# CISS 370: Operating Systems
# Assignment 3
# Due date: March 1, 2023

## Question 1 (15 points)

Read Section 5.3.2 (Case Study: Thread-Safe Bounded Queue).

```
// Thread-safe queue interface
const int MAX = 10;
class TSQueue {
        // Synchronization variables
        Lock lock;

        // State variables
        int items[MAX];
        int front;
        int nextEmpty;

        public:
                TSQueue();
                ~TSQueue(){};
                bool tryInsert(int item);
                bool tryRemove(int *item);
};

// Initialize the queue to empty
// and the lock to free.
TSQueue::TSQueue() {
        front = nextEmpty = 0;
}

// Try to insert an item. If the queue is
// full, return false; otherwise return true.
bool TSQueue::tryInsert(int item) {
        bool success = false;

        lock.acquire();
        if ((nextEmpty - front) < MAX) {
                items[nextEmpty % MAX] = item;
                nextEmpty++;
                success = true;
        }
        lock.release();
        return success;
}
// Try to remove an item. If the queue is
// empty, return false; otherwise return true.
bool TSQueue::tryRemove(int *item) {
        bool success = false;

        lock.acquire();
        if (front < nextEmpty) {
                *item = items[front % MAX];
                front++;
                success = true;
        }
        lock.release();
        return success;
}
```

**Figure 5.3: A thread-safe bounded queue. For implementation simplicity, we assume the queue stores integers (rather than arbitrary objects) and the total number of items stored is modest.**

```
// TSQueueMain.cc
// Test code for TSQueue.
int main(int argc, char **argv) {
        TSQueue *queues[3];
        sthread_t workers[3];
        int i, j;
        // Start worker threads to insert.
        for (i = 0; i < 3; i++) {
                queues[i] = new TSQueue();
                thread_create_p(&workers[i], putSome, queues[i]);
        }

        // Wait for some items to be put.
        thread_join(workers[0]);

        // Remove 20 items from each queue.
        for (i = 0; i < 3; i++) {
                printf("Queue %d:\n", i);
                testRemoval(&queues[i]);
        }
}
// Insert 50 items into a queue.
void *putSome(void *p) {
        TSQueue *queue = (TSQueue *)p;
        int i;
        for (i = 0; i < 50; i++) {
                queue->tryInsert(i);
        }
        return NULL;
}

// Remove 20 items from a queue.
void testRemoval(TSQueue *queue) {
        int i, item;
        for (i = 0; i < 20; j++) {
                if (queue->tryRemove(&item))
                        printf("Removed %d\n", item);
                else
                        printf("Nothing there.\n");
        }
}
```
**Figure 5.5: This code creates three TSQueue objects and then adds and removes some items from these queues. We use thread_create_p instead of thread_create so that we can pass to the newly created thread a pointer to the queue it should use.**

Precisely describe the set of possible outputs that could occur when the program shown in Figure 5.5 is run. If you are interested in running the code, I believe there are some errors in *TSQueueMain.cc*, and to correct them, I suggest the following:

1. Download the code from https://homes.cs.washington.edu/~tom/code/
2. Unzip the file
3. Open *TSQueueMain.cc*
   - Line 14: Replace "`sthread_t`" with "`thread_t`"
   - Line 15: Remove "`, j`"
   - Line 30: Replace "`&queues`" with "`queues`"
   - Line 49: Replace "`j++`" with "`i++`"
   - Line 54: Remove "`}`"
4. Compile and link
   ```
   gcc -g -Wall -Werror -D_POSIX_THREAD_SEMANTICS -c  Lock.cc
   TSQueue.cc thread.c TSQueueMain.cc

   gcc -lpthread TSQueueMain.o TSQueue.o thread.o Lock.o  -o TSQueue
   -lstdc++
   ```
5. Run
   ```
   ./TSQueue
   ```

**Question 2 (10 points)**
Suppose that you mistakenly create an automatic (local) variable v in one thread t1 and pass a pointer to v to another thread t2. Is it possible that a write by t1 to some variable other than v will change the state of v as observed by t2? If so, explain how this can happen and give an example. If not, explain why not.

**Question 3 (10 points)**
Suppose that you mistakenly create an automatic (local) variable v in one thread t1 and pass a pointer to v to another thread t2. Is it possible that a write by t2 to v will cause t1 to execute the wrong code? If so, explain how. If not, explain why not.

**Question 4 (10 points)**

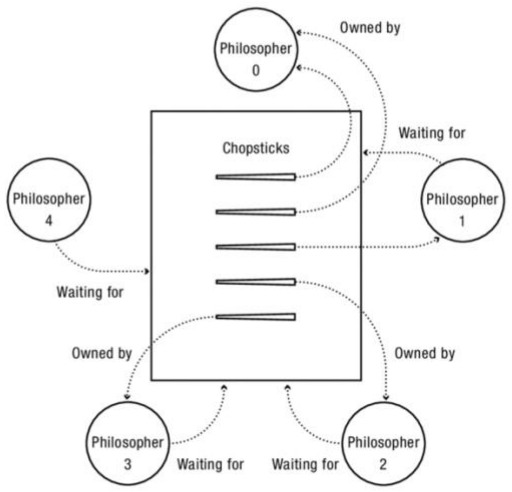

**Figure 6.17:** Graph representation of the state of a Dining Philosophers system that includes a cycle among waiting threads and resources but that is not deadlocked. Circles represent threads, boxes represent resources, dots within a box represent

Consider the variation of the Dining Philosophers problem shown in Figure 6.17, where all unused chopsticks are placed in the center of the table and any philosopher can eat with any two chopsticks.

One way to prevent deadlock in this system is to provide sufficient resources. For a system with n philosophers, what is the minimum number of chopsticks that ensures deadlock freedom? Why?

**Question 5 (18 points)**
We previously covered this exercise in class. Please redo it to ensure you have a solid understanding of different scheduling algorithms.

For each of the following scheduling algorithms, draw a Gantt chart and calculate the average turnaround time, waiting time, and response time for each process:

- First Come First Serve (FCFS)

- Shortest Job First (SJF) non-preemptive
- Shortest Remaining Time First (SRTF) preemptive
- Round Robin (RR) with a quantum of 4
- Priority scheduling preemptive and non-preemptive

For each scheduling algorithm, submit a Gantt chart and a table with the following information:

| **Process** | **Turnaround time** | **Waiting time** | **Response time** |
|---|---|---|---|
| $P_1$ | | | |
| $P_2$ | | | |
| $P_3$ | | | |
| $P_4$ | | | |
| Avg | | | |

**Question 6 (10 points)**
For shortest job first, if the scheduler assigns a task to the processor, and no other task becomes schedulable in the meantime, will the scheduler ever preempt the current task? Why or why not?

**Question 7 (10 points)**
Suppose you do your homework assignments in SJF order. After all, you feel like you are making a lot of progress! What might go wrong?

**Question 8 (17 points)**
Three tasks, A, B, and C are run concurrently on a computer system.

- Task A arrives first at time 0, and uses the CPU for 100 ms before finishing.
- Task B arrives shortly after A, still at time 0. Task B loops ten times; for each iteration of the loop, B uses the CPU for 2 ms and then it does I/O for 8 ms.
- Task C is identical to B, but arrives shortly after B, still at time 0.

Assuming there is no overhead to doing a context switch, identify when A, B and C will finish for each of the following CPU scheduling disciplines:

a. FIFO
b. Round robin with a 1 ms time slice
c. Round robin with a 100 ms time slice
d. Multilevel feedback with four levels, and a time slice for the highest priority level is 1 ms.
e. Shortest job first