Ian Brown
Prof. Greg Ozbirn
SE 4348.003
April 6, 2015

Project 2 Design Specifications

Semaphores

Semaphore max_capacity = 10 :

This semaphore is used to regulate the maximum number of 10 customers within the post office at a time.

Semaphore customerRequest = 0 :

This semaphore guarantees that the customer will output their request before the postal worker begins to process the requested task. This is a result of the way I coordinated the customer and postal worker getting each other's information, which has the postal worker receiving the customer's information first and having the possibility to start processing their task before the request is output.

Semaphore workerNbrSet = 0 :

This semaphore keeps the customer from reading the postal worker number from the customer's specified location within the workerAssignment array before the postal worker has placed their number in this location.

Semaphore workerAvailable = 3 :

The workerAvailable semaphore is used to enforce the acquiring of postal workers by customers, so that only 3 postal workers are available to acquire and customers must wait until a postal worker becomes available before they attempt to acquire one.

Semaphore scaleMutex = 1 :

The scaleMutex semaphore is used to make sure that only one postal worker tries to access the scale resource at a time.

Semaphore queueMutex = 1 :

The queueMutex semaphore is used to regulate the use of the queue. Through there only being one permit available for this semaphore, it guarantees that only 1 thread, customer or postal worker, is accessing the queue at a time to make sure the transfer of customer information does not get interrupted/messed up.

Semaphore pw_wait = 0 :

The pw_wait semaphore is used to make sure that the customer receives the queue mutex before the postal worker, so that the postal worker won't try to read customer information that is not in the queue yet. Without this semaphore, the postal worker may try to grab the queue mutex before the customer can.

Semaphore[50] custFinished = {0} :

The custFinished semaphore array is used to make sure the correct customer is being released when its job is done. With a normal semaphore, the current customer that had acquired a postal worker for the longest amount of time would be released, even if they had not finished their task yet. By providing each customer thread a semaphore, it guarantees that the correct customer is released.

Pseudocode

Project2 class

```
main():
    create PostOffice object and Thread arrays for customers and postal workers
    create instances of customer threads and start them
    create instances of postal worker threads, set them to daemon threads, and start
        them
    join customer threads as they finish and print that they are joined
    exit program
```

PostOffice class

```
PostOffice():
    set all semaphores contained within the semaphore array custFinished to 0

acquireWorker( customer_number ):
    return the postal worker number contained at workerAssignment[customer_number]

enqueue( customer_number, task ):
    add customer number to queue
    add requested task to the queue

dequeue() :
    return the value at the head of the queue

setWorkerAssignment( worker_number, location ) :
    make the location within worker assignment equal to the worker_number
```

Customer class

```
Customer() :
    assign customer number
    assign PostOffice parameter to local instance of PostOffice class, office
    randomly assign customer task

run() :
    try {
        Print that customer was created
        semWait(office.max_capacity)
        Print that customer has entered the post office
        semWait(office.workerAvailable)
        semWait(office.queueMutex)
        office.enqueue(customer_number, task)
        semSignal(office.pw_wait)
        semSignal(office.queueMutex)
        semWait(office.workerNbrSet)
```

```
                Print customer task request using office.acquireWorker and stringTask methods
                semSignal(office.customerRequest)
                semWait(customerFinished[customer_number])
                Print that task is completed and that customer is leaving the post office
                semSignal(office.workerAvailable)
                semSignal(office.max_capacity)
            }catch (thread interrupted) {return from interrupt}


        stringTask() :
            switch statement to determine which String-version of task should be output
```

PostalWorker class

```
        PostalWorker() :
            assign postal worker number and assign PostOffice parameter to local PostOffice object,
                office

        run() :
            try {
                print postal worker was created
                while (true) {
                    semWait(office.pw_wait)
                    semWait(office.queueMutex)
                    gatherCustInformation()
                    print which customer postal worker is serving
                    semSignal(office.queueMutex)
                    office.setWorkerAssignment(postal_worker_number, customer_number)
                    semSignal(office.workerNbrSet)
                    semWait(office.customerRequest)
                    processTask()
                    Print that postal worker is finished with customer task
                    semSignal(customerFinished[customer_number])
            }catch(thread interrupts) {return}

        gatherCustInformation() :
            dequeue head value of queue using office.dequeue and assign it to customer number
            dequeue head value of queue using office.dequeue and assign it to task

        processTask() :
            switch statement to determine thread sleep time

            case "mailing package" :
                semWait(office.scaleMutex)
                Print that scale is acquired
                sleep for 2 seconds
                semSignal(office.scaleMutex)
                Print that scale is released
```