

# Operating Systems 2 - Assignment Semaphores

## Supermarket Simulation

### *What are semaphores and threads?*

First let us define what both are before we begin the program.

A **thread**, is a path of execution within a process. There can be multiple threads within a process.

A **semaphore** is a kernel mechanism for signaling. They are variables that are non-negative and shared between threads to help synchronize process in a multi-processing environment.

### Step 1: Define the problem

**Customer Behavior:** - Customers arrive at the supermarket every 2 to 4 seconds.

**Resources Available:** - 20 carts. - 10 handheld scanners. - 3 cashiers for normal checkout. - 2 checkout terminals for handheld scanner users.

**Synchronization Needs:** - Manage the availability of carts, handheld scanners, cashiers, and checkout terminals.

### Step 2: Identify Activities (Threads)

#### Customers Threads:

1. Arrives at the supermarket.
2. Takes a cart (if available).
3. Takes a handheld scanner (if available and if not a cart if available).
4. Shops for groceries.
5. Chooses the checkout method (cashier (cart) or terminal (handheld scanner or cart)).
6. checks out.
7. Returns the cart (after bringing groceries to car / bicycle)/ handheld scanner.

### Step 3: Define Semaphores and Shared Data

#### Semaphores:

- **Carts Semaphore:** Manages the availability of 20 carts.
- **Choose cashier Semaphore:** Manages choosing the cashier.
- **Join chosen cashier queue Semaphore:** Manages the queue for the chosen cashier.
- **Update that the chosen cashier is ready Semaphore:** Manages the turns for the cashier.

- **Update the cashier queue Semaphore:** Manages the updating of the cashier queue.
- **Return cart Semaphore:** Manages returning the carts.
- **Handheld Scanners Semaphore:** Manages the availability of 10 handheld scanners.
- **Choose terminal Semaphore:** Manages choosing the terminal.
- **Join terminal queue Semaphore:** Manages the queue for the terminal.
- **Update that the chosen terminal is ready Semaphore:** Manages the turns for the terminal.
- **Update the terminal queue Semaphore:** Manages the updating of the terminal queue.
- **Return scanners Semaphore:** Manages returning the handheld Scanners.

## Step 4: Flowchart

Flowchart will be at the end in the appendix [Figure 10].

## Step 5: Initializations

Now that we have identified the program lets look at the code. But, first lets look at the initializations and functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_CARTS 20
#define NUM_CASHIERS 3
#define NUM_TERMINAL 2

sem_t cartSemaphore;
sem_t terminalSemaphore;
sem_t terminalSemaphorequeue;
sem_t terminal_Semaphores[NUM_TERMINAL];
sem_t terminal_queue_mutex[NUM_TERMINAL];
sem_t terminal_ready[NUM_TERMINAL];
sem_t cashiers_Semaphores[NUM_CASHIERS];
sem_t cashier_queue_mutex[NUM_CASHIERS];
sem_t cashier_ready[NUM_CASHIERS];

int queue = 0;
int NUM_CUSTOMERS = 20;
int carts_available = NUM_CARTS;
```

```

int scannersAvailable = 10;
int scannersPickup = 0;
int cashier_queue[NUM_CASHIERS] = {0};
int scanner_queue[NUM_CASHIERS] = {0};

void *customer(void *arg);
void *customerScanner(void *arg);
void shopping(int customer_id);
void checkout(int customer_id, int chosen_cashier);
void checkoutScanner(int customer_id, int chosen_terminal);
void return_cart(int customer_id);
void backupTerminal(int customer_id, int chosen_terminal);

```

The main libraries I am using here are:

```

#include <pthread.h>
#include <semaphore.h>

```

In C programming, to utilize threading and synchronization, you can use the pthread and semaphore libraries.

Using **pthread.h** for Thread Management To work with threads in C, include the pthread.h header file. This library provides functions to create threads. Here are some of the key functions:

Creating a Thread:

```

pthread_t thread;
pthread_create(&thread, NULL, thread_function, NULL);

```

**pthread\_create** initiates a new thread. \* The first argument is a pointer to your thread. \* The second argument can specify thread attributes, but NULL means default attributes. \* The third argument is the function to be executed by the thread. \* The fourth argument is the argument passed to the thread function.

Joining a Thread:

```

pthread_join(thread, NULL);

```

**pthread\_join** makes the calling thread wait for the specified thread to terminate. \* The first argument is the thread. \* The second argument can be used to obtain the return value of the thread, but NULL means you don't want the return value.

Using **semaphore.h** for Synchronization To handle synchronization between threads, you can use semaphores. The semaphore.h library provides these functions.

Initializing a Semaphore:

```
sem_t semaphore;  
sem_init(&semaphore, 0, 1);
```

`sem_init` initializes a semaphore. \* The first argument is a pointer to the semaphore. \* The second argument is 0 if the semaphore is to be shared between threads of the same process; otherwise, it is non-zero. \* The third argument sets the initial value of the semaphore.

Waiting on a Semaphore:

```
sem_wait(&semaphore);
```

`sem_wait` decrements (locks) the semaphore. If the semaphore's value is greater than zero, you can decrement if not it's in a lock state until a `sem_post` happens, which will be explained next.

```
sem_post(&semaphore);
```

`sem_post` increments (unlocks) the semaphore. If the semaphore's value is zero and `sem_post` was called it will increment by 1 to unlock.

```
sem_destroy(&semaphore);
```

`sem_destroy` destroys the semaphore and frees any resources allocated to it.

Now Let's begin understanding what each function does:

### Function Explanations

**1. void \*customer(void \*arg);** This function simulates the behavior of a customer who picks a shopping cart and continues through the shopping process until he check out at a cashier or terminal. It involves selecting the shortest queue in cashier, and deciding on a checkout method depending on the queue.

**2. void \*customerScanner(void \*arg);** This function simulates a customer who uses a handheld scanner for self-checkout. It involves the customer picking waiting the queue, and proceeding to a terminal for the final checkout process.

**3. void shopping(int customer\_id);** This function simulates the time taken by a customer to shop. The `customer_id` parameter is used to differentiate between customers.

**4. void checkout(int customer\_id, int chosen\_cashier);** This function simulates the checkout time at a cashier for a given customer. The `customer_id` parameter is used to differentiate between customers, and `chosen_cashier` indicates which cashier the customer has selected for checkout.

5. `void checkoutScanner(int customer_id, int chosen_terminal);`  
This function simulates the checkout time using a handheld scanner at a terminal. The `customer_id` parameter is used to differentiate between customers, and `chosen_cashier` indicates which terminal the customer has selected for checkout. Additionally, there's a 25% chance that the customer will be checked.

6. `void return_cart(int customer_id);` This function simulates the time of returning a cart after the customer has completed their shopping and checkout. The `customer_id` parameter is used to differentiate between customers.

7. `void backupTerminal(int customer_id, int chosen_terminal);`  
This function provides a backup checkout option for customers with a cart if the cashier queue is too long. The `customer_id` parameter is used to differentiate between customers, and `chosen_cashier` indicates which terminal the customer has selected for checkout.

Now that everything is set up we can write the code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_CARTS 20
#define NUM_CASHIERS 3
#define NUM_TERMINAL 2

sem_t cartSemaphore;
sem_t terminalSemaphore;
sem_t terminalSemaphorequeue;
sem_t terminal_Semaphores[NUM_TERMINAL];
sem_t terminal_queue_mutex[NUM_TERMINAL];
sem_t terminal_ready[NUM_TERMINAL];
sem_t cashiers_Semaphores[NUM_CASHIERS];
sem_t cashier_queue_mutex[NUM_CASHIERS];
sem_t cashier_ready[NUM_CASHIERS];

int queue = 0;
int NUM_CUSTOMERS = 20;
int carts_available = NUM_CARTS;
```

```

int scannersAvailable = 10;
int scannersPickup = 0;
int cashier_queue[NUM_CASHIERS] = {0};
int scanner_queue[NUM_CASHIERS] = {0};

void *customer(void *arg);

void *customerScanner(void *arg);

void shopping(int customer_id);

void checkout(int customer_id, int chosen_cashier);

void checkoutScanner(int customer_id, int chosen_terminal);

void return_cart(int customer_id);

void backupTerminal(int customer_id, int chosen_terminal);

/*
   This function simulates a customer who uses a handheld scanner for self-checkout.
   It involves the customer picking waiting the queue, and proceeding to a terminal
   for the final checkout process.
*/

void *customerScanner(void *arg) {
    int customer_id = *((int *) arg);

    sem_wait(&terminalSemaphore);

    // check if scannersAvailable < 5 to then return the scanners to the pickup location
    if (scannersAvailable < 5 && scannersPickup > 4) {
        printf("An Employee is moving the handheld scanners to the pickup Location.\n");
        sleep(4);
        scannersAvailable = scannersAvailable + scannersPickup;
        scannersPickup = 0;
        printf("An Employee has returned the handheld scanners to the pickup Location. Handl\n",
            scannersAvailable);
    }

    scannersAvailable--;
    printf("Customer [%d] takes a handheld scanner. Handheld scanners available: %d\n",
        scannersAvailable);
    sem_post(&terminalSemaphore);
}

```

```

// Customer starts shopping
shopping(customer_id);

int chosen_terminal = -1;

// Choose the terminal with the no queue
for (int i = 0; i < 1; i++) {
    sem_wait(&terminal_Semaphores[i]);

    // Check terminals queue for the smallest, if full join queue
    if (scanner_queue[i] == 0) {
        chosen_terminal = i;
        scanner_queue[chosen_terminal]++;
    } else if (scanner_queue[i + 1] == 0) {
        chosen_terminal = i + 1;
        scanner_queue[chosen_terminal]++;
    } else {
        queue++;
        printf("Customer [%d] has joined the queue. The current queue is [%d].\n", customer_id, queue);
    }

    sem_post(&terminal_Semaphores[i]); // Release semaphore for each cashier
}

// loop to find the next empty terminal
if (chosen_terminal == -1) {
    sem_wait(&terminalSemaphorequeue);
    while (1) {
        if (scanner_queue[0] == 0) {
            chosen_terminal = 0;
            scanner_queue[chosen_terminal]++;
            queue--;
            sem_post(&terminalSemaphorequeue);
            break;
        } else if (scanner_queue[1] == 0) {
            chosen_terminal = 1;
            scanner_queue[chosen_terminal]++;
            queue--;
            sem_post(&terminalSemaphorequeue);
            break;
        }
    }
}

sem_wait(&terminal_queue_mutex[chosen_terminal]);
printf("Customer [%d] is checking out at terminal [%d].\n", customer_id, chosen_terminal);

```

```

sem_post(&terminal_queue_mutex[chosen_terminal]);

// Wait for the chosen terminal to signal the customer's turn
sem_wait(&terminal_ready[chosen_terminal]);

// Simulate checkout time
checkoutScanner(customer_id, chosen_terminal);

// Customer finishes checkout, signal to the next customer in line
sem_post(&terminal_ready[chosen_terminal]);

// Update terminal queue
sem_wait(&terminal_queue_mutex[chosen_terminal]);
printf("Customer [%d] checking at terminal [%d] has checked out.\n", customer_id, chosen_terminal);
scanner_queue[chosen_terminal]--;
sem_post(&terminal_queue_mutex[chosen_terminal]);

// Return the scanner and increment the pickup
sem_wait(&terminalSemaphore);
scannersPickup++;
printf("Customer [%d] returned the scanner.\n", customer_id);
sem_post(&terminalSemaphore);

pthread_exit(NULL); // exit the thread
}

/*
This function simulates the behavior of a customer who picks a shopping cart
and continues through the shopping process until he checks out at a cashier
or terminal. It involves selecting the shortest queue in the cashier, and
deciding on a checkout method depending on the queue.
*/

void *customer(void *arg) {
    int customer_id = *((int *) arg);

    // Customer arrives and tries to take a cart
    sem_wait(&cartSemaphore);
    if (carts_available > 0) {
        carts_available--;
        printf("Customer [%d] takes a cart. Carts available: %d\n", customer_id, carts_available);
        sem_post(&cartSemaphore);

        // Customer starts shopping
        shopping(customer_id);
    }
}

```



```

int chosen_cashier = 0;
int chosen_terminal;

// Choose the cashier with the smallest or no queue
for (int i = 1; i < NUM_CASHIERS; i++) {
    sem_wait(&cashiers_Semaphores[i]);

    if (cashier_queue[i] < cashier_queue[chosen_cashier]) {
        chosen_cashier = i;
    }
    sem_post(&cashiers_Semaphores[i]); // Release semaphore for each cashier
}

// if cashier queue for each one is more than two go to the terminal for check out
if (cashier_queue[chosen_cashier] >= 2) {
    if (scanner_queue[0] <= 1) {
        chosen_terminal = 0;
        scanner_queue[chosen_terminal]++;
        backupTerminal(customer_id, 0);
        pthread_exit(NULL); // exit the thread
    } else if (scanner_queue[1] <= 0) {
        chosen_terminal = 1;
        scanner_queue[chosen_terminal]++;
        backupTerminal(customer_id, 1);
        pthread_exit(NULL); // exit the thread
    }
}

// Join the chosen cashier's queue
sem_wait(&cashier_queue_mutex[chosen_cashier]);
cashier_queue[chosen_cashier]++;
if (cashier_queue[chosen_cashier] - 1 == 0) {
    printf("Customer [%d] is checking out at Cashier [%d].\n", customer_id, chosen_cashier);
} else {
    printf("Customer [%d] is checking out at Cashier [%d] and his turn is after %d customers\n",
           customer_id, chosen_cashier,
           cashier_queue[chosen_cashier] - 1);
}

sem_post(&cashier_queue_mutex[chosen_cashier]);

// Wait for the chosen cashier to signal the customer's turn
sem_wait(&cashier_ready[chosen_cashier]);

```

```

        // Simulate checkout time
        checkout(customer_id, chosen_cashier);

        // Customer finishes checkout, signal to the next customer in line
        sem_post(&cashier_ready[chosen_cashier]);

        // Update cashier queue
        sem_wait(&cashier_queue_mutex[chosen_cashier]);
        printf("Customer [%d] checking at Cashier [%d] has checked out.\n", customer_id, chosen_cashier);
        cashier_queue[chosen_cashier]--;
        sem_post(&cashier_queue_mutex[chosen_cashier]);

        // Return the cart
        return_cart(customer_id);
    } else {

        // No cart available, go home empty handed
        sem_post(&cartSemaphore);
        printf("Customer [%d] goes back home empty handed.\n", customer_id);
    }

    pthread_exit(NULL); // exit the thread
}

/*
This function provides a backup checkout option for customers with a cart if the cashier queue is full.
The `customer_id` parameter is used to differentiate between customers, and `chosen_cashier` is the
which terminal the customer has selected for checkout.
*/

void backupTerminal(int customer_id, int chosen_terminal) {

    sem_wait(&terminal_queue_mutex[chosen_terminal]);
    printf("Customer [%d] is checking out at terminal [%d].\n", customer_id, chosen_terminal);
    sem_post(&terminal_queue_mutex[chosen_terminal]);

    // Wait for the chosen cashier to signal the customer's turn
    sem_wait(&terminal_ready[chosen_terminal]);

    // Simulate checkout time
    checkoutScanner(customer_id, chosen_terminal);

    // Customer finishes checkout, signal to the next customer in line
    sem_post(&terminal_ready[chosen_terminal]);
}

```

```

        // Update cashier queue
        sem_wait(&terminal_queue_mutex[chosen_terminal]);
        printf("Customer [%d] checking at terminal [%d] has checked out.\n", customer_id, chosen_terminal);
        scanner_queue[chosen_terminal]--;
        sem_post(&terminal_queue_mutex[chosen_terminal]);

        // Return the cart and go home
        return_cart(customer_id);
    }

    /*
    This function simulates the time taken by a customer to shop. The `customer_id` parameter is used
    to differentiate between customers, and `chosen_cashier` indicates which cashier the customer is
    */

    void shopping(int customer_id) {
        int shopping_time = (rand() % 10) + 1;; // Random shopping time between 1 to 10 seconds
        printf("Customer [%d] is shopping for %d seconds.\n", customer_id, shopping_time);
        sleep(shopping_time); // wait
    }

    /*
    This function simulates the checkout time at a cashier for a given customer. The `customer_id` parameter is used
    to differentiate between customers, and `chosen_cashier` indicates which cashier the customer is
    */

    void checkout(int customer_id, int chosen_cashier) {
        int checkout_time = (rand() % 8) + 3;; // Random checkout time between 3 to 10 seconds
        printf("Customer [%d] is checking out at Cashier [%d] for %d seconds.\n", customer_id, chosen_cashier, checkout_time);
        sleep(checkout_time);
    }

    /*
    This function simulates the checkout time using a handheld scanner at a terminal. The `customer_id` parameter is used
    to differentiate between customers, and `chosen_cashier` indicates which terminal the customer is using.
    Additionally, there's a 25% chance that the customer will be checked.
    */

    void checkoutScanner(int customer_id, int chosen_terminal) {

        int checkout_time = 1; // Random checkout time 1 seconds
        int randomCheck = rand() % 4; // 25 % check of checking
        if (randomCheck == 0) {

```

```

        printf("Customer [%d] is being checked at Terminal [%d]. Total checkout time is now\n",
               chosen_terminal, checkout_time + 4);
        sleep(checkout_time + 4);
    } else {
        printf("Customer [%d] is checking out at Terminal [%d] for %d seconds.\n", customer_id,
               chosen_terminal, checkout_time);
        sleep(checkout_time); // wait
    }
}

/*
This function simulates the time of returning a cart after the customer has completed their checkout.
The `customer_id` parameter is used to differentiate between customers.
*/

void return_cart(int customer_id) {
    int return_time = (rand() % 2) + 1; // Random return cart time between 1 to 2 seconds
    sleep(return_time); // wait
    sem_wait(&cartSemaphore);
    carts_available++;
    printf("Customer [%d] returned the cart. Carts available: %d\n", customer_id, carts_available);
    sem_post(&cartSemaphore);
}

/*
This function is used to initialize the semaphores, create the threads(customers, cashiers, scanners),
create a customer every 2 to 4 seconds, afterwards join the threads, and finally destroy the threads.
*/

int main() {
    pthread_t customers[NUM_CUSTOMERS];
    int customer_ids[NUM_CUSTOMERS];

    // initialize cart and cashier semaphores
    sem_init(&cartSemaphore, 0, 1);
    for (int i = 0; i < NUM_CASHIERS; i++) {
        sem_init(&cashiers_Semaphores[i], 0, 1);
        sem_init(&cashier_queue_mutex[i], 0, 1);
        sem_init(&cashier_ready[i], 0, 1);
    }

    // initialize scanner and terminal semaphores
    sem_init(&terminalSemaphore, 0, 1);
    sem_init(&terminalSemaphorequeue, 0, 1);
    for (int i = 0; i < NUM_CASHIERS; i++) {
        sem_init(&terminal_Semaphores[i], 0, 1);
    }
}

```

```

        sem_init(&terminal_queue_mutex[i], 0, 1);
        sem_init(&terminal_ready[i], 0, 1);
    }

    srand(time(NULL));

    // create customer thread every 2 to 4 seconds
    for (int i = 0; i < NUM_CUSTOMERS; i++) {
        int useScanner = rand() % 2; // 50% chance cart or scanner
        customer_ids[i] = i + 1;
        if (useScanner == 0) {
            pthread_create(&customers[i], NULL, customer, &customer_ids[i]);
            sleep((rand() % 3) + 2); // Random time between 2 to 4 seconds for customer arrival
        } else {
            if (scannersAvailable > 0) {
                pthread_create(&customers[i], NULL, customerScanner, &customer_ids[i]);
                sleep((rand() % 3) + 2); // Random time between 2 to 4 seconds for customer arrival
            } else {
                printf("No handheld scanners available. Customer [%d] will go for a cart.\n", i);
                pthread_create(&customers[i], NULL, customer, &customer_ids[i]);
                sleep((rand() % 3) + 2); // Random time between 2 to 4 seconds for customer arrival
            }
        }
    }

    // join the threads
    for (int i = 0; i < NUM_CUSTOMERS; i++) {
        pthread_join(customers[i], NULL);
    }

    // return the rest of the scanners
    if (scannersAvailable < 11) {
        printf("An Employee is moving the rest of the handheld scanners to the pickup Location.\n");
        sleep(4);
        scannersAvailable = scannersAvailable + scannersPickup;
        scannersPickup = 0;
        printf("An Employee has returned the handheld scanners to the pickup Location. Handheld scanners available: %d\n",
               scannersAvailable);
    }

    // initialize cart and cashier semaphores
    sem_destroy(&cartSemaphore);
    for (int i = 0; i < NUM_CASHIERS; i++) {
        sem_destroy(&cashiers_Semaphores[i]);
        sem_destroy(&cashier_queue_mutex[i]);
    }

```

```

        sem_destroy(&cashier_ready[i]);
    }

    // destroy scanner and terminal semaphores
    sem_destroy(&terminalSemaphore);
    sem_destroy(&terminalSemaphorequeue);
    for (int i = 0; i < NUM_CASHIERS; i++) {
        sem_destroy(&terminal_Semaphores[i]);
        sem_destroy(&terminal_queue_mutex[i]);
        sem_destroy(&terminal_ready[i]);
    }
    return 0;
}

```

## Simulations

1. Every 2 to 4 seconds a customer arrives and takes a cart to go shopping, there are 20 carts, if no cart is available, the customer goes back home empty-handed.

To test this simulation, I began by first decreasing the number of customers to 6 for this scenario, and I decreased the number of carts to 5. Additionally, I removed returning back carts.

```

Customer [4] takes a cart. Carts available: 1
Customer [4] is shopping for 2 seconds.
Customer [3] checking at Cashier [0] has checked out.
Customer [4] is checking out at Cashier [0].
Customer [4] is checking out at Cashier [0] for 4 seconds.
Customer [5] takes a cart. Carts available: 0
Customer [5] is shopping for 2 seconds.
Customer [5] is checking out at Cashier [1].
Customer [5] is checking out at Cashier [1] for 4 seconds.
Customer [4] checking at Cashier [0] has checked out.
Customer [6] goes back home empty handed.
Customer [5] checking at Cashier [1] has checked out.

```

Figure 1: Scenario: customer goes back home empty-handed

As you can see every 2 to 4 seconds a customer enters, and shown above after the 5 the customer the number of carts are 0, that means that customer 6 will not have a cart, which then it gave the message that Customer [6] goes back home empty handed.

**2. About half the customers take a handheld scanner with them, there are 10 handheld scanners, if none are available, the customer does the shopping the normal way.**

To test this, I began by decreasing the number of handheld scanners to 2 and the number of customers to 3. Additionally, I made it where every customer always starts with a handheld to test.

```
Customer [1] takes a handheld scanner. Handheld scanners available: 1
Customer [1] is shopping for 2 seconds.
Customer [1] is checking out at terminal [0].
Customer [1] is checking out at Terminal [0] for 1 seconds.
Customer [1] checking at terminal [0] has checked out.
Customer [2] takes a handheld scanner. Handheld scanners available: 0
Customer [2] is shopping for 2 seconds.
No handheld scanners available. Customer [3] will go for a cart.
Customer [2] is checking out at terminal [0].
Customer [2] is checking out at Terminal [0] for 1 seconds.
Customer [3] takes a cart. Carts available: 19
Customer [3] is shopping for 2 seconds.
Customer [2] checking at terminal [0] has checked out.
Customer [3] is checking out at Cashier [0].
Customer [3] is checking out at Cashier [0] for 10 seconds.
```

Figure 2: Scenario: if no handheld is available

As we can see, customer 3 went for a cart when the handheld were not available. It gave the message that No handheld scanners available. Customer [3] will go for a cart.

**3. Customers without a handheld scanner go to one of three cashiers, and if there are already customers waiting in a queue for a cashier, they choose the cashier with the least (or none) customers and join the end of that queue. When it is their turn, it takes 3 to 10 seconds to register all the items and pay.**

For this scenario, I began by making the number of customer 10 and then increased the checkout time.

As we can see, when customer 6 arrived to the checkout cashier he saw that all cashier are being used, so he picked the cashier with the least queue, but since all are 1, the customer went with cashier[0] and it says Customer [6] is checking out at Cashier [0] and his turn is after 1 customer.

```

Customer [2] checking at Cashier [1] has checked out.
Customer [5] is checking out at Cashier [1].
Customer [5] is checking out at Cashier [1] for 6 seconds.
Customer [6] takes a cart. Carts available: 15
Customer [6] is shopping for 2 seconds.
Customer [2] returned the cart. Carts available: 16
Customer [6] is checking out at Cashier [0] and his turn is after 1 customer.

```

Figure 3: Scenario: cashier queue

**4. Customers with a handheld scanner go to one of two checkout terminals, if none are available, they wait in one queue, first customer in the queue takes the first terminal which becomes available. Registering the handheld scanner and paying only takes 1 second, after that the handheld scanner becomes immediately available for new customers to pick up.**

For this scenario, I began by disabling the carts and decreasing the shopping, but increasing the checkout time to 10 seconds.

```

Customer [1] is checking out at terminal [0].
Customer [1] is being checked at Terminal [0]. Total checkout time is now 14 seconds.
Customer [2] is checking out at terminal [1].
Customer [2] is checking out at Terminal [1] for 10 seconds.
Customer [3] takes a handheld scanner. Handheld scanners available: 7
Customer [3] is shopping for 2 seconds.
Customer [3] has joined the queue. The current queue is [1].
Customer [4] takes a handheld scanner. Handheld scanners available: 6
Customer [4] is shopping for 2 seconds.
Customer [4] has joined the queue. The current queue is [2].
Customer [5] takes a handheld scanner. Handheld scanners available: 5
Customer [5] is shopping for 2 seconds.
Customer [2] checking at terminal [1] has checked out.
Customer [2] returned the scanner.
Customer [3] is checking out at terminal [1].
Customer [3] is checking out at Terminal [1] for 10 seconds.

```

Figure 4: Scenario: terminal queue

As one can see, customer 1 and 2 both took terminal [0] and [2], and when customer arrived at the terminal he had to wait and was added to the queue as it says **Customer [3] has joined the queue. The current queue is [1]**, then customer 4 was also added to the queue, and because customer [2] checkout before customer [1], customer [3] then checked out at terminal [1] as it says **Customer [3] is checking out at terminal [1]**.



```
Customer [1] takes a cart. Carts available: 19
Customer [1] is shopping for 2 seconds.
Customer [1] is checking out at Cashier [0].
Customer [1] is checking out at Cashier [0] for 10 seconds.
Customer [2] takes a cart. Carts available: 18
Customer [2] is shopping for 2 seconds.
Customer [2] is checking out at Cashier [1].
Customer [2] is checking out at Cashier [1] for 10 seconds.
Customer [3] takes a cart. Carts available: 17
Customer [3] is shopping for 2 seconds.
Customer [3] is checking out at Cashier [2].
Customer [3] is checking out at Cashier [2] for 10 seconds.
Customer [1] checking at Cashier [0] has checked out.
Customer [1] returned the cart. Carts available: 18
Customer [2] checking at Cashier [1] has checked out.
Customer [2] returned the cart. Carts available: 19
Customer [3] checking at Cashier [2] has checked out.
Customer [3] returned the cart. Carts available: 20
```

Figure 5: Scenario: return cart

5. After paying, every customer takes 1 or 2 seconds to bring the groceries to their bicycle or car and return the cart, which then becomes available for new customers, and then they go back home.

In this scenario, we can see that after each customer was done they returned the cart and the carts available were incremented.

## Additional features

1. Every once in a while, a customer with a handheld scanner is randomly checked to see if they scanned all groceries. Implement that you have a 25% chance of being checked. Of course, this takes significantly longer then.

```
Customer [2] is checking out at terminal [0].
Customer [2] is checking out at Terminal [0] for 1 seconds.
Customer [2] checking at terminal [0] has checked out.
Customer [2] returned the scanner.
Customer [3] takes a handheld scanner. Handheld scanners available: 7
Customer [3] is shopping for 2 seconds.
Customer [3] is checking out at terminal [0].
Customer [3] is checking out at Terminal [0] for 1 seconds.
Customer [4] takes a handheld scanner. Handheld scanners available: 6
Customer [4] is shopping for 2 seconds.
Customer [3] checking at terminal [0] has checked out.
Customer [3] returned the scanner.
Customer [4] is checking out at terminal [0].
Customer [4] is being checked at Terminal [0]. Total checkout time is now 5 seconds.
Customer [4] checking at terminal [0] has checked out.
Customer [4] returned the scanner.
```

Figure 6: Scenario: random checks

As we can see in the output, customer [4] was randomly checked, and the normal checkout time is 1 second, however, because he was randomly checked the checkout time increased to 5 seconds as it says here **Customer [4] is being checked at Terminal [0]. Total checkout time is now 5 seconds.**

## 2. Handheld Scanner Pickup

To test this I made the threshold on when the employee refills the handheld scanners when `available < 5`. additionally, it takes the employee 4 seconds to take them to the pickup.

As we can see, after customer [6] took a handheld and the availability became 4, which is less than 5 then the employee returned them which took 4 seconds, and then of course the number of available handheld increased. **An Employee has**

```

Customer [6] takes a handheld scanner. Handheld scanners available: 4
Customer [6] is shopping for 2 seconds.
Customer [5] checking at terminal [0] has checked out.
Customer [5] returned the scanner.
Customer [6] is checking out at terminal [0].
Customer [6] is checking out at Terminal [0] for 1 seconds.
Customer [6] checking at terminal [0] has checked out.
Customer [6] returned the scanner.
An Employee is moving the handheld scanners to the pickup Location.
Customer [8] takes a handheld scanner. Handheld scanners available: 9
Customer [8] is shopping for 2 seconds.
An Employee has returned the handheld scanners to the pickup Location. Handheld scanners available: 9

```

Figure 7: Scenario: handheld scanner pickup

returned the handheld scanners to the pickup Location. Handheld scanners available: 9.

### 3. Implement manual checkout.

To create this simulation, I began by making the number of cashiers to 2 and decreased the shopping time and increased the checkout time. Most importantly, I made it where if the queue is more than one person at each cashier and the terminals have a queue less than 1, then the cart customer can check out at the terminal.

```

Customer [3] is checking out at Cashier [0] for 12 seconds.
Customer [5] is checking out at Cashier [0] and his turn is after 1 customer.
Customer [1] returned the cart. Carts available: 16
Customer [6] takes a cart. Carts available: 15
Customer [6] is shopping for 2 seconds.
Customer [2] checking at Cashier [1] has checked out.
Customer [4] is checking out at Cashier [1] for 12 seconds.
Customer [6] is checking out at Cashier [1] and his turn is after 1 customer.
Customer [7] takes a cart. Carts available: 14
Customer [7] is shopping for 2 seconds.
Customer [2] returned the cart. Carts available: 15
Customer [7] is checking out at terminal [0].
Customer [7] is being checked at Terminal [0]. Total checkout time is now 5 seconds.

```

Figure 8: Scenario: manual checkout

As we can see in the image, when customer [7] came to check out he saw that the queue is more than one and then checked out at one of the free terminals, where it says Customer [7] is checking out at terminal [0], and funny enough a random check occurred.

## Final: Reflection

To reflect on this assignment, I have successfully completed all the tasks with their bonuses. Additionally, I found it to be very enjoyable and I have also learned a lot about semaphores. I believe that working on assignments in this way, you end up getting a better idea about not only the topic theoretically, but also how to use it in a practical manner.

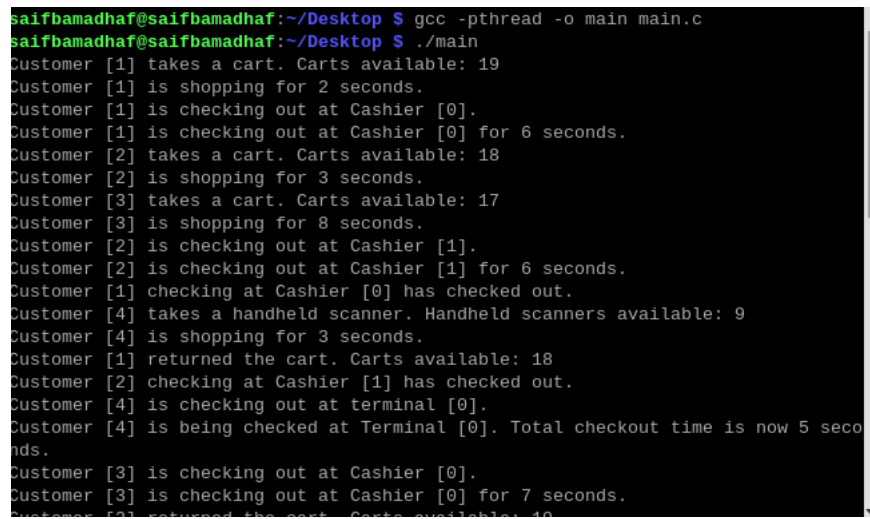
## Help

[https://www.youtube.com/watch?v=YSn8\\_XdGH7c](https://www.youtube.com/watch?v=YSn8_XdGH7c)

<https://www.youtube.com/watch?v=ldJ8WGZVXZk>

## Appendix:

### Running on PI



```
saifbamadhaf@saifbamadhaf:~/Desktop $ gcc -pthread -o main main.c
saifbamadhaf@saifbamadhaf:~/Desktop $ ./main
Customer [1] takes a cart. Carts available: 19
Customer [1] is shopping for 2 seconds.
Customer [1] is checking out at Cashier [0].
Customer [1] is checking out at Cashier [0] for 6 seconds.
Customer [2] takes a cart. Carts available: 18
Customer [2] is shopping for 3 seconds.
Customer [3] takes a cart. Carts available: 17
Customer [3] is shopping for 8 seconds.
Customer [2] is checking out at Cashier [1].
Customer [2] is checking out at Cashier [1] for 6 seconds.
Customer [1] checking at Cashier [0] has checked out.
Customer [4] takes a handheld scanner. Handheld scanners available: 9
Customer [4] is shopping for 3 seconds.
Customer [1] returned the cart. Carts available: 18
Customer [2] checking at Cashier [1] has checked out.
Customer [4] is checking out at terminal [0].
Customer [4] is being checked at Terminal [0]. Total checkout time is now 5 seconds.
Customer [3] is checking out at Cashier [0].
Customer [3] is checking out at Cashier [0] for 7 seconds.
Customer [2] returned the cart. Carts available: 19
```

Figure 9: Running on Raspberry Pi

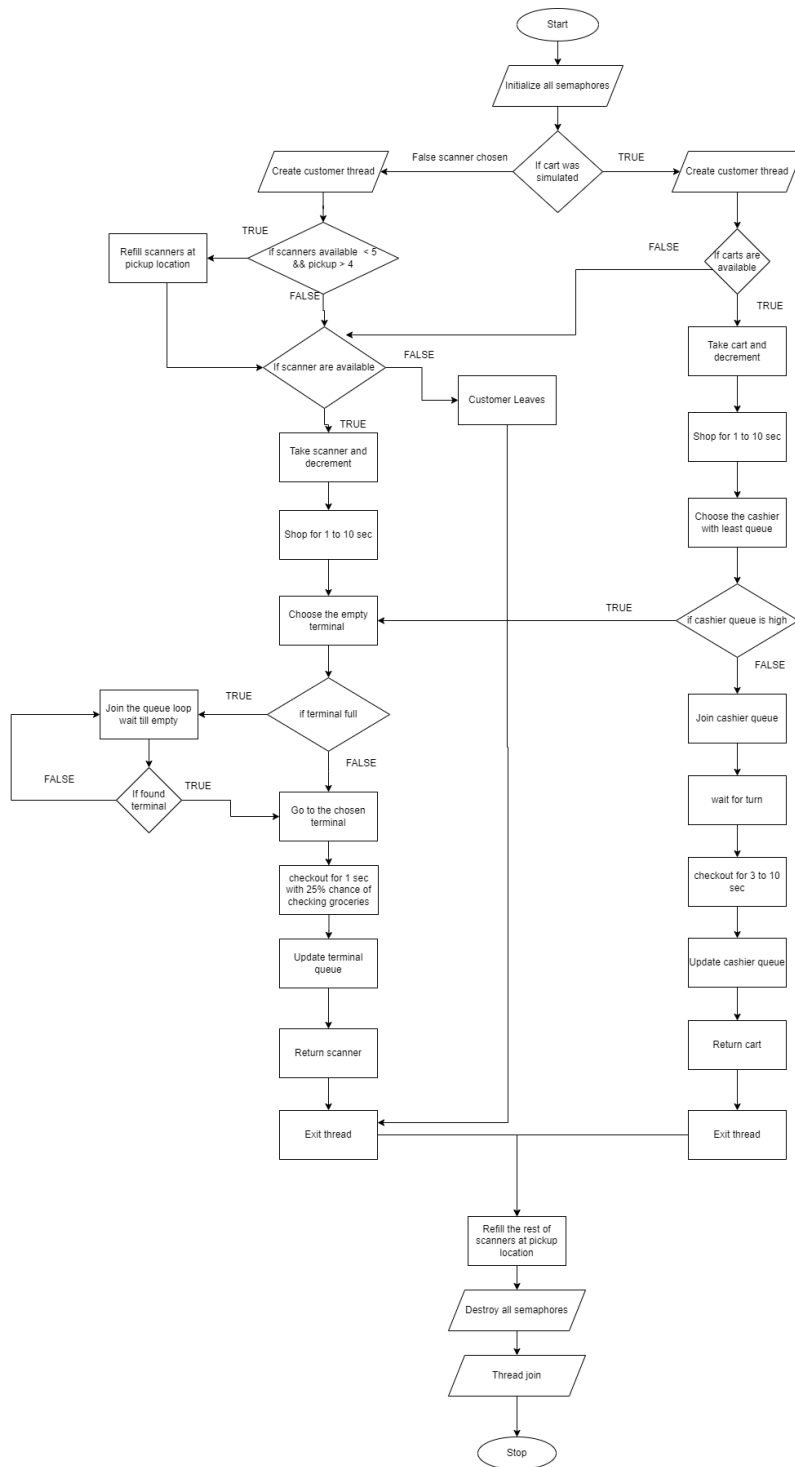


Figure 10: Flowchart  
21