1. **List of Semaphores, its purpose, and initial values:**

**max\_capacity:** Its initial value is 10.

This semaphore makes sure that, only 10 customers can be inside the post office at any point of time.

**mutex1:** Its initial value is 1.

This semaphore makes sure that there is mutual exclusion for the customer ids queue, which is the same for both customer and postal worker threads.

**mutex2:** Its initial value is 1.

This semaphore makes sure that there is mutual exclusion for the worker ids queue, which is the same for both customer and postal worker threads.

**mutex3:** Its initial value is 1.

This semaphore makes sure that there is mutual exclusion for the tasks queue, which is the same for both customer and postal worker threads.

**cust\_ready:** Its initial value is 0.

This semaphore makes sure that the postal worker waits for the next customers in the line.

**postal\_worker\_ready:** Its initial value is 0.

This semaphore makes sure that the customer waits for the postal worker to be available for his service

**finished[customerThreads]:** The initial values for all is 0.

This semaphore informs that the postal worker has serviced a particular customer task and that customer can leave the post office, once the postal worker signals this semaphore.

**cust\_order[customerThreads]:** The initial values for all is 0.

This semaphore informs the customer to place the order whenever the postal worker is ready to start the service for that customer.

**processOrder[postalWorkerThreads]:** The initial values for all is 0.

This semaphore is used to let the postal worker know that the customer has placed the order and that postal worker can start processing that order.

**scales\_ready:** Its initial value is 1.

This semaphore makes sure that only one postal worker thread can access this resource at a time.

1. **Pseudo code for the implementation of post office:**

/\* **program** post office simulation \*/

semaphore max\_capacity = 10;

semaphore mutex1=1, mutex2=1, mutex3=1;

semaphore cust\_ready=0;

semaphore postal\_worker\_ready=0;

semaphore finished[customerThreads]={0};

semaphore scales ready=1;

semaphore cust\_order[customerThreads]={0};

semaphore processOrder[postalWorkerThreads]={0};

void customer(){

int custId,task,workerID;

task=rand()%3;

semWait(max\_capacity);

enter\_postOffice();

semWait(postal\_worker\_ready);

semWait(mutex1);

enqueue1(custId);

semSignal(cust\_ready);

semSignal(mutex1);

semWait(cust\_order[custId]);

semWait(mutex2);

workerID=dequeue2();

semSignal(mutex2);

custTask();

semWait(mutex3);

enqueue0(task);

semSignal(processOrder[workerID]);

semSignal(mutex3);

semWait(finished[custId]);

finishedTask();

exit\_postOffice();

semSignal(max\_capacity);

}

void postalWorker(){

int custId,workerId,task;

while(1){

semSignal(postal\_worker\_ready);

semWait(cust\_ready);

semWait(mutex1);

custId=dequeue1();

semSignal(mutex1);

startService();

semWait(mutex2);

enqueue2(workerId);

semSignal(cust\_order[custId]);

semSignal(mutex2);

semWait(processOrder[workerId]);

semWait(mutex3);

task=dequeue0();

semSignal(mutex3);

semWait(scales\_ready);

process\_mailPackage();

semSignal(scales\_ready);

finishService();

semSignal(finished[custId]);

}

}

void main(){

parbegin(customer ….. 50 times, postalWorker, postalWorker, postalWorker);

}

Note: queue0() is for tasks, queue1() is for customer IDs and queue2() is for worker IDs.