

Final Project
CMSC 312, Spring 2014

Report

Melissa Nierle
Allen Woods

Part 2: Design Descriptions (e.g., locks or other synchronization mechanisms used, data structures, threads, etc.).

For this project simulating a bank teller counter, our program utilizes a few different techniques to ensure efficiency. The main design techniques used were mutexes, condition variables, and threads.

First, we used threading the most throughout the entire program. Threads make it possible for the program to utilize several processors throughout runtime. They make it possible for multitasking to happen simultaneously during the execution of different portions of the one program. We create a few threads at the beginning that are utilized throughout the entire program. Each thread represents a process, which is why there are so many threads used, because they are necessary to ensure multi-tasking. In our code, the lines 60 through 71, (beginning with the `if()`), show a few different threads being created and used. This block itself produces customers sometimes when the scheduler wakes. However, threads are not the only synchronization technique used in this block, or throughout the program. Equally as important are the mutex and conditional variable operations

The mutex class, or mutual exclusive, is a primitive used for synchronization. Mutexes effectively keeps data that is shared from being used by multiple threads at the same time. Essentially, it makes it possible for a thread to be in control of a mutex for a given time, and restrict access to any other threads during that time. This is very beneficial for the bank teller system as it is necessary to make sure that threads are not over-lapping in their processes. For reference, in our code, lines 29 through 43 provide an example of how we utilized mutex. The customer queue is locked and cannot be used until it locks `qcount_mutex`, in the next line, the

lock of `cout_mutex` prevents other threads from outputting to the screen. These are just two examples in the program of mutexes, however this technique is utilized throughout the program.

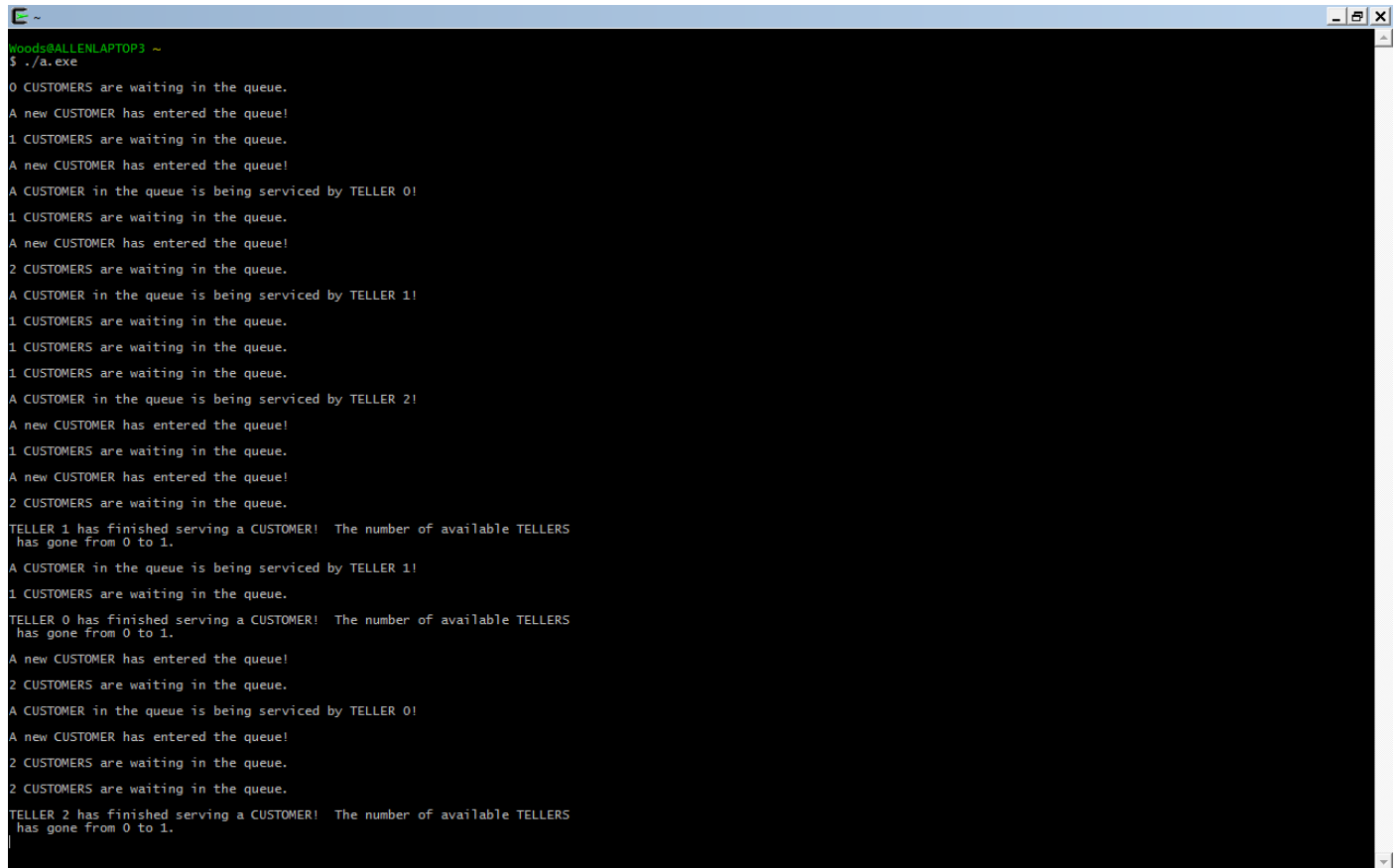
Lastly, conditional variables, or CVs are used to further the synchronization of processes. Conditional variables are similar to mutexes in that they are both primitives that help prevent simultaneous use of the same tasks. However, conditional variables are both slightly more advanced, and lock with a signaling mechanism. Therefore, when threads have to wait for a source to become available, conditional variables are used. The way CVs work is by using a variable as a signal, the producer will signal the variable, thus notifying the waiting threads. In our code, one place we use CVs is in the block from line 271 to line 277. Here we are signaling the scheduler with the conditional variable “condition2” that a teller is available. Before this signal was made, the scheduler was awaiting a teller to become available, and by utilizing the condition variable, we essentially wake it up, to let it know that a new process can be performed because a teller is now available.

The coupling of these three techniques ensures the smooth synchronization of our Bank Teller program.

Part 3. Test Results (2-5 pages to show proper synchronizations).

Here are screen shots of only the first 2 minutes of operation, however the program can run indefinitely

40 sec:

A terminal window titled "woods@ALLENLAPTOP3 ~" showing the execution of a program. The output displays a sequence of events: customers entering and waiting in a queue, and tellers servicing customers. The number of customers waiting in the queue fluctuates between 0 and 2. Teller 1 is the first to be serviced, followed by Teller 0, and then Teller 2. The program appears to be running correctly with proper synchronization.

```
woods@ALLENLAPTOP3 ~  
$ ./a.exe  
0 CUSTOMERS are waiting in the queue.  
A new CUSTOMER has entered the queue!  
1 CUSTOMERS are waiting in the queue.  
A new CUSTOMER has entered the queue!  
A CUSTOMER in the queue is being serviced by TELLER 0!  
1 CUSTOMERS are waiting in the queue.  
A new CUSTOMER has entered the queue!  
2 CUSTOMERS are waiting in the queue.  
A CUSTOMER in the queue is being serviced by TELLER 1!  
1 CUSTOMERS are waiting in the queue.  
1 CUSTOMERS are waiting in the queue.  
1 CUSTOMERS are waiting in the queue.  
A CUSTOMER in the queue is being serviced by TELLER 2!  
A new CUSTOMER has entered the queue!  
1 CUSTOMERS are waiting in the queue.  
A new CUSTOMER has entered the queue!  
2 CUSTOMERS are waiting in the queue.  
TELLER 1 has finished serving a CUSTOMER! The number of available TELLERS  
has gone from 0 to 1.  
A CUSTOMER in the queue is being serviced by TELLER 1!  
1 CUSTOMERS are waiting in the queue.  
TELLER 0 has finished serving a CUSTOMER! The number of available TELLERS  
has gone from 0 to 1.  
A new CUSTOMER has entered the queue!  
2 CUSTOMERS are waiting in the queue.  
A CUSTOMER in the queue is being serviced by TELLER 0!  
A new CUSTOMER has entered the queue!  
2 CUSTOMERS are waiting in the queue.  
2 CUSTOMERS are waiting in the queue.  
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS  
has gone from 0 to 1.
```

1 Min:

```
C~
2 CUSTOMERS are waiting in the queue.
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 2!
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
TELLER 0 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A new CUSTOMER has entered the queue!
A CUSTOMER in the queue is being serviced by TELLER 0!
1 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
2 CUSTOMERS are waiting in the queue.
TELLER 0 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 0!
1 CUSTOMERS are waiting in the queue.
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
A CUSTOMER in the queue is being serviced by TELLER 2!
A new CUSTOMER has entered the queue!
1 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
2 CUSTOMERS are waiting in the queue.
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 2!
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
TELLER 1 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 1!
0 CUSTOMERS are waiting in the queue.
```

2 Min:

```
C~
A CUSTOMER in the queue is being serviced by TELLER 2!
A new CUSTOMER has entered the queue!
1 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
2 CUSTOMERS are waiting in the queue.
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 2!
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
TELLER 1 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
A CUSTOMER in the queue is being serviced by TELLER 1!
0 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
TELLER 1 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
1 CUSTOMERS are waiting in the queue.
A CUSTOMER in the queue is being serviced by TELLER 1!
0 CUSTOMERS are waiting in the queue.
0 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
1 CUSTOMERS are waiting in the queue.
1 CUSTOMERS are waiting in the queue.
TELLER 2 has finished serving a CUSTOMER! The number of available TELLERS
has gone from 0 to 1.
1 CUSTOMERS are waiting in the queue.
A CUSTOMER in the queue is being serviced by TELLER 2!
0 CUSTOMERS are waiting in the queue.
0 CUSTOMERS are waiting in the queue.
A new CUSTOMER has entered the queue!
1 CUSTOMERS are waiting in the queue.
```

Part 4. Discussion of possible deadlock and your solutions

A deadlock occurs when two threads (or more), are blocked from processing anymore because they are waiting for each other. After running our program continuously for several hours, even with the computer in sleep mode, it never came to a deadlock. We believe there is no possibility of deadlock for this program. However, in general, an example would be if two of our threads began to wait for each other to finish at the exact same time, resulting in a stalemate. The solution in our program to deadlock is contained in the block from line 222 to line 228. Here, we tended to the possibility of no teller having any costumers at one given time. This situation is handled by first making sure that the teller is waiting on the scheduler. A CV is used for the scheduler to wait for the teller to become available. The next step in preventing this deadlock is through another CV and mutex, it waits until the scheduler has a customer to send to the teller. This way we make sure no other process begins before a customer goes to a teller. This prevents deadlock by making sure that both the consumer and producer are not waiting on each other. We isolate the teller and ensure that a customer is sent to it before any other proceedings.