Homework1 Report

Name: Ali Jahanshahi, Mohsen Karimi
Date: January 30, 2018

Question 1

How does a system call execute? Explain the steps from making the call in the userspace process to returning from the call with a result.

Question 2

When does access to data structures in userspace need to be synchronized? A program in user mode invokes a system call using *syscall* function and

passes the system call's arguments and its unique system call number. When this function is called, OS looks up its unique system call entry number in the *system call table*, and copies all the parameters from user space to a specific User Area in kernel. This User Area contains the information about the process and a system stack segment for the process use. It also stores the current context of the process to use it for returning. Then it calls a Trap to switch from user mode to kernel mode and passing system call number and parameters to the kernel. By doing so, kernel knows what system call is invoked. The kernel calculates the actual address of the system call (using the unique number that user provided when invoking the system call) and executes all the system call routines. After execution it passes the return value to the user space, and the execution mode is set to user mode. Then, the user-space process continue execution in user mode.

When multiple processes or threads need to access shared resources, the access to data structures should be synchronized to avoid multiple write access to the shared memory at the same time which may cause race conditions.

Question 3

What synchronization mechanism can be used to access shared kernel data structures safely?

Locks (including Spin lock, mutex, and semaphores), interrupt disabling, and atomic operation are synchronization mechanisms that are supported in Linux kernel.

Question 4

Can spinlocks be used on single-processor platforms? Why or why not?

Spin lock is not a suitable synchronization mechanism to use on single-processor platforms. Depending on the scheduling algorithm it may cause waste of CPU resources (e.g. in time sliced scheduling which is preemptive scheduling) or even Deadlocks (e.g. earliest deadline First scheduling which is non-preemptive).