Semaphore list:

Max\_customers: 10 // only ten people are allowed in the building at a time

Scale = 1 //there is only a single scale to be used by each of the

Customer\_ready = 0 //used for when a customer is ready for service by a worker.

Worker\_ready = 3 //sets up the 3 open spots for the workers

MUTEX1 = 1 //used to provide mutual exclusion for creating threads

MUTEX2 = 1 //used to provide mutual exclusion for pushing and popping onto the stack.

MUTEX3 = 1 // used to provide mutual exclusion for print statements, so they do not overlap.

-note: for the sake of ease of reading, print statements and MUTEX3 have not been put in the psudocode.

Finished [50] = {0} //used to make sure that individual threads are correctly finished.

Global items:

Count =0;

Finishedcount=0

Std::stack<int> gqueue;

Void Customer()

{

int task;

int customerid;

wait (max\_customers);

wait (MUTEX1);

customerid = count

count++;

task = random number % 3;

signal (MUTEX1);

wait (MUTEX2);

gqueue.push (customerid);

gqueue.push (task);

Signal (customer\_ready);

Signal (MUTEX2);

Wait (finished[customerid])

Signal (max\_customers)

}

Void Postal\_worker()

{

Int customer\_id;

Int customer\_task;

While (finishedcount>50)

{

Wait(Customer\_ready)

Wait (worker\_ready

Wait (MUTEX2);

customer\_customerid = gqueue.pop()

current\_task = gqueue.pop()

signal (MUTEX2);

switch (customer\_task)

{

case 0: sleep (1); //in seconds

break;

case 1: sleep (1.5); //in seconds

break:

case 2: wait (scale);

sleep (2); //in seconds

signal (scale);

break;

}

finishedcount++;

signal (worker\_ready)

signal (finished[customer\_id])

}

}