Tutorial 3
Threads & Concurrency and Synchronization tools & examples – Parts I & II
Operating Systems Comp Sci 3SH3 Term 2, Winter 2022

Prof. Neerja Mhaskar

Tutorials are not mandatory. They are simply a tool for you to understand the course material better.
Tutorial Format: The questions will be posted a day before or on the day of the tutorial on the course website. You can choose to solve these problems before hand and come in with your solutions. If you have all of the questions correct you can choose to leave. If you have any of them incorrect, it is recommended that you stay and understand the solution.

Questions:

1. For the program below:

```
int main() {
      Pid_t pid1, pid2;
      Pid1 = fork();
      pid2 = fork();
      if (pid1 == 0) { /* child process */
      pthread_create(. . .);
      }
      if (pid2 == 0) { /* child process */
      pthread_create(. . .);
      }
}
```

a. How many unique processes are created?
b. How many unique threads are created?
c. Draw the process and thread tree for this code.

2. The program shown in Figure 4.16 (page 194 of the textbook) uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

 pid = fork();

 if (pid == 0) { /* child process */
   pthread_attr_init(&attr);
   pthread create(&tid,&attr,runner,NULL);
   pthread_join(tid,NULL);
   printf("CHILD: value = %d",value); /* LINE C */
 }
 else if (pid gt; 0) { /* parent process */
   wait(NULL);
   printf("PARENT: value = %d",value); /* LINE P */
 }

 void *runner(void *param) {
    value = 5;
    pthread_exit(0);
 }
```

3. Consider a multicore 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 processing cores.
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

4. A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

a) How many threads will you create to perform the input and output? Explain.
b) How many threads will you create for the CPU-intensive portion of the application? Explain.

5. Consider a banking system that maintains an account balance with two functions: `deposit(amount)` and `withdraw(amount)`. These two functions are passed the amount that is to be deposited or withdrawn from the bank account. Assume that a husband and wife share a bank account. Concurrently, the husband calls the `withdraw()` function and the wife calls `deposit()`. Describe with an example how a race condition is possible and what might be done to prevent the race condition from occurring.

6. Prove that the following critical solution problem fails. Also state explicitly state which of the critical section problem solution requirements are not satisfied and why.

Note: We assume that load and store machine language instructions are atomic.

The two processes share the following variable:
`int turn;`

1. **`turn`** indicates whose turn it is to enter the critical section.
2. You can initialize `turn = 0` or `turn = 1.` The failure of the solution is independent of the initialization.

Algorithm for process P$_i$

```
do {
    while (turn==j);
    critical section
    turn = j;
    remainder section
} while (true);
```

7. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism—a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available:

1. The lock is to be held for a short duration.
2. The lock is to be held for a long duration.
3. A thread may be put to sleep while holding the lock.

8. Illustrate how you would use semaphores to synchronize processes P1, P2, P3, P4 where P1 is executed before P2, P2 is executed before P3, P3 is executed before P4.