding the lock happens to release it, then the park() by the first thread would then park and sleep forever. That's why adding setpark() right before l -> guard = false; will fix this problem. This way, a thread can indicate it is about to park and even if it switches to another thread right after, when the other thread calls unpark before park is called, the original park returns immediately instead of sleeping. Looking at the previous slide, when it gets to unpark(), rather

than unparking something that isn’t parked, park() from thread 1 returns immediately so no blocking going on. This solves the race condition of thread 1 possibly sleeping forever and thread 2 unparking even when nothing is parked.