FIT1047 Tutorial 6 – Sample Solution

Topics

- Operating Systems
- Processes

Instructions

The tasks are supposed to be done in groups of two or three students.

For some of the questions, you may have to refer to the free online textbook at `http://www.ostep.org/`, or do some internet research.

Task 1: Processes

1. What is the difference between a *process* and a *program*?

   A program is the code stored on disk or in memory. A process is an instance of that program while it is running (a program "in execution"). The same program can be running as several processes at the same time – on different computers, or even on the same computer.

2. A *context switch* happens when execution switches between a process and the operating system (e.g. to handle an interrupt). What does a context switch have to do?

   It has to save the current process state so that it can be restored when control is passed back to the process. The process state consists of the contents of all registers that can be accessed through the code (this includes the PC, but doesn't include special-purpose internal registers such as MBR or MAR in MARIE). A context switch also sets the CPU mode, switching from user mode to kernel mode or back (depending on the direction of the switch).

3. Explain the different *states* that a process can be in.

   A process can be in one of three states: *ready*, *running*, and *blocked*. In the *ready* state, a process ready to run, but it's not currently running on the CPU. After loading code from a file into memory and setting up all the necessary bookkeeping in the OS, the *ready* state is the first state a process is placed in.

   A process in the *running* state is currently being executed on the CPU. A process goes from *ready* to *running* when the operating system *schedules* the process and switches to it. There can only be one process in the running state at any point in time (or one per CPU in a multi-CPU system).

A process is in the *blocked* state when it is currently waiting for a system call to return. E.g. if a process requests some data to be read from a file, it makes the corresponding system call, and since file I/O is slow, the OS may decide to schedule other *ready* processes in the meantime. As soon as the data has been read successfully, the process is moved from *blocked* to the *ready* state, and at some point will be scheduled to become *running* again.

Task 2: Protection

An important concept in Operating Systems is *Protection*, meaning that one malicious or buggy process cannot bring another process or even the entire system down.

1. Explain the difference between *user mode* and *kernel mode*

   In kernel mode, code is completely unrestricted, it can contain any instruction, and it has full access to the hardware. In user mode, code cannot access the hardware directly, because the CPU will refuse to execute the corresponding instructions. A lot of operating system code needs to run in kernel mode. Application programs run in user mode.

2. Why are *system calls* necessary when application programs run in user mode?

   Since applications cannot execute CPU instructions to access the hardware, they need some other mechanism to e.g. open a file or read from the keyboard. A system call is a call from an application program (in user mode) into the operating system (running in kernel mode).

3. Find a list of system calls in the Linux operating system. How many are there? Which system call numbers would you have to use to get the current time, open a file, create a new directory, and exit the process?

   A list can be found e.g. at `http://syscalls.kernelgrok.com`.

System call	*Description*	*number*
time	get current time	13
open	open a file	5
mkdir	create a directory	39
exit	exit the process	1

Task 3: Examining running processes

Explore the processes currently running on your computer. On Windows, start the *Task Manager*. On Mac OS, start the *Activity Monitor*. On many Linux systems, you can run a program called *System Monitor*, or run the *top* command line tool.

1. How many processes are currently running?

2. How many processors (or cores) does your computer have?
3. Find out how much of the processor(s) is currently in use, and by which process. Try a few things (e.g. playing a video in a web browser) to see how CPU usage goes up and down.
4. Can you find out how much time is spent in user mode vs. kernel mode?