Section 10 Solutions

Exercise 1

a) **List some reasons why it's better to use multiple threads within the same process rather than multiple processes running the same program**

Processes are more expensive, since they need their own address space. Threads are more lightweight.

b) What benefits could there be to using multiple processes instead of multiple threads?

Memory safety and (possible) crash tolerance. Processes can't overwrite each other's work because they don't share an address space. Multiple processes can keep running independently if one crashes (depends of the task), whereas one thread seg faulting could crash the whole program.

c) Which registers will for sure be different between two threads that are executing different functions?

The stack pointer is guaranteed to be different, since threads have their own stacks.
The program counters run independently, but might hold the same value if two threads are running the same function.

d) How does the OS distinguish the threads?

Thread IDs. The OS will track its own data about threads, including the current register states, and the pthread_t type is used as an identifier from the user program (similar to how a file descriptor identifies a file or socket).

Exercise 2

Consider the following multithreaded program…

```
int g = 0;
void *worker(void *ignore) {
  for (int k = 1; k <= 3; k++) {
    g = g + k;
  }
  printf("g = %d\n", g);
  return NULL;
}

int main() {
  pthread_t t1, t2;
  int ignore;
  ignore = pthread_create(&t1, NULL, &worker, NULL);
  ignore = pthread_create(&t2, NULL, &worker, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  return EXIT_SUCCESS;
}
```

Give three different possible outputs (there are many)

Here are a few

g = 6 **g = 12**	**g = 12** **g = 12**	**g = 7** **g = 9**	**g = 6** **g = 11**	**g = 7** **g = 10**

What are the possible final values of the global variable ‘g’? (circle all possible)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15+

See diagram for sample interleavings that lead to each possible result:
https://courses.cs.washington.edu/courses/cse333/19sp/sections/10/worksheet-sol-diagram.pdf

Exercise 3

It's payday! It's time for UW to pay each of the 333 TAs their monthly salary. Each of the bank account is inside the *bank_accounts[]* array and the person who is in charged of paying the TAs is a 333 student and decided to use pthreads to pay the TAs by adding 1000 into each bank account. Here is the program the student wrote:

```
// Assume all necessary libraries and header files are included
const int NUM_TAS = 10;

static int bank_accounts[NUM_TAS];
static pthread_mutex_t sum_lock;

void *thread_main(void *arg) {
  int *TA_index = reinterpret_cast<int*>(arg);

  pthread_mutex_lock(&sum_lock);
  bank_accounts[*TA_index] += 1000;
  pthread_mutex_unlock(&sum_lock);

  delete TA_index;
  return nullptr;
}

int main(int argc, char** argv) {
  pthread_t thds[NUM_TAS];
  pthread_mutex_init(&sum_lock, nullptr);

  for (int i = 0; i < NUM_TAS; i++) {
    int *num = new int(i);
    if (pthread_create(&thds[i], nullptr, &thread_main, num) != 0) {
      /*report error*/
    }
  }


  pthread_mutex_destroy(&sum_lock);
  return 0;
}
```

Exercise 3

a) Does the program increase the TAs' bank accounts correctly? Why or why not?

No its not correct. It needs to use pthread_join to wait for each thread to finish before exiting the main program. pthread_exit() might not be the best solution here. You want to check the return value of join to make sure the transaction applied rather than just exiting and trusting the threads to finish successfully. Gotta get those TA dolla's.

b) Could we implement this program using processes instead of threads? Why would or why wouldn't we want to do this?

We could, but doing so would require some way for the processes to communicate with each other so that the data structure can be "shared" (remember that inter-process communication can be difficult and time consuming). It is much easier to just use threads since each thread could directly access the data structure.

c) Assume that all the problems, if any, are now fixed. The student discovers that the program they wrote is kinda slow even though its a multithreaded program. Why might it be the case? And how would you fix that?

Because there is a lock over the entire bank account array, so only one thread can increase the value of one account at a time and there is no difference from incrementing each account sequentially. To fix this, we can have one lock per account so that multiple threads can increment the account at the same time. (With the current setup, we could also just not use a lock since we know that no thread will have a conflicting TA_index. For a more generalized program, it would be better to use the first answer.)

Exercise 4

Calculating primes is slow. In C++, use 10 threads to calculate the primes less than 1,000. Then, print them out in ascending order:

```
#define NTHREAD 10
struct Bounds {
  int lo;
  int hi;
  Bounds(int lo, int hi): lo(lo), hi(hi) {}
};

bool isPrime(int num) { … }

void *getPrimes`(void *data) {
  Bounds *b = reinterpret_cast<Bounds*>(data);
  // setup a way to store the primes we find in order
  std::vector<int> *primes = new std::vector<int>();
  // calculate primes
  for (int i = b->lo; i < b->hi; ++i) {
    if (isPrime(i))
      primes->push_back(i);
  }
  return reinterpret_cast<void*>(primes);
}
int main() {
  // make space to store our threads and data
  std::vector<std::unique_ptr<Bounds>> bounds;
  pthread_t threads[NTHREAD];
  // create and run our threads
  int err;
  for (int i = 0; i < NTHREAD; i++) {
    int lo = (i * 100) + 1;
    int hi = ((i + 1) * 100) + 1;
    bounds.push_back(std::unique_ptr<Bounds>(new Bounds(lo,hi)));

    if ((err = pthread_create(threads+i, nullptr, &getPrimes,
                              bounds.back().get())) != 0) {
      std::cout << "thread create error on thread " << i <<
std::endl;
      std::cout << strerror(err) << std::endl;
     return EXIT_FAILURE;
    }
  }
```

```
  // wait for each thread to finish and get its data
  for (int i = 0; i < NTHREAD; i++) {
    // wait for thread, storing its return value
    std::vector<int> *out;
    err = pthread_join(threads[i], reinterpret_cast<void**>(&out));
    if (err != 0) {
      std::cout << "thread join error on thread " << i << std::endl;
      std::cout << strerror(err) << std::endl;
      continue;
    }
    // print the data
    for (auto prime : *out) {
      std::cout << prime << std::endl;
    }
    delete out;
  }
  return 0;
}
```