CSCI 4210 — Operating Systems †
Spring 2020 Exam 2 (April 13, 2020)

- You have 105 minutes to complete this exam (i.e., 2:00-3:45PM); note that 50% extra time is 160 minutes and 100% extra time is 210 minutes.
- Submitty will open for submissions at 2:00 and will remain open until 4:45. I am leaving extra time to account for issues with downloading, scanning, uploading, etc. Please monitor your own time. This is on the honor system.
- When you submit your exam, you can upload a text file with your answers, or a pdf formatted scan of your written answers on the answer sheet. Please don't upload an encrypted file or a docx file.
- **Accomodations group:** Exam 2 is targeted for 105 minutes; therefore, if you have 50% additional time, you will have 160 minutes; if you have 100% additional time, you will have 210 minutes. If you take all of your time, Submitty may mark you as late. **Ignore this and submit anyway.** I have your names and I will take care of it. (Please contact me if you will be **substantially** late.)
- I will hang out in the chatroom (https://webchat.freenode.net/) channel #rpi-csci-4210 and in my WebEx (https://rensselaer.webex.com/meet/turnew2) from no later than 2:00 until at least 5:30. Contact me if there are any questions.
- **If you have problems submitting on time please submit as soon as possible even if it is late.** Submitty will still accept your exam. Then contact me via my WebEx or via email.
- The exam is open book and open notes. You can run anything you have available on your computer including class examples and code from the test. Please watch your time! Please do not use web resources (Google, Stack Overflow, etc.) and do not ask other people for help! Class Materials that I posted to Submitty are allowed.
- There are no syntax errors anywhere on this exam. Note that `#include` directives are omitted to save space on the page.
- For all C programs, assume a 64-bit architecture and that system calls all return successfully.
- Questions will not be answered except when there is a glaring mistake or ambiguity in a question. Please do your best to interpret and answer each question.
- The above notwithstanding, I will use post any required clarifications to the **Submitty discussion forum**. If possible, please keep the Discussion Forum open and refer to it occasionally. Please do not post there for the duration of the Exam.
- Below is an honor code pledge for this course. By submitting this exam for grading, you are asserting that you agree with and will abide by this pledge.

Honor Pledge: On my honor, I have neither given nor received any unauthorized aid on this exam.

1. **(6 POINTS)** When you call `pthread_join()`, what happens? Indicate the best answer.

   (a) The thread terminates and returns a specified value
   (b) The child thread that calls `pthread_join()` is acknowledged as being terminated
   (c) The thread disconnects from its parent thread
   (d) The thread detaches by terminating and returning `NULL`
   (e) A child thread is joined back into the corresponding parent thread
   (f) The process exits via the `exit()` system call

2. **(6 POINTS)** Consider a `mutex` declared as type `p_thread_mutex_t()` and shared memory being used by multiple `fork()` processes. Whichof the following are true? Indicate the best answer.

   (a) `mutex` cannot be used with shared memory
   (b) `mutex` can be used with shared memory with the correct attributes set
   (c) `mutex` is not needed because shared memory automatically enforces mutual exclusion
   (d) Items a & c
   (e) Items b & c
   (f) None of the above

3. **(6 POINTS)** The code below has memory leaks. How many **total** bytes are leaked when you run it. Indicate the best answer.

```
void * q3( void * y )
{
  int * x = (void *)y;
  char * s = calloc( *x + 16, sizeof( char ) );
  sprintf( s, "ABCDEFGH" );
  fprintf( stderr, "%s", s );
  return NULL;
}
int main()
{
  int * z = calloc( 1, sizeof( int ) );
  if (fork() == 0) *z = 10;
  pthread_t t1;
  pthread_create( &t1, NULL, q3, z );
  fprintf( stderr, "ERROR" );
  pthread_join( t1, NULL );
  return EXIT_SUCCESS;
}
```

(a) `50`
(b) `32`
(c) `4`
(d) `46`
(e) `20`
(f) `16`

4. **(6 POINTS)** When you call `shmget()`, what happens? Indicate the best answer.

(a) The process is attached to the given shared memory ID, if valid
(b) Given a shared memory key, the shared memory ID is determined (and returned)
(c) A shared memory segment is removed and its data is destroyed.
(d) The process or thread detaches from the given shared memory ID
(e) Given a shared memory ID, the size of the shared memory segment is returned
(f) None of the above

5. **(6 POINTS)** Assume that initially there is no shared memory segment with key `4000`. If you run the given code below three times, what is the terminal output of the **last** program execution and how many times is the shared memory segment removed? Indicate the best answer.

```
int main( int argc, char * argv[] )
{
  int shmkey = 4000;
  int shmid = shmget( shmkey, sizeof( int ), IPC_CREAT | 0660 );
  int * data = shmat( shmid, NULL, 0 );
  int flush=0;
  pid_t pid = fork();
  if ( pid > 0 ) waitpid( pid, NULL, 0 );
  int i, stop = 4;
  for ( i = 1 ; i < stop ; i++ ) *data += i;
  if ( pid == 0 ) printf( "Sum is %d\n", *data );
  if (*data > 31) flush = 1;
  shmdt( data );
  if (flush) shmctl( shmid, IPC_RMID, 0 );
  return EXIT_SUCCESS;
}
```

(a) `"Sum is 6"` shared memory removed 2 times.
(b) `"Sum is 18"` shared memory removed 0 times.
(c) `"Sum is 30"` shared memory removed 1 times.
(d) `"Sum is 42"` shared memory removed 0 times.
(e) `"Sum is 6"` shared memory removed 1 times.
(f) `"Sum is 18"` shared memory removed 1 times.

6. **(6 POINTS)** For a UDP socket, what does a return value of `0` indicate in the `recvfrom()` system call? Indicate the best answer.

(a) No space is available to store the received data (i.e., we avoid buffer overflow)
(b) An error occurred
(c) No data (i.e., zero bytes) were received from the remote end
(d) The remote end closed its socket
(e) For UDP-based communication, `recvfrom()` will not return `0`
(f) None of the above

For Questions 7 and 8, use the code shown below.

```
1 unsigned long x = 5;
2 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
3 #define CHILDREN 8
4 void * q7_8( void * arg )
5 {
6   for (int ctr=1; ctr<=10; ctr++)
7   {
8     x++;
9   }
10   pthread_exit( NULL );
11 }
12 int main()
13 {
14   pthread_t tid[CHILDREN];
15   int i;
16   for ( i = 0 ; i < CHILDREN ; i++ )
17   {
18     int * t = malloc( sizeof( int ) );
19     *t = 2 + i * 2;
20     pthread_create( &tid[i], NULL, q7_8, NULL );
21   }
22   for ( i = 0 ; i < CHILDREN ; i++ )
23   {
24     pthread_join( tid[i], NULL );
25   }
26   printf("x is %ld\n", x);
27   return EXIT_SUCCESS;
28 }
```

7. **(6 POINTS)** What output **CANNOT** occur? Indicate the best answer.

   (a) `67`
   (b) `85`
   (c) `35`
   (d) `87`
   (e) `12`
   (f) `All outputs are possible.`

8. **(6 POINTS)** Where should `pthread_mutex_lock()` and `pthread_mutex_unlock()` be placed to protect the critical section. Indicate the best answer.

   (a) `Before 16 and after 21`
   (b) `Before 22 and after 25`
   (c) `Before and after 24`
   (d) `Before and after 8`
   (e) `Before and after 26`
   (f) `There is no critical section`

9. **(6 POINTS)** Consider the pseudocode for the **Producer/Consumer** problem. What code should be inserted ar **Position 1** and **Position 2** to make this pseudocode correct? Indicate the best answer.

```
int n = 20;
buffer_t buffer[n];
semaphore empty_slots = n;
semaphore used_slots = 0;

/* producer */                        /* consumer */
while ( 1 )                           while ( 1 )
{                                     {
  item = produce_next_item();           P( used_slots );
  P( empty_slots );                       item = remove_from_buffer();
    add_to_buffer( item );              /* Position 2 */
  /* Position 1 */                      consume( item );
}                                     }
```

(a) Position 1: V(empty_slots); Position 2: V(used_slots);

(b) Position 1: P(used_slots); Position 2: P(empty_slots);

(c) Position 1: V(used_slots); Position 2: V(empty_slots);

(d) Position 1: P(empty_slots);, Position 2: P(used_slots);

(e) No answer is correct

(f) All answers will work correctly

10. **(6 POINTS)** What distinguishes a binary semaphore from a counting semaphore? Indicate the best answer.

(a) Binary semaphores can only be implemented with mutexes, counting semaphores cannot have mutexes

(b) Counting semaphores can only be used for process synchronization, not data synchronization

(c) Binary semaphores provide unique (single process) access to a resource or critical section

(d) Only counting semaphores define the `V()` operation.

(e) All of the above

(f) None of the above

11. **(6 POINTS)** Assume that the server program below has just started running and has received the following 4 datagrams: "I never eat cereal, but I know a professor who does!\n", "Ping\n", "Ping\n", "Ping\n". Receiving which of the following messages would result in a the server sending a return datagram?

Note that in UDP, when a datagram is received, if the given buffer is not large enough, datagrams are truncated (i.e., bytes in the datagram beyond the size of the buffer are ignored).

Indicate the best answer.

```
#define MAXBUFFER 8
int main()
{
  int flag = 0;
  char buffer[ MAXBUFFER + 1 ];
  int sd = socket( AF_INET, SOCK_DGRAM, 0 );
  struct sockaddr_in server;
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl( INADDR_ANY );
  server.sin_port = htons( 8128 );
  bind( sd, (struct sockaddr *) &server, sizeof( server ) );
  struct sockaddr_in client;
  int n, len = sizeof( client );
  while ( 1 )
  {
    n = recvfrom( sd, buffer, MAXBUFFER, 0, (struct sockaddr *) &client,
                        (socklen_t *) &len );
    if ( n > 0 )
    {
      flag = (flag + n) % 17;
      if (flag == 7 )
      {
        sendto( sd, "Okay, enough!", 13, 0, (struct sockaddr *) &client, len );
      }
}
else return EXIT_FAILURE;
```

(a) `"\n"`
(b) `"g\n"`
(c) `"ng\n"`
(d) `"ing\n"`
(e) `"Ping\n"`
(f) None of the above

12. **(6 POINTS)** How many distinct values could variable `x` have at the end of the `main()` function in the code below? Indicate the best answer.

```
void * q12( void * arg )
{
  int * s = (int *)arg;
  *s += 3;
  return NULL;
}

int main()
{
  pthread_t tid1, tid2;
  int x = 5;
  pthread_create( &tid1, NULL, q12, &x );
  pthread_create( &tid2, NULL, q12, &x );
  x = 9;
  pthread_join( tid1, NULL );
  pthread_join( tid2, NULL );

  /* what could variable x be here? */

  return EXIT_SUCCESS;
}
```

(a) Only one possible value

(b) Exactly two possible values

(c) Exactly three possible values

(d) Exactly four possible values

(e) Exactly five possible values

(f) Exactly six possible values

13. **(12 POINTS)** What is the **exact** terminal output of the code below? If multiple outputs are possible, succinctly describe all possibilities. You may use a diagram or enumerate the outcomes.

```
void * q13( void * arg )
{
  int * s = (int *)arg;
  *s += 3;
  return NULL;
}
int main()
{
  int *x = malloc(sizeof(int));
  *x = 5;
  int pid1, pid2;
  pid1 = fork();
  *x = 7;
  if (pid1 != 0) pid2 = fork();
  if (pid1 == 0)
  {
    q13(x);
  } else if (pid2 == 0) {
    q13(x);
  } else {
    *x += 9;
  }
  printf("x = %d\n", *x);
  free(x);
  int status;
  if (pid1 != 0 && pid2 != 0)
  {
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);
  }
  return EXIT_SUCCESS;
}
```

Write your answer to this question on the answer sheet or type it into your text file for submission via Submitty.

14. **(16 POINTS)** What is the **exact** terminal output of the code below? If multiple outputs are possible, succinctly describe all possibilities. You may use a diagram or enumerate the outcomes.

```
void * q14( void * args )
{
  int t = *(int *)args;
  char buffer[32];
  int rc = read( t, buffer, 9 );
  buffer[rc] = '\0';
  printf( "%s", buffer );
  return NULL;
}

int main()
{
  char * s = "CUCKOO FOR COCOA PUFFS";
  int p[2];
  close( 2 );
  close( 0 );
  pipe( p );
  write( p[1], s, 11 );
  pid_t pid = fork();
  pthread_t t1, t2, t3, t4;
  if ( pid == 0 )
  {
    fprintf( stderr, "%s", s );
    printf( "I'M " );
    pthread_create( &t1, NULL, q14, &p[0] );
    pthread_create( &t2, NULL, q14, &p[0] );
    pthread_join( t1, NULL );
    pthread_join( t2, NULL );
  }
  else /* pid > 0 */
  {
    wait( NULL );
    pthread_create( &t3, NULL, q14, &p[0] );
    pthread_create( &t4, NULL, q14, &p[0] );
    pthread_join( t3, NULL );
    pthread_join( t4, NULL );
  }
  return EXIT_SUCCESS;
}
```

Write your answer to this question on the answer sheet or type it into your text file for submission via Submitty.