VE482
Introduction to Operating Systems

HOMEORK 4

October 31, 2021

Yihua Liu 518021910998

Ex. 1 — Simple questions

1. Consider a system in which threads are implemented entirely in user space, with the run-time system getting a clock interrupt once a second. Suppose that a clock interrupt occurs while some thread is executing in the run-time system. What problem might occur? Can you suggest a way to solve it?

Race condition might occur and the result is ”indeterminate”. Since threads share data and scheduling is uncontrollable, when multiple executing threads enters the critical region simultaneously, they all try to update shared data structure. Thus, the result is determined by which threads executed at which time.

To solve it, the earliest solution is to control the interrupts (close interrupts before the critical region). Later, Dekker's algorithm and Peterson's algorithm are proposed. Now, Pthread locks should be introduced.

2. Suppose that an operating system does not have anything like the `select` system call (man `select` for more details on the command) to see in advance if it is safe to read from a file, pipe, or device, but it does allow alarm clocks to be set that interrupt blocked system calls. Is it possible to implement a threads package in user space under these conditions? Discuss.

Yes, it is. Since it does allow alarm clocks to be set that interrupt blocked system calls, we can make threads to check the status of a file, a pipe, or device before a system call. If a system call is blocked, then threads can regain the control.

Ex. 2 — Monitors

During the lecture monitors were introduced (3.30). They use condition variables as well as two instructions, `wait` and `signal`. A different approach would be to have only one operation called `waituntil`, which would check the value of a boolean expression and only allow a process to run when it evaluates as True. What would be the drawback of such a solution?

The drawback o such a solution is that `waituntil` has to check the exiting condition every time, so it would waste a lot of resources.

Ex. 3 — Race condition in Bash

Write a Bash script which generates a file composed of one integer per line. The script should read the last number in the file, add one to it, and append the result to the file.

1. Run the script in both background and foreground at the same time. How long does it take before observing a race condition?

   ```
   chmod +x ex3-1.sh
   ./ex3-1.sh & ./ex3-1.sh
   ```

   On WSL, the race condition takes place at number 5 (or 1, 9, 16), and on VMware Ubuntu virtual machine, the race condition takes place at number 2.
   See ex3-1.sh.

2. Modify the script such as to prevent the race condition.

   ```
   chmod +x ex3-2.sh
   ./ex3-2.sh & ./ex3-2.sh
   ```

   See ex3-2.sh.

Ex. 4 — Programming with semaphores

The following C code creates two threads which increment a common global variable. When run it generates a random and inaccurate output. In order to solve this problem we want to use semaphores.

1. On Linux, find the file *semaphore.h*.
2. Read the documentation to understand how to use the functions described in the file *semaphore.h*.
3. Using semaphores adjust the program such as to always return the correct answer.

See ex4.c.