**Customer**(customerNo)

{

int orderNo, cPostalWorkerID;

customer\_created();

wait(max\_capacity);

enter\_office();

wait(postalWorker\_ready);

wait(mutex1);

orderNo = rand()%3;

enqueue( 0 , customerID );

signal(customer\_ready);

signal(mutex1);

wait(askOrder[customerID]);

wait(mutex2);

dequeue( 1, cPostalWorkerID );

signal(mutex2);

askPostalWorker();

wait(mutex3);

enqueue( 2, orderNo );

signal(placeOrder[cPostalWorkerID]);

signal(mutex3);

wait(finished[customerID]);

finished\_job();

exit\_office();

sem\_post(&max\_capacity);

}

**PostalWorker**(postalWorkerID)

{

postalWorker\_created();

while(1)

{

int pwCustomerID,pwOrderNo;

signal(postalWorker\_ready);

wait(customer\_ready);

wait(mutex1);

dequeue(0, pwCustomerID);

signal(mutex1);

serving\_customer();

wait(mutex2);

enqueue(1, postalWorkerID);

signal(askOrder[pwCustomerID]);

signal(mutex2);

wait(placeOrder[postalWorkerID]);

wait(mutex3);

dequeue(2, pwOrderNo);

signal(mutex3);

switch(pwOrderNo)

{

case 0:

sleep(1);

break;

case 1:

sleep(1.5);

break;

case 2:

wait(scales\_ready);

sleep(2);

signal(scales\_ready);

break;

}

finished\_serving();

signal(finished[pwCustomerID]);

}

}

**void main**()

{

parbegin (Customer(0),.……..Customer(49), PostalWoker(0), PostalWoker(1), PostalWoker(2));

}

**SEMAPHORES**

max\_capacity= 10;

This semaphore makes sure that only 10 customers are inside the post office at any point of time during execution.

customer\_ready=0;

This semaphore synchronizes the postal worker threads to wait for the next customer in line.

postalWorker\_ready = 0;

This semaphore makes the customer wait for the postal worker to be created and then for the postal worker to become available for use.

finished[50]= {0};

This semaphore informs the specific customer that it has been serviced by the postal worker. Now, the customer can leave the post office. This semaphore synchronizes the conclusion of the relationship between a postal worker thread and a customer thread.

askOrder[50]={0};

This semaphore informs the specific customer to place the order. The customer thread enqueue the order number into the relevant queue and signals the placeOrder[3] semaphore to let the postal worker thread know that it can start processing the order now.

placeOrder[3]= {0};

This semaphore is to inform the postal worker that the customer has placed the order and he can begin processing it. He then uses the scales\_ready semaphore depending on the order that it has received.

scales\_ready=1;

This semaphore is used to ensure that only one thread/postal worker can access the scales(resource).

mutex1=1;

This semaphore makes sure that the Customer IDs queue, global to all threads, has mutual exclusion for all threads.

mutex2=1;

This semaphore makes sure that the Postal Worker IDs queue, global to all threads, has mutual exclusion for all threads.

mutex3=1;

This semaphore makes sure that the Order Numbers queue, global to all threads, has mutual exclusion for all threads.