CSCI 4210 — Operating Systems
Exam 2 Prep and Sample Questions (document version 1.0)

- Exam 2 is scheduled for the extended window that starts at 6:55PM ET on Wednesday 4/14 and ends at 2:00AM ET on Thursday 4/15
- Exam 2 will be a 90-minute exam, but you will have a full **three-hour window of time** to complete and submit your exam solutions
- Submitty will start the three-hour "clock" for you when you first download the exam, so please plan accordingly by avoiding any distractions or interruptions; and note that **you must start your exam by 11:00PM ET on Wednesday 4/14 to have the full three-hour window**
- There will be a mix of auto-graded and free response questions on the exam; for auto-graded questions, you will submit your code in the same manner as the homeworks; for free response questions, please place all of your answers in **a single PDF file called `upload.pdf`**
- Exam 2 is open book(s), open notes; given that you are working remotely, you may use any and all of the posted course materials, including all previous questions and answers posted in the Discussion Forum
- Make-up exams are given only with an official excused absence (`http://bit.ly/rpiabsence`); also re-read the syllabus
- Exam 2 covers everything through live lecture Thursday 4/8, including all assignments due through Friday 4/9; while Exam 2 is cumulative, approximately 75% of the exam questions will focus on topics covered since Exam 1, i.e., shared memory, multi-threaded programming, Java threads, threads in C using the Pthread library, mutual exclusion, synchronization, and semaphores
- To prepare for the exam, focus on the coding examples and suggested "to do" items, as well as other code modifications you can think of; review `man` pages and behavior for all system calls and library functions we have covered
- For free response questions, be as concise as you can in your answers; long answers are difficult to grade
- Pretend that you are taking this exam in West Hall Auditorium; therefore, all work on the exam **must** be your own; **do not even consider copying from other sources or communicating with others**
- **Any copying or collaborating with others will result in a grade of zero on the exam;** this includes posting in the Discussion Forum or other public or private forums

Sample problems

Use the previous sample questions provided to prepare for Exam 1. In addition, practice problems for Exam 2 are on the pages that follow. Feel free to post your solutions in the Discussion Forum; and reply to posts if you agree or disagree with the proposed approaches/solutions. Some selected solutions will be posted by Sunday 4/11.

1. What do the following system calls do? What are their input arguments and return values?
   - `shmget()`
   - `shmat()`
   - `shmdt()`
   - `shmctl()`

2. How can two processes share data via a shared memory segment, i.e., what are the series of system calls required? Since `shmat()` returns a generic pointer (i.e., `void *`), how can we be sure both processes interpret the shared data correctly?

3. Without any synchronization between the two processes from Question 2 above, what could go wrong?

4. How is a shared memory segment deleted? And if a shared memory segment is deleted while other processes are attached to it, what happens?

5. Write a C program to create a shared memory segment with a known key and a size of 1024 bytes. Write a separate C program to attach to the shared memory segment and write the Fibonacci sequence starting with $F(0) = 0$ and $F(1) = 1$. Use an `unsigned long` data type for each value and either fill up the 1024 bytes or stop when overflow occurs.

   Finally, write a third C program to attach to the shared memory segment and simply display the Fibonacci sequence from the shared memory. As an optional command-line argument, if `-DELETE` is specified (as shown below), the shared memory segment is also deleted by this third program.

   ```
   bash$ ./a.out -DELETE
   ```

6. Write a C program to create a shared memory segment with a known key and a size of 128 bytes. Next, have this program prompt the user to repeatedly enter a line of text. Each line is written to the shared memory segment, truncating the line to fit, if necessary.

   Write a second C program to attach to the shared memory segment and wait for data to be added. When data is written by the first process, this second process downcases all data and displays it.

   Note that the first process must not display the prompt until the second process has downcased and displayed the data. Further, the two processes must be completely separate, i.e., do not use `fork()` to solve this problem.

   As a hint, use the first few bytes of the shared memory segment to relay synchronization information from one process to the other. Create a synchronization protocol to ensure there are no race conditions in your solution. (Do not use semaphores to solve this problem.)

7. What do the following library functions do? What are their input arguments and return values? And how do they get compiled in to your code?

    - `pthread_create()`
    - `pthread_detach()`
    - `pthread_self()`
    - `pthread_join()`
    - `pthread_exit()`

8. In a multi-threaded program, describe at least two types of synchronization errors that can occur. How best can these situations be avoided?

9. What is a critical section? Why is it important in multi-threaded programming? Are there critical sections in multi-process programming, i.e., in support of inter-process communication (IPC)?

10. For semaphores, what are the definitions of the `P()` and `V()` operations? Further, how does a binary semaphore differ from a counting semaphore? How are each used to avoid synchronization problems?

11. Using `P()` and `V()` operations, write a pseudocode solution to the Readers/Writers problem for a shared reservation system that governs an airplane with 20 rows of seats, each row consisting of four seats. More specifically, any number of readers can simultaneously read the seating chart, including which seats are available. And while there can be any number of writers, your solution must guarantee that each seat can only be reserved by one person.

12. Using `P()` and `V()` operations, write a solution to the Dining Philosophers problem.

13. What conditions are required for deadlock to occur? How can you avoid deadlock? Describe at least two techniques for recovering from deadlock.

14. Given the following C program, what is the **exact** terminal output? Assume all system calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
  close( 0 );
  close( 1 );

  char * s = "ARE YOU READY FOR THE EXAM?";

  int p[2];
  pipe( p );

  pid_t pid = fork();

  if ( pid == 0 )
  {
    write( p[1], s, 20 );
    printf( "%s!\n", s );
  }
  else /* pid > 0 */
  {
    waitpid( -1, NULL, 0 );
    char buffer[1024];
    int rc = read( p[0], buffer, 1024 );
    printf( "(read %d bytes)\n", rc );
    buffer[14] = buffer[18] = buffer[rc] = '\0';
    fprintf( stderr, "%s%sNOT!\n", buffer + 8, buffer + 15 );
    fprintf( stderr, "%s\nHOPEFULLY.\n", buffer + 28 );
  }

  return EXIT_SUCCESS;
}
```

From the shell, how would you redirect terminal output to go to an output file?

How would the output change if the `waitpid()` call was removed?

How would the output change if the two `close()` calls were removed?

15. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SHARED_KEY 8000

int main( int argc, char * argv[] )
{
  int shmid = shmget( SHM_SHARED_KEY, sizeof( int * ), IPC_CREAT | 0660 );
  int * data = shmat( shmid, NULL, 0 );

  int pid = fork();

  if ( pid > 0 ) waitpid( pid, NULL, 0 );

  int i, stop = 6;

  for ( i = 1 ; i <= stop ; i++ )
  {
    data[i%2] += i;
  }

  printf( "%s: data[%d] is %d\n", pid > 0 ? "PARENT" : "CHILD", 0, data[0] );
  printf( "%s: data[%d] is %d\n", pid > 0 ? "PARENT" : "CHILD", 1, data[1] );

  shmdt( data );

  return EXIT_SUCCESS;
}
```

Add code to delete the shared memory segment such that only one process actually deletes the segment. For this, do not assume that system calls will always return successfully.

What is the **exact** terminal output if the `waitpid()` call is removed? Clearly show all output possibilities.

16. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Also assume that the main thread ID is 256, while child thread IDs are assigned sequentially starting at 512.

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

void * go( void * arg )
{
  char * s = calloc( strlen( arg ) + 1, sizeof( char ) );
  strcpy( s, arg );
  s[6] = '\0';
  fprintf( stderr, "%s", s );
  return NULL;
}

int main()
{
  int a = 1;
  pthread_t tid;
  char * q = "READY FOR THE EXAM";
  pthread_create( &tid, NULL, go, q );
  if ( a == 1 ) pthread_join( tid, NULL );
  fprintf( stderr, "%s", q );
  if ( a == 2 ) pthread_join( tid, NULL );
  fprintf( stderr, "!\n" );
  return EXIT_SUCCESS;
}
```

How does the output change if variable a is initialized to 2?

How does the output change if variable a is initialized to 3?

17. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Also assume that the main thread ID is 256, while child thread IDs are assigned sequentially starting at 512.

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

void * charms( void * arg )
{
  int * s = (int *)arg;
  printf( "%ld lucky %d\n", pthread_self(), *s );
  return NULL;
}

int main()
{
  pthread_t tid1, tid2;
  int rc = -1, x = 7;
  rc = pthread_create( &tid1, NULL, charms, &x );
  rc = pthread_create( &tid2, NULL, charms, &x );
  x = 13;
  rc = pthread_join( tid1, NULL );
  printf( "%d unlucky %d\n", rc, x );
  rc = pthread_join( tid2, NULL );
  return EXIT_SUCCESS;
}
```

18. Given the following C program, what is the **exact** terminal output? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Also assume that the main thread ID is `256`, while child thread IDs are assigned sequentially starting at `512`.

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

void * qwerty( void * arg )
{
  int * q = (int *)arg;
  *q += 6;
  printf( "%ld lucky %d\n", pthread_self(), *q );
  return NULL;
}

int main()
{
  pthread_t tid1, tid2;
  int x = 7;
  int rc = pthread_create( &tid1, NULL, qwerty, &x );
  rc = pthread_create( &tid2, NULL, qwerty, &x );
  x = 13;
  rc = pthread_join( tid1, NULL );
  printf( "%d unlucky %d\n", rc, x );
  rc = pthread_join( tid2, NULL );
  return EXIT_SUCCESS;
}
```

Identify critical section(s) in the code above and use at most one `mutex` variable to implement a fully synchronized solution that avoids corruption with variable `x`. More specifically, add to the code above without changing it or deleting any of the existing code; use arrows to show where the additional code would be added.

Next, write a C program that makes use of a shared memory segment to implement the same functionality and behavior of the above code. In other words, convert the multi-threaded code above into a multi-process version by using `fork()` and `waitpid()` instead of `pthread_create()` and `pthread_join()`.

19. Given the following C program, what is the **exact** terminal output? Further, what is the **exact** contents of the `E.txt` file? If multiple outputs are possible, succinctly describe all possibilities. Assume that all system calls complete successfully. Also assume that the main thread ID is `256`, while child thread IDs are assigned sequentially starting at `512`.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define SONNY 0

void * wtf( void * arg )
{
  int * f = (int *)arg;
  printf( "%ldA%d\n", pthread_self(), *f );
  fprintf( stderr, "%ldB\n", pthread_self() );
  return NULL;
}

int main()
{
  close( SONNY );
  printf( "%ldC\n", pthread_self() );
  fprintf( stderr, "%ldD\n", pthread_self() );

  int fd = open( "E.txt", O_WRONLY | O_CREAT | O_TRUNC, 0660 );
  printf( "%ldF\n", pthread_self() );
  fprintf( stderr, "%ldG%d\n", pthread_self(), fd );

  pthread_t tid1, tid2;
  int rc = pthread_create( &tid1, NULL, wtf, &fd );
  rc = pthread_create( &tid2, NULL, wtf, &rc );

  pthread_join( tid1, NULL );
  pthread_join( tid2, NULL );
  printf( "%ldH\n", pthread_self() );
  fprintf( stderr, "%ldI\n", pthread_self() );

  fflush( NULL );
  close( fd );
  return EXIT_SUCCESS;
}
```

How do the output and file contents change if `SONNY` is defined as `1` instead of `0`?