HY-345: Operating Systems

Fall Semester 2026

2nd Assignment

E. Markatos

In this exercise you will become familiar with:

- The creation and management of threads (POSIX threads)
- The use of POSIX semaphores for synchronizing multiple threads that cooperate in a shared environment
- The use of mutexes for protecting shared data from concurrent access

Description – Post Office and Parcel Delivery

We consider a post office that receives parcels from customers.

Each parcel must first pass through a bounded intake belt before being classified and forwarded to its destination.

- The post office has **one intake belt** with a capacity of up to **B** parcels
- There are three bounded destination belts, one for each destination type: **LOCAL**, **NATIONAL**, **INTERNATIONAL**, each with a capacity of up to **B** parcels
- **C customer threads** are created. Each customer sends exactly **one** parcel with a random destination
- **E employee threads** are created. Each employee continuously picks parcels from the intake belt, classifies them, and places them onto the correct destination belt
- **Three (3) postman threads** are created. One per destination belt
- After a parcel is delivered, the customer who sent it must be **notified**
- The program terminates after all **C** parcels have been delivered

Program Logic
Customer Behavior (Customer Thread)

Each customer is an independent thread and performs the following steps:

1. Waits a random amount of time (1-5 seconds) before creating their parcel
2. Creates a parcel with a random destination (LOCAL, NATIONAL, or INTERNATIONAL)
    - Customer arrivals should occur in random order
3. Waits until there is space on the intake belt
    - Waiting is implemented using a semaphore
4. Places the parcel onto the intake belt
5. Waits for notification that their parcel has been delivered
    - Waiting is implemented using a per-customer semaphore that is signaled by the postman
6. Receives the notification and exits

Employee Behavior (Employee Thread)

Each employee is an independent thread and continuously runs the following loop:

1. Monitors the intake belt for available parcels
2. Picks up a parcel from the intake belt
3. Waits a random amount of time (1-5 seconds) while processing each parcel
4. Reads the parcel's destination (LOCAL, NATIONAL, INTERNATIONAL)
5. Waits until there is space on the correct destination belt
6. Places the parcel onto the destination belt
7. The program terminates when all C parcels have been delivered

Postman Behavior (Postman Thread)

Each postman is an independent thread. There are exactly 3 postman threads, one per destination belt.

Each postman continuously runs the following loop:

1. Monitors their assigned destination belt for available parcels
2. Picks up a parcel from the belt
3. Delivers the parcel, simulating the process with a random delay of 1-5 seconds
4. Notifies the customer who sent the parcel that it has been delivered
5. Increments the global delivered counter
6. The program terminates when all C parcels have been delivered

Synchronization (Semaphores and Mutexes)

The solution must use both semaphores and mutexes.

Bounded buffer pattern / semaphore usage:

Each belt must be implemented as a bounded buffer using a circular array. Your Belt struct should track the buffer, its capacity, the current item count, and the read/write positions. It must include the appropriate semaphores and a mutex to enable safe concurrent access by multiple threads.

Adding a parcel to a belt must block when the belt is full; removing must block when it is empty. Use two semaphores per belt, one counting available slots and one counting available items, combined with a mutex that protects the shared buffer during the actual read or write.

Termination

Particular attention must be paid to the termination of the program. When all parcels have been delivered, all threads must terminate cleanly, including those currently blocked in a waiting state.

Execution Parameters:

The program must accept three parameters from the command line:

./postoffice <num_customers> <num_employees> <belt_capacity>

Example execution for 4 customers, 2 employees, belt capacity 3:

./postoffice 4 2 3

The terminal output below shows an example run.

Full terminal output after implementation:

[CUSTOMER 2] Created parcel (LOCAL)

[CUSTOMER 0] Created parcel (INTERNATIONAL)

[CUSTOMER 0] Placed parcel on intake belt (1/3)

[CUSTOMER 2] Placed parcel on intake belt (2/3)

[EMPLOYEE 0] Picked up parcel from Customer 0 (INTERNATIONAL)

[EMPLOYEE 0] Placed parcel on INTERNATIONAL belt (1/3)

[EMPLOYEE 1] Picked up parcel from Customer 2 (LOCAL)

[EMPLOYEE 1] Placed parcel on LOCAL belt (1/3)

[CUSTOMER 1] Created parcel (NATIONAL)

[CUSTOMER 1] Placed parcel on intake belt (1/3)

[POSTMAN 2] Delivered parcel from Customer 0 (INTERNATIONAL). Total delivered: 1/4

[CUSTOMER 0] Notified: parcel delivered.

[POSTMAN 0] Delivered parcel from Customer 2 (LOCAL). Total delivered: 2/4

[CUSTOMER 2] Notified: parcel delivered.

[CUSTOMER 3] Created parcel (LOCAL)

[CUSTOMER 3] Placed parcel on intake belt (2/3)

[EMPLOYEE 0] Picked up parcel from Customer 1 (NATIONAL)

[EMPLOYEE 0] Placed parcel on NATIONAL belt (1/3)

[EMPLOYEE 1] Picked up parcel from Customer 3 (LOCAL)

[EMPLOYEE 1] Placed parcel on LOCAL belt (1/3)

[POSTMAN 1] Delivered parcel from Customer 1 (NATIONAL). Total delivered: 3/4

[CUSTOMER 1] Notified: parcel delivered.

[POSTMAN 0] Delivered parcel from Customer 3 (LOCAL). Total delivered: 4/4

[CUSTOMER 3] Notified: parcel delivered.

All parcels delivered. Closing post office.

Evaluation Criteria

1. Correct synchronization (no race conditions)
2. Proper use of mutexes and semaphores
3. Realistic simulation (correct synchronization between all thread types)
4. Console output should be clear and provide all necessary information
5. Completeness of comments and README

Submission

The submission will be done through **E-Learn**.

You must create a zip file that contains:

1. Your **code**, including explanatory comments
2. A **Makefile** that supports the following commands:
    - make all -> compiles and produces the executable
    - make clean -> deletes all generated files
3. A **README** describing your implementation and any notes you consider important

A suggested structure for your deliverables:

- main.c
- postoffice.c
- postoffice.h
- Makefile
- README.md

Notes

- This exercise is strictly individual. Submitted files will be checked by an anti-plagiarism system. Any detected plagiarism will result in automatic zero
- At the beginning of your code, include in comments your full name and student ID number
- Keep your code tidy and include comments explaining its functionality where needed
- The use of code generated by AI is strictly prohibited