Operating Systems
Test 1 – Spring 2016

1. Examine the following correct program.
    - Assume that the OS always schedules threads according to the rule that a new thread is allowed to run until completion starting immediately upon being created or unblocked, unless it is explicitly blocked (by a sleep or wait, etc.). What would this program print under such conditions?

    - In reality, an OS would not have such a scheduling algorithm, because it would make it impossible to achieve concurrency with threads: instead, the threads would interleave. Are there any *critical sections* in this code? If so, please clearly mark them in the code below.

```
int x = 2;

void *thread_A( void *arg )
{
            int   y = 3;

            x = x + y;
            printf( "x is %d, y is %d.\n", x, y );

            pthread_exit( NULL );
}

void *thread_B( void *arg )
{
            int   y = x;

            x = x + y;
            printf( "x is %d, y is %d.\n", x, y );

            pthread_exit( NULL );
}

int main()
{
            pthread_t  tid;
            int        y = 1;

            pthread_create( &tid, NULL, thread_A, NULL );
            pthread_create( &tid, NULL, thread_B, NULL );

            printf( "x is %d, ", x );

            printf( "y is %d.\n", y );

            pthread_exit( 0 );
}
```

2. Examine the following correct program. Make the same assumption about scheduling as in the first part of question 1. Fill in the following table indicating the order of the operations that will take place on the just the semaphores and the variable **x** (first 2 lines are entered for you):

Main Thread	***Thread A***	***Thread B***
sem_init A to 1		
sem_init B to 1		

```
int x = 5;
sem_t A, B;

void *thread_A( void *arg )
{
     sem_wait( &A );
     x = x + 1;
     sem_post( &A );
     pthread_exit( NULL );
}

void *thread_B( void *arg )
{
     sem_wait( &B );
     x = x + 2;
     sem_post( &B );
     pthread_exit( NULL );
}

int main()
{
     pthread_t tid;

     sem_init( &A, 0, 1 );
     sem_init( &B, 0, 1 );

     sem_wait( &A );
     sem_wait( &B );
     pthread_create( &tid, NULL, thread_A, NULL );
     pthread_create( &tid, NULL, thread_B, NULL );
     sem_post( &B );
     sem_post( &A );

     pthread_exit( 0 );
}
```

3. Examine the following correct program. Please describe ***precisely*** how this program will behave. The system call **`sleep`** puts a thread to sleep for the given number of seconds.

```
sem_t A, B;

void *threadA( void *ign )
{
        sem_wait( &A );

        printf( "Outer area\n" );
        sleep( 3 );

        sem_wait( &B );

        printf( "Inner area\n" );
        sleep( 3 );

        sem_post( &B );
        sem_post( &A );

        pthread_exit( NULL );
}


int main()
{
        pthread_t     threads[THREADS];
        int           i;

        sem_init( &A, 0, THREADS );
        sem_init( &B, 0, 1 );

        for ( i = 0; i < THREADS; i++ )
                pthread_create( &threads[i], NULL, threadA, NULL );

        for ( i = 0; i < THREADS; i++ )
                pthread_join( threads[i], NULL );

        printf( "Finished\n" );
        pthread_exit( 0 );
}
```

4. Examine the following code which reads lines from a file into an array of strings.
   - Write a loop that traverses the array, and for each string in the array, calls a function with the signature

     ```
     void reverse ( char *str );
     ```

     Also write the function reverse, which reverses its string argument (e.g., “what?” becomes “?tahw”)

   - Write a second loop that sorts the array, in place, by the length of each string (longest to shortest).

   - Write a third loop that prints each string, and then frees the memory for it.

```
#define MAX_LINES        10
#define BUFSZ            128

int main()
{
    int    i = 0, count = 0;
    char   *lines[MAX_LINES];
    char   buf[BUFSZ];
    FILE   *fp;

    if ( ( fp = fopen( "something.txt", "rw" )) == NULL )
           exit( -1 );

    while ( fgets( buf, BUFSZ-1, fp ) != NULL && i < MAX_LINES )
    {
           lines[i] = (char *) malloc( strlen(buf) + 1 );
           strcpy( lines[i], buf );
           i++;
           count++;
    }
```

5. Examine the following correct program.
   - Write down what will be printed.

   - Will the output of this program always be the same every time it is run? Why or why not?

```
int main()
{
        pid_t           pid;
        int             x = 3;
        int             status;

        if ( ( pid = fork() ) == 0 )
        {
                x++;
                printf( "x is %d.\n", x );
                if ( ( pid = fork() ) == 0 )
                {
                        x++;
                        printf( "x is %d.\n", x );
                }
                else
                {
                        waitpid( pid, &status, 0 );
                        printf( "x is %d.\n", x );
                }
        }
        else
        {
                waitpid( pid, &status, 0 );
                printf( "x is %d.\n", x );
        }
        printf( "x is %d.\n", x );
}
```