Drew Smith
COP4610
February 12, 2016
Assignment 1

## Processes (25 pts)

1. **(Exercise 3.8) Describe differences among short-term, medium-term, and long-term scheduling.**

Answer:

The short-term scheduler manages the transition of processes from RAM to the CPU as it is invoked very frequently (milliseconds). So a short-term scheduler selects a process from those that are in memory and ready to execute and allocates the CPU to it. The medium-term scheduler manages the transition of processes from RAM to swap space (usually on hard disk). So we can say that the medium-term scheduler takes processes that are currently in memory or from the ready or blocked queue and removes them from memory, then reinstates them later to continue running. This helps improve process mix and the act of freeing of memory. The long-term scheduler is invoked very infrequently, as it manages the start of new processes. The long-term scheduler determines which jobs are brought into the system for processing. In conclusion, the primary difference is the frequency of their execution. The short-term scheduler must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

2. **(Exercise 3.11) Explain the role of the init process on UNIX and Linux systems in regard to process termination.**

Answer:

The init process is the parent of all processes. Its primary role is to create processes from a script stored in the file /etc/inittab. The process init is the first process started during booting of the computer system. When a process is terminated, it briefly moves to the zombie state and remains in that state until the parent invokes a call to the wait() function. When this occurs, the process id, as well as entry in the process table are both released. However, if a parent does not invoke wait(), the child process remains a zombie as long as the parent remains alive. Once the parent process terminates, the init process becomes the new parent of the zombie. Periodically, the init process calls the wait() function, which ultimately releases the pid and entry in the process table of the zombie process.

Drew Smith
COP4610
February 12, 2016
Assignment 1

3. **(Exercise 3.12) Including the initial parent process, how many processes are created by the program shown in Figure 3.32.**

Answer:

16 processes are created.

4. **(Exercise 3.14) Using the program in Figure 3.34, identify the values of pid at lines A, B, C, and D.**

Answer:

A = 0, B = 2603, C = 2603, D = 2600

5. **(Exercise 3.17) Using the program shown in Figure 3.35, explain what the output will be at Line X and Y.**

Answer:

Since the child is a copy of the parent, any alterations the child makes will occur in its copy of the data and won't be reflected in the parent. As a result, the values output by the child at line X are 0, -1, -4, -9, -16. The values output by the parent at line Y are 0, 1, 2, 3, 4

## Threads (25 pts)

1. **(Exercise 4.6) Provide two programming examples in which multithreading does *not* provide better performance than a single-threaded solution.**

Answer:

Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return. (2) Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

2. **(Exercise 4.8) Which of the following components of program state are shared across threads in a multithreaded process?**

Answer:

Threads share the heap, global memory and the page table. They have private register values and private stack segments.

3. **(Exercise 4.15) Consider the following code segment...**

    **a. How many unique processes are created?**

    **b. How many unique threads are created?**

Answer:

6 processes and 2 threads.

4. **(Exercise 4.17) The program shown in Figure 4.16 uses the Pthread API. What would be the output from the program at LINE C and LINE P?**

Answer:

Output at LINE C is 5. Output at LINE P is 0.

5. **(Exercise 4.18) Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processing cores in the system. Discuss the performance implications of the following scenarios.**

    **a. The number of kernel threads allocated to the program is less than the number of processors.**

    **b. The number of kernel threads allocated to the program is equal to the number of processors.**

    **c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user level threads.**

Answer:
When the number of kernel threads allocated to the program is less than the number of processors, then some of the processors would remain idle since the scheduler maps only

kernel threads to processors and not user-level threads to processors. When the number of kernel threads allocated to the program is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when kernel-thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle. When there are more kernel threads allocated to the program than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.