CSSE 332 -- Operating Systems

Rose-Hulman Institute of Technology

Exam 2

Name: ________________________________

Section: ________________________________

Question	Points	Score
Problem 1	15	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	55	
Total:	100	

Problem 1. (15 points) Consider we have two threads `T`$_1$ and `T`$_2$ that are waiting on a certain condition variable `cv`. In other words, both threads have called `pthread_cond_wait(&cv, &m);`, for some mutex lock `m`.

Assume now that thread `T`$_3$ performs a signal operation on the `cv` condition variable by calling `pthread_cond_signal(&cv);`

Check all statements that are **correct** from the statements below.

A. Both `T`$_1$ and `T`$_2$ will wake up.

B. Exactly one of `T`$_1$ and `T`$_2$ will wake up.

C. If `T`$_1$ called `pthread_cond_wait(&cv, &m);` before `T`$_2$, then it is guaranteed that `T`$_1$ will wake up before `T`$_2$.

D. Before going to sleep, `T`$_1$ will release the lock `m`.

E. If `T`$_1$ wakes up, then it will first attempt to lock `m`.

Problem 2. (10 points) Mohammad is a developer that newly joined *super cool startup* as a systems developer. His first task is to write a parallel algorithm, using `pthreads`, where each thread will read an input file (`char *in_file`), write to an output file (`char *out_file`), and perform a specific function indicated by a function pointer (`int (*op)(char *, char *)`). In other words, we would like each thread to call the function:

```c
/*
 * the op function is passed it as an argument, so what it actually
   does is
 * determined at runtime.
 */
int v = op(in_file, out_file);
// do stuff with the return value v, if needed!
```

Mohammad will need to create `100` of such threads, but should make the code as simple and efficient as possible. In other words, he should be able to write a generic thread code that would work regardless of the values for `in_file`, `out_file`, and `op`.

However, when looking at the `man` page for `pthread_create`, Mohammad is stumped by the fact that it accepts a single argument. In the box below, describe a way for him to be able to use `pthread_create` in a way that makes each one of his threads accept three arguments, `char * in_file`, `char *out_file`, and `int (*op)(char*, char*)`. Show a sample call to `pthread_create` that Mohammad can use as part of his program.

Problem 3. (10 points) After you have helped him with the previous question, Mohammad goes on to write the following piece of code:

```c
void *thread1(void *ignored) {
  // some initialization code
  execlp("ls", "ls", "-a", NULL);
  perror("execlp failed!");
  return 0;
}

void *thread2(void *ignored) {
  // some threaded code...
  return 0;
}

int main(int argc, char **argv) {
  pthread_t t1, t2;

  pthread_create(&t1, 0, thread1, 0);
  pthread_create(&t2, 0, thread2, 0);

  pthread_join(t1);
  pthread_join(t2);

  printf("Everything is done\n");
  return 0;
}
```

In the box below, describe why Mohammad is an idiot (specifically for writing such code, not in general. Save those other remarks for course evals!).

Hint: You might find it useful to think about the differences between threads and processes, then reflect on what can possibly go wrong in the code listing above.

Problem 4. (10 points) Consider the following code listings for two threads, `T`$_1$ and `T`$_2$. You can assume that `lk1` and `lk2` are mutex locks that are defined globally, `data_array` and `data_matrix` are global variables, and `complex_op` and `other_complex_op` are functions defined elsewhere.

```c
void *thread1(void *id) {
  int tid = *(int*)id;

  printf("Th %d started...\n",
   tid);
  pthread_mutex_lock(&lk1);
  pthread_mutex_lock(&lk2);

  // critical section!
  complex_op(data_array,
   data_matrix);
  // end of critical section!

  pthread_mutex_unlock(&lk1);
  pthread_mutex_unlock(&lk2);

  return 0;
}
```

```c
void *thread2(void *id) {
  int tid = *(int*)id;

  printf("Th %d started...\n",
   tid);
  pthread_mutex_lock(&lk2);
  pthread_mutex_lock(&lk1);

  // critical section!
  other_complex_op(data_array,
   data_matrix);
  // end of critical section!

  pthread_mutex_unlock(&lk2);
  pthread_mutex_unlock(&lk1);

  return 0;
}
```

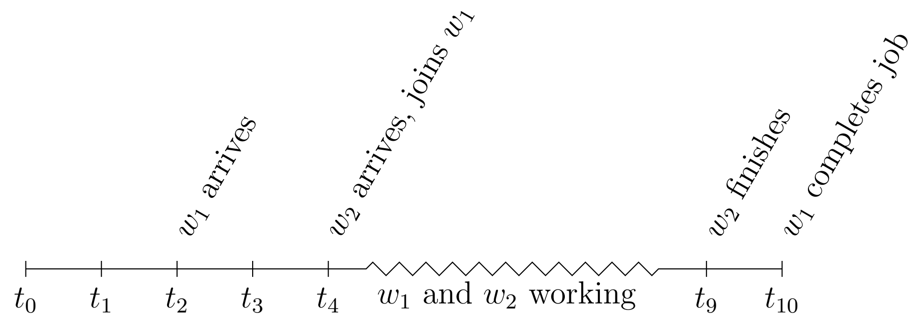
In the box below, describe a sample execution trace of `T`$_1$ and `T`$_2$ that would lead to a **deadlock** state, i.e., a state where both threads are stuck, and neither of them can make any progress forward.

Note: The question below is a **design** question, there is no one right solution. Use your creative thinking along with the tools we learned in this class to design a possible solution. Should you need to make any assumptions, please state them clearly in the answer box below.

Problem 5. Consider a factory where each job requires two workers to work on it. In addition, such jobs **cannot** be started until both workers have arrived and are ready to work on them. In other words, if a worker w_1 picks up a job j_1, then they cannot start working on j_1 before another worker (say w_2) comes in and helps them with the job. It is then and only then that both workers can start working on j_1.

Once a job is started by two workers (say w_1 and w_2), then both workers can work independently and can thus finish at different times. **It does not matter the order in which the workers finish**. For example, say w_1 arrives first and waits for w_2. Then w_2 arrives and both can work on the job. Then, it is perfectly okay if w_2 finishes working on the job before w_1. However, it is essential that the worker that finishes the job (i.e., the second one to complete, or w_1 in this case) marks the job as complete so that other workers would not redo it over again.

Here's an example timeline of the above scenario:



At the start of the day, the manager marks all jobs as `idle`, i.e., no worker has picked up the job yet. You can assume that each day, there are only `NUM_JOBS` jobs to complete.

We would like to write the synchronization code for a worker thread that would implement the concurrency semantics described above. You do not have to worry about the manager or the jobs themselves.

Answer the following questions in the space allocated for each of them. You should aim for correctness rather than syntax. In other words, it is okay to use pseudocode (e.g., `lock(m) instead of pthread_mutex_lock(&m)`). Should you need to make any assumptions, make sure to state those for every question.

(a) (15 points) At a given point in time, a job j can be in one of many states, depending on what is going on in the factory. In the box below, describe each state that a job j can be in. Make sure to give each state a label and an English description. We have already discussed the `idle` state, so I have added it below as an example.

`idle:` No worker has picked up the job yet.

(b) (5 points) For a worker thread, describe the state of the world (or the concurrency state) that the thread must be aware of.

(c) (10 points) For a worker thread, describe the waiting conditions, if any.

(d) (10 points) In correspondence to your waiting conditions, describe any signaling/broadcast that a worker thread must do and **specify when it should be done**.

Hint: Try to be as efficient as possible, i.e., avoid a broadcast when a signal is enough.

(e) (15 points) Finally, write a pseudocode for a worker thread that combines your state of the world, your waiting conditions, your signaling/broadcast, and any condition variables/mutex locks you might need.

Hint: A worker might first have to search for an available job to either pick up or join.

Hint: You may assume that workers search the jobs **sequentially**, starting from j_0 up to $j_{\texttt{NUM\_JOBS}-1}$ every time. In other words, workers favor joining a job over picking up a new one.