CSCI 4210 — Operating Systems
Lecture Exercise 3 (document version 1.0)
Shared Memory and Multi-Threaded Programming

- This lecture exercise is due by 11:59PM ET on Friday, July 23, 2021
- This lecture exercise consists of practice problems and problems to be handed in for a grade; graded problems are to be done individually, so **do not share your work on graded problems with anyone else**
- For all lecture exercise problems, take the time to work through the corresponding video lecture(s) to practice, learn, and master the material; while the problems posed here are usually not exceedingly difficult, they are important to understand before attempting to solve the more extensive homeworks in this course
- You **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with no warning messages when the `-Wall` (i.e., warn all) compiler option is used; we will also use `-Werror`, which will treat all warnings as critical errors
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v18.04.5 LTS and `gcc` version 7.5.0 (`Ubuntu 7.5.0-3ubuntu1~18.04`)

Practice problems

Work through the practice problems below, but do not submit solutions to these problems. Feel free to post questions, comments, and answers in our Discussion Forum.

1. Why are there two separate numeric IDs for a shared memory segment, i.e., a shared memory key and a shared memory ID?
2. Given the diagram of processes `P1` and `P2` shown in the notes for July 12, when both processes reach `<point A>` and `<point B>`, assume that shared variable `x` has an "uncorrupted" value of 10. What are all of the possible values of local variables `y` and `z`? Clearly show all possibilities.
3. In `count-shm.c`, what would happen if the parent process did not call `waitpid()` and removed the shared memory segment (via `shmctl()`) before the child process completed its work?
4. What are the differences between the `waitpid()` system call and the `pthread_join()` library function?
5. Review the `pthread-lexec3.c` code posted along with this lecture exercise (also shown on the next page). Assuming no errors occur, determine exactly how many distinct possible outputs there could be. Show all possible outputs. Also, how many bytes are allocated on the runtime heap after all threads are joined back in to the main thread?

```
/* pthread-lecex3.c */

/*
 * Lecture Exercise 3 -- Practice Problem 5
 *
 * How many distinct possible outputs are there?
 *
 * Show all possible outputs.
 *
 * How many bytes are dynamically allocated on the heap?
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>

void * pthread_function( void * y )
{
  int * x = (int *)y;
  char * s = calloc( 16, sizeof( char ) );
  s[0] = ' ';
  strcat( s, "CHARMS" );
  for ( int i = 0 ; i < 8 ; i++ )
  {
    int tmp = *(x + i);
    tmp += i;
    *(x + i) = tmp;
  }
  fprintf( stderr, "%s\n", s );
  return NULL;
}

int main()
{
  int * z = calloc( 8, sizeof( int ) );
  pthread_t t1, t2;
  pthread_create( &t1, NULL, pthread_function, z );
  pthread_create( &t2, NULL, pthread_function, z );
  fprintf( stderr, "LUCKY" );
  pthread_join( t2, NULL );
  pthread_join( t1, NULL );
  for ( int i = 0 ; i < 8 ; i++ ) printf( "%d%s", *(z + i), i == 7 ? "" : "-" );
  printf( "\n" );
  return EXIT_SUCCESS;
}
```

Graded problems

Complete the two problems below and submit via Submitty for a grade. Please do not post any answers to these questions. All work on these problems is to be your own.

1. Review the `lecex3-q1-main.c` code posted along with this lecture exercise. Do **not** change this code or submit this code to Submitty. Submitty will compile your own code file in with this given `lecex3-q1-main.c` code, plus a hidden source file that contains the implementation of the `lecex3-q1-parent()` function.

   Your task is to write the `lecex3-q1-child()` function in your own `lecex3-q1.c` code file. In this `lecex3-q1-child()` function, you must attach to the shared memory segment created by the parent process, convert all of the character data in the shared memory segment to `ALL-CAPS`, then detach from the shared memory segment and exit the child process.

   To convey the shared memory key and the size of the shared memory segment to your child process, the parent process will write these two four-byte values to a pipe. The first `int` value is the shared memory key; the second `int` value is the size of the shared memory segment.

   The parent process will display the shared memory after your child process terminates. Your child process should therefore produce no output to `stdout`.

   As an example, assume the shared memory segment contains the following data:

   ```
   abcdefghijklmnopq rstUVWXYz12 34567890yAy!
   ```

   Program execution would produce the following output in the parent process:

   ```
   bash$ ./a.out
   ABCDEFGHIJKLMNOPQ RSTUVWXYZ12 34567890YAY!
   ```

   If any errors occur in the child process, display an error message to `stderr` and return `EXIT_FAILURE`; otherwise, return `EXIT_SUCCESS`.

   Write all of your code in `lecex3-q1.c` for this problem.

2. Review the `lecex3-q2-copy-file.c` code posted along with this lecture exercise. Do **not** change this code or submit this code to Submitty. Submitty will compile your own code file in with this given `lecex3-q2-copy-file.c` code.

   The given code defines a `copy_file()` function that you will use for your thread code. The argument given to `copy_file()` is simply the name of the file to copy to a backup file. Trace through the given function to see that it will attempt to create a backup file with filename prefix `"backup_"` (e.g., file `"stuff.txt"` would be copied to a `"backup_stuff.txt"` file).

   Your task is to write the `main()` function in your own `lecex3-q2.c` code file. In your `main()` function, you will need to create a child thread for each file given as a command-line argument. Once all threads are created, you then need to call `pthread_join()` on each child thread to obtain the number of bytes copied by each thread.

   Parallelize these threads to the extent possible, but since `pthread_join()` is a blocking call, you can simply join the threads in the same order that you create them.

   Only submit your `lecex3-q2.c` code file for this problem.

   Sample output from `main()` and the given `copy_file()` function is shown below. Note that interleaving may occur for some of the given output when multiple files are given; however, the last line must always be the `"Successfully copied"` line.

```
bash$ ./a.out
MAIN: Successfully copied 0 bytes via 0 child threads

bash$ ./a.out stuff.txt
MAIN: Creating thread to copy "stuff.txt"
CHILD THREAD: Copying "stuff.txt"...
MAIN: Thread completed copying 6294 bytes for "stuff.txt"
MAIN: Successfully copied 6294 bytes via 1 child thread

bash$ ./a.out file1.txt file2.txt file3.txt
MAIN: Creating thread to copy "file1.txt"
MAIN: Creating thread to copy "file2.txt"
CHILD THREAD: Copying "file1.txt"...
MAIN: Creating thread to copy "file3.txt"
CHILD THREAD: Copying "file2.txt"...
CHILD THREAD: Copying "file3.txt"...
MAIN: Thread completed copying 4290 bytes for "file2.txt"
MAIN: Thread completed copying 14903 bytes for "file1.txt"
MAIN: Thread completed copying 34096 bytes for "file3.txt"
MAIN: Successfully copied 53289 bytes via 3 child threads
```