Homework 3 – Ταυτόχρονος Προγραμματισμός

Γαρυφαλλιά Αναστασία Παπαδούλη | 3533

Δημήτριος Τσαλαπάτας | 3246

Νικόλαος Μπέτσος | 3267

Ιωάννης Ρείνος | 3390

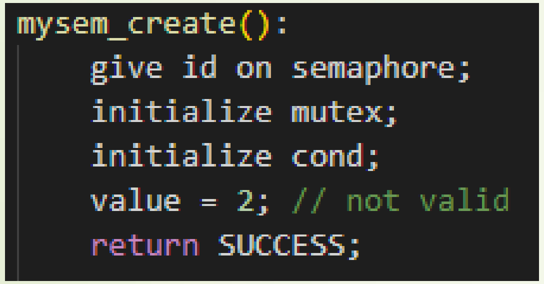
10/01/2024

Ομάδα 4

Assignment 1

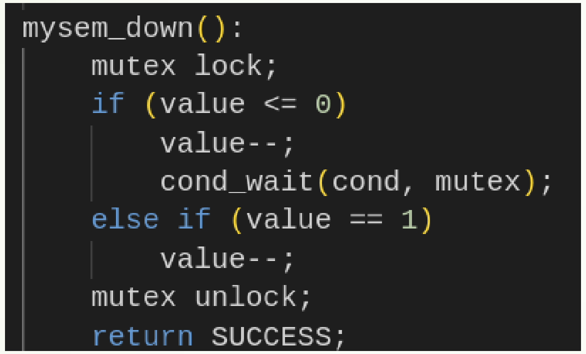
Binary semaphore library using monitors
Tested using hw2_3

- *int mysem_create(mysem_t *s)*



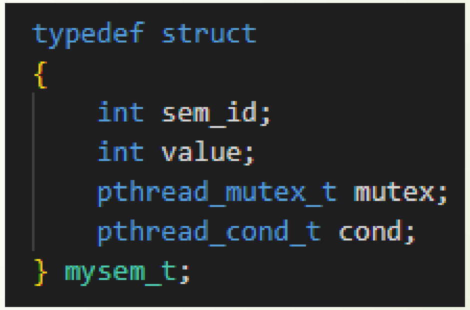
```
mysem_create():
    give id on semaphore;
    initialize mutex;
    initialize cond;
    value = 2; // not valid
    return SUCCESS;
```

- *int mysem_down(mysem_t *s)*



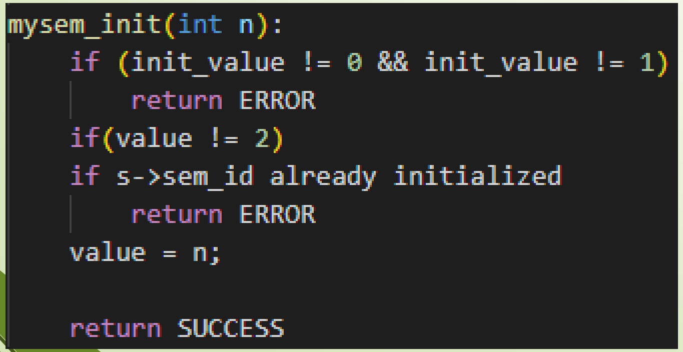
```
mysem_down():
    mutex lock;
    if (value <= 0)
        value--;
        cond_wait(cond, mutex);
    else if (value == 1)
        value--;
    mutex unlock;
    return SUCCESS;
```

- *Struct of each semaphore*



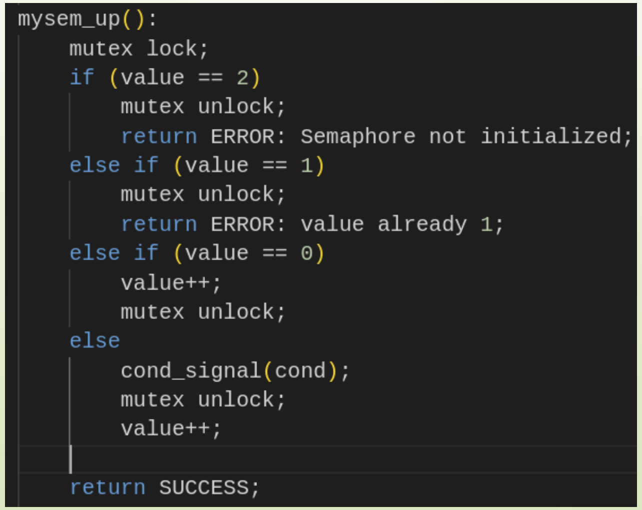
```
typedef struct
{
    int sem_id;
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
} mysem_t;
```

- *int mysem_init(mysem_t *s, int n)*



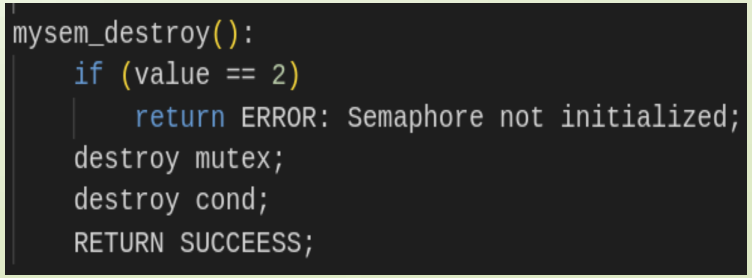
```
mysem_init(int n):
    if (init_value != 0 && init_value != 1)
        return ERROR
    if(value != 2)
    if s->sem_id already initialized
        return ERROR
    value = n;

    return SUCCESS
```

- *int mysem_up(mysem_t *s)*



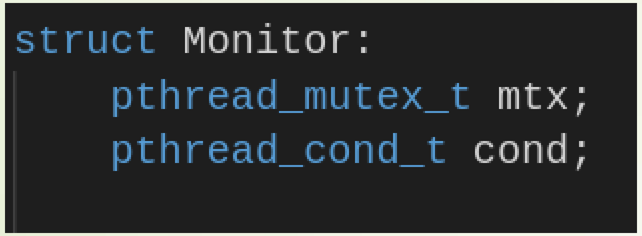
```
mysem_up():
    mutex lock;
    if (value == 2)
        mutex unlock;
        return ERROR: Semaphore not initialized;
    else if (value == 1)
        mutex unlock;
        return ERROR: value already 1;
    else if (value == 0)
        value++;
        mutex unlock;
    else
        cond_signal(cond);
        mutex unlock;
        value++;

    return SUCCESS;
```

- *int mysem_destroy(mysem_t *s)*

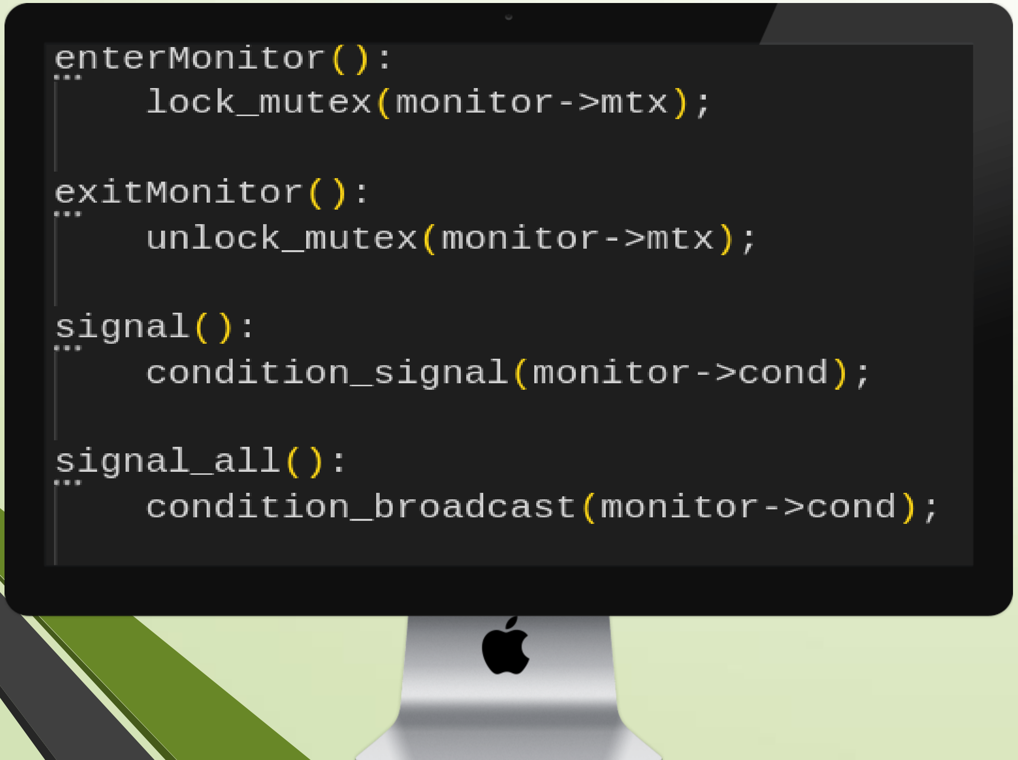


```
mysem_destroy():
    if (value == 2)
        return ERROR: Semaphore not initialized;
    destroy mutex;
    destroy cond;
    RETURN SUCCEESS;
```

Monitor Library

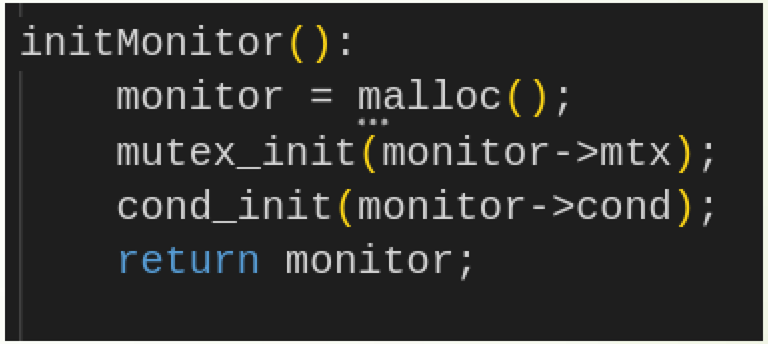


```
struct Monitor:
    pthread_mutex_t mtx;
    pthread_cond_t cond;
```



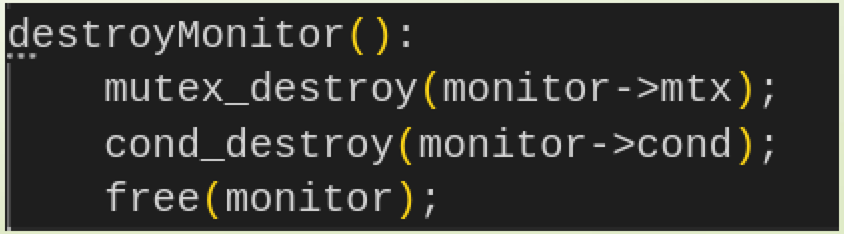
```
enterMonitor():
    lock_mutex(monitor->mtx);

exitMonitor():
    unlock_mutex(monitor->mtx);

signal():
    condition_signal(monitor->cond);

signal_all():
    condition_broadcast(monitor->cond);
```

- *initMonitor(Monitor *monitor)*



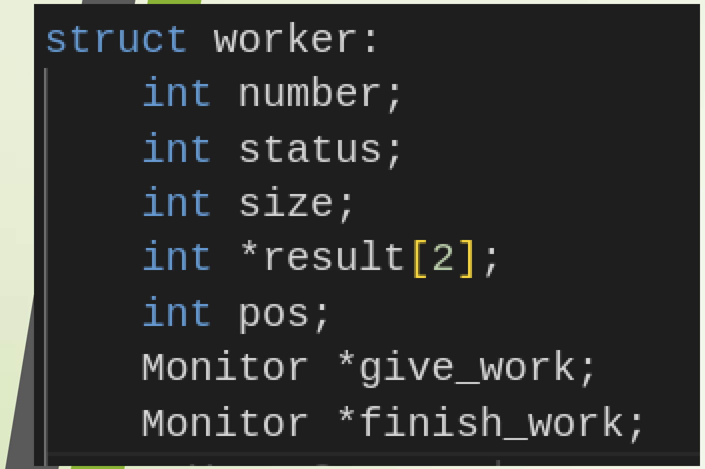
```
initMonitor():
    monitor = malloc();
    mutex_init(monitor->mtx);
    cond_init(monitor->cond);
    return monitor;
```

- *DestroyMonitor(Monitor *monitor)*



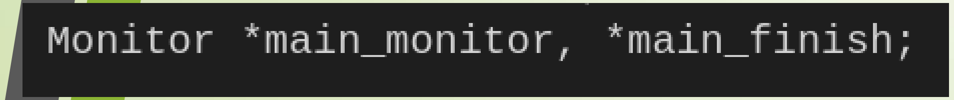
```
destroyMonitor():
    mutex_destroy(monitor->mtx);
    cond_destroy(monitor->cond);
    free(monitor);
```

Assignment 2



```
struct worker:
    int number;
    int status;
    int size;
    int *result[2];
    int pos;
    Monitor *give_work;
    Monitor *finish_work;
```

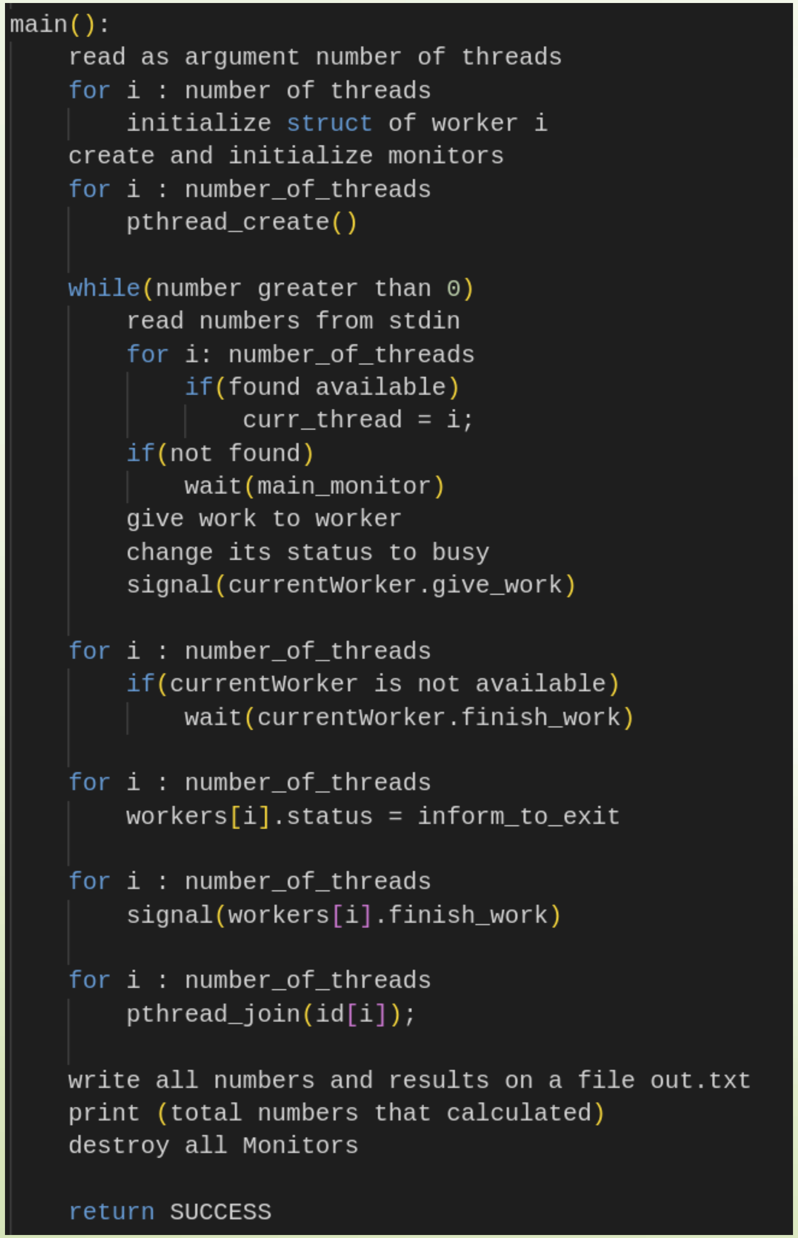
- **Number**: the number that this worker has to process.
- **Status**: shows if the worker is available, busy or terminating
- **Pos**: the position of the worker in the array that is stored
- **Give_work**: monitor that synchronizes the assignment of work to each worker
- **Finish work**: monitor that notifies main that the worker has finished the work
- **Result**: an array that stores the numbers that this worker has calculated and a Flag that shows if the number is prime or not.



```
Monitor *main_monitor, *main_finish;
```

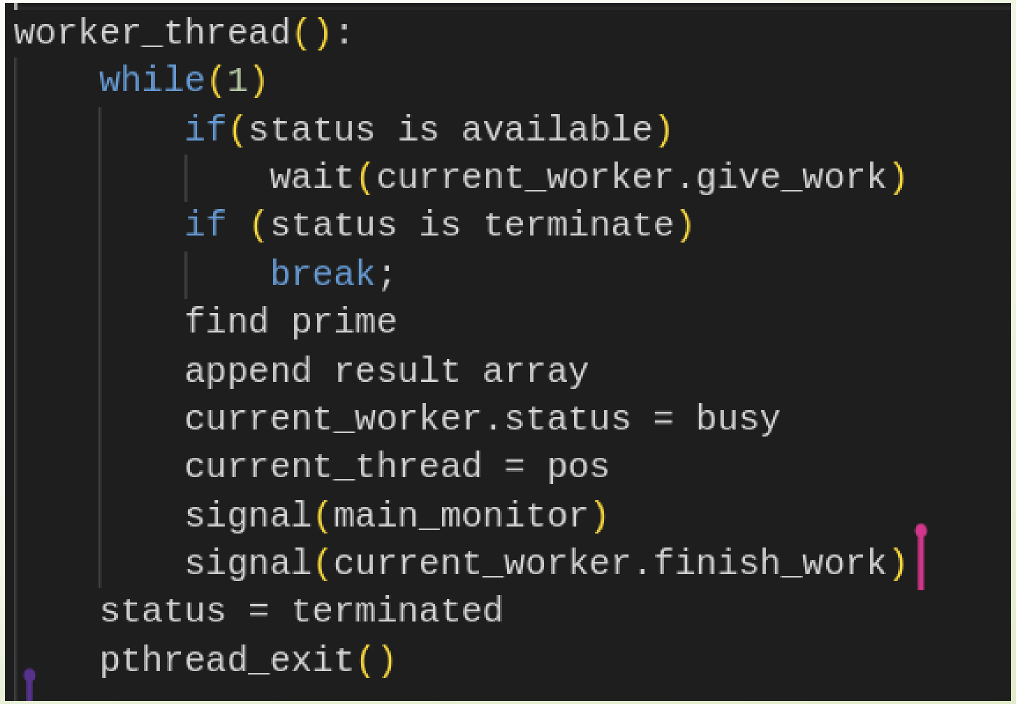
Main_monitor: blocks main when there is no worker available
Main_finish: synchronizes main thread when waiting for the workers to terminate

MAIN



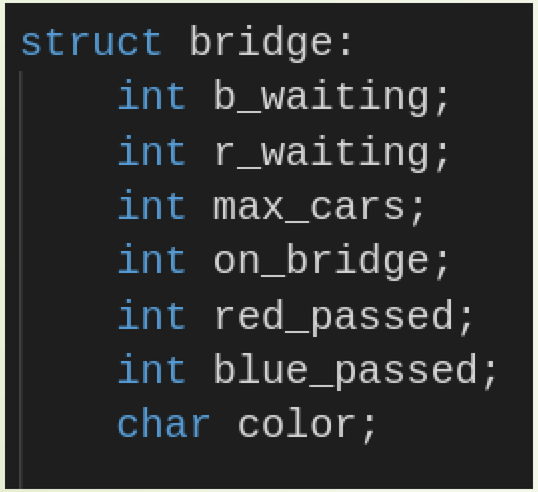
```
main():
    read as argument number of threads
    for i : number of threads
        initialize struct of worker i
    create and initialize monitors
    for i : number_of_threads
        pthread_create()

    while(number greater than 0)
        read numbers from stdin
        for i: number_of_threads
            if(found available)
                curr_thread = i;
        if(not found)
            wait(main_monitor)
        give work to worker
        change its status to busy
        signal(currentWorker.give_work)

    for i : number_of_threads
        if(currentWorker is not available)
            wait(currentWorker.finish_work)

    for i : number_of_threads
        workers[i].status = inform_to_exit

    for i : number_of_threads
        signal(workers[i].finish_work)

    for i : number_of_threads
        pthread_join(id[i]);

    write all numbers and results on a file out.txt
    print (total numbers that calculated)
    destroy all Monitors

    return SUCCESS
```

THREAD



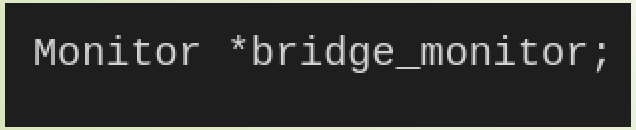
```
worker_thread():
    while(1)
        if(status is available)
            wait(current_worker.give_work)
        if (status is terminate)
            break;
        find prime
        append result array
        current_worker.status = busy
        current_thread = pos
        signal(main_monitor)
        signal(current_worker.finish_work)
    status = terminated
    pthread_exit()
```

Assignment 3



```
struct bridge:
    int b_waiting;
    int r_waiting;
    int max_cars;
    int on_bridge;
    int red_passed;
    int blue_passed;
    char color;
```

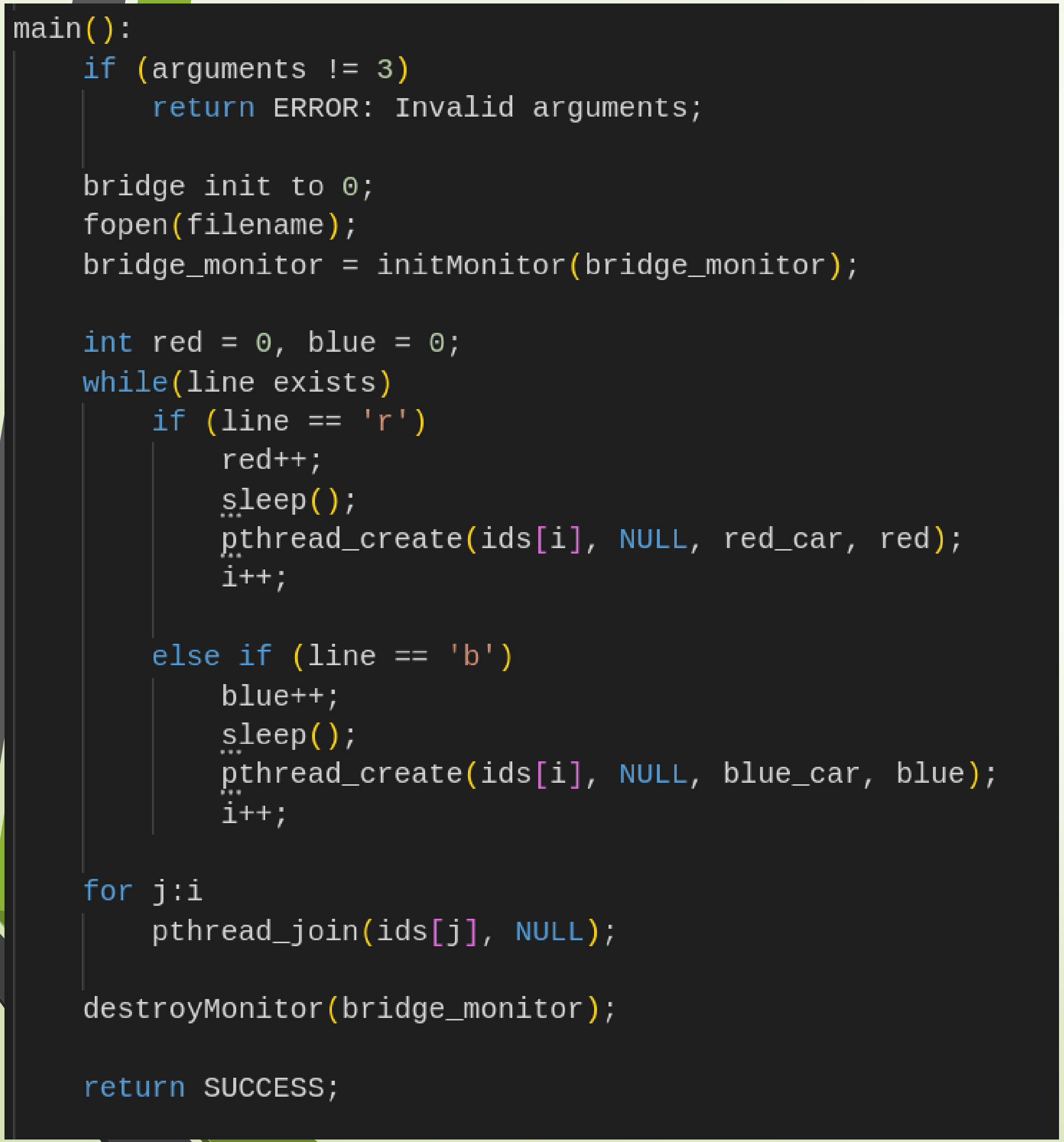
- **b_waiting**: num of blue cars waiting to pass
- **r_waiting**: num of red cars waiting to pass
- **max_cars**: limit of cars that can cross bridge simultaneously
- **on_bridge**: num of cars currently on bridge
- **Red_passed**: num of red cars passed the bridge
- **Blue_passed**: num of blue cars passed the bridge
- **Color**: Color of cars currently in bridge – r , b or \o if empty



```
Monitor *bridge_monitor;
```

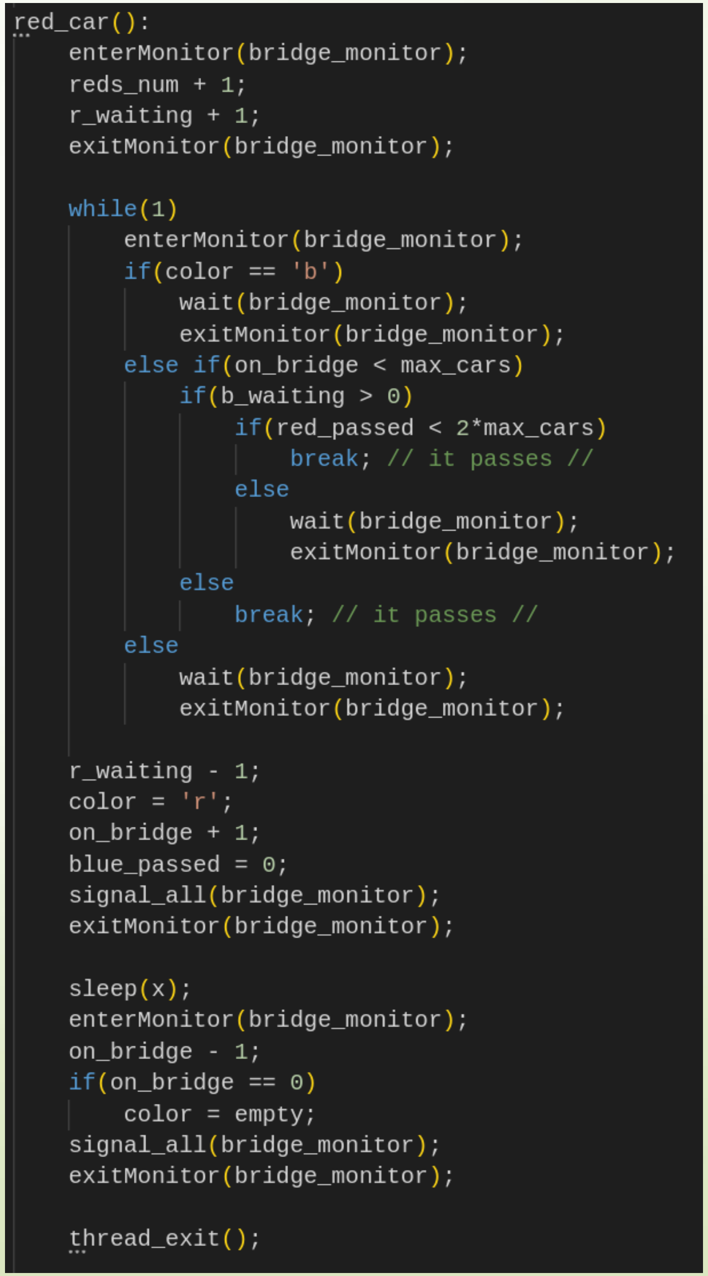
- **BridgeMonitor**: synchronizes which car should pass the bridge

MAIN



```
main():
    if (arguments != 3)
        return ERROR: Invalid arguments;

    bridge init to 0;
    fopen(filename);
    bridge_monitor = initMonitor(bridge_monitor);

    int red = 0, blue = 0;
    while(line exists)
        if (line == 'r')
            red++;
            sleep();
            pthread_create(ids[i], NULL, red_car, red);
            i++;

        else if (line == 'b')
            blue++;
            sleep();
            pthread_create(ids[i], NULL, blue_car, blue);
            i++;

    for j:i
        pthread_join(ids[j], NULL);

    destroyMonitor(bridge_monitor);

    return SUCCESS;
```

Red car



```
red_car():
    enterMonitor(bridge_monitor);
    reds_num + 1;
    r_waiting + 1;
    exitMonitor(bridge_monitor);

    while(1)
        enterMonitor(bridge_monitor);
        if(color == 'b')
            wait(bridge_monitor);
            exitMonitor(bridge_monitor);
        else if(on_bridge < max_cars)
            if(b_waiting > 0)
                if(red_passed < 2*max_cars)
                    break; // it passes //
                else
                    wait(bridge_monitor);
                    exitMonitor(bridge_monitor);
            else
                break; // it passes //
        else
            wait(bridge_monitor);
            exitMonitor(bridge_monitor);

    r_waiting - 1;
    color = 'r';
    on_bridge + 1;
    blue_passed = 0;
    signal_all(bridge_monitor);
    exitMonitor(bridge_monitor);

    sleep(x);
    enterMonitor(bridge_monitor);
    on_bridge - 1;
    if(on_bridge == 0)
        color = empty;
    signal_all(bridge_monitor);
    exitMonitor(bridge_monitor);

    thread_exit();
```

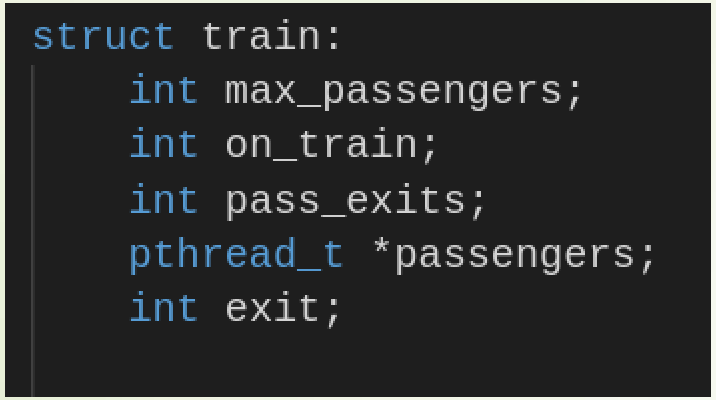
Test cases for hw3_3

- 2_waits: creates a scenario where the time the bridge is empty there are both red and blue cars waiting to pass

- More_reds: creates 20 red cars and 4 blue cars, it checks if red cars will let blue cars pass after a specific number of red_passed

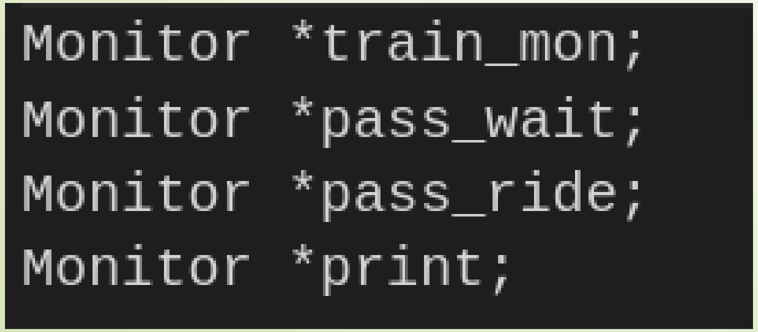
- 3 more tests with random generated tests and random sleep time for further testing

Assignment 4



```
struct train:
    int max_passengers;
    int on_train;
    int pass_exits;
    pthread_t *passengers;
    int exit;
```

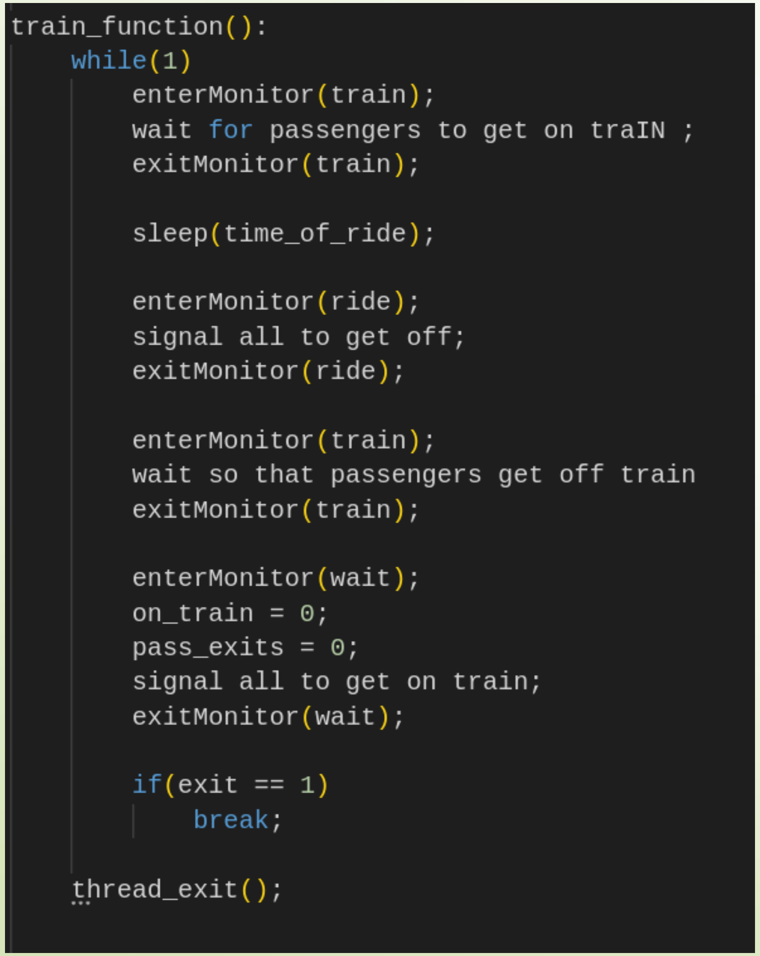
- **Max_passengers**: limit of passengers per ride
- **on_train**: passengers currently on bridge
- **Passengers**: handles of threads corresponding to passengers [for debug purposes]
- **Exit**: signifies no more rides should be performed
- **Pass_exits**: number of passengers that exit the train



```
Monitor *train_mon;
Monitor *pass_wait;
Monitor *pass_ride;
Monitor *print;
```

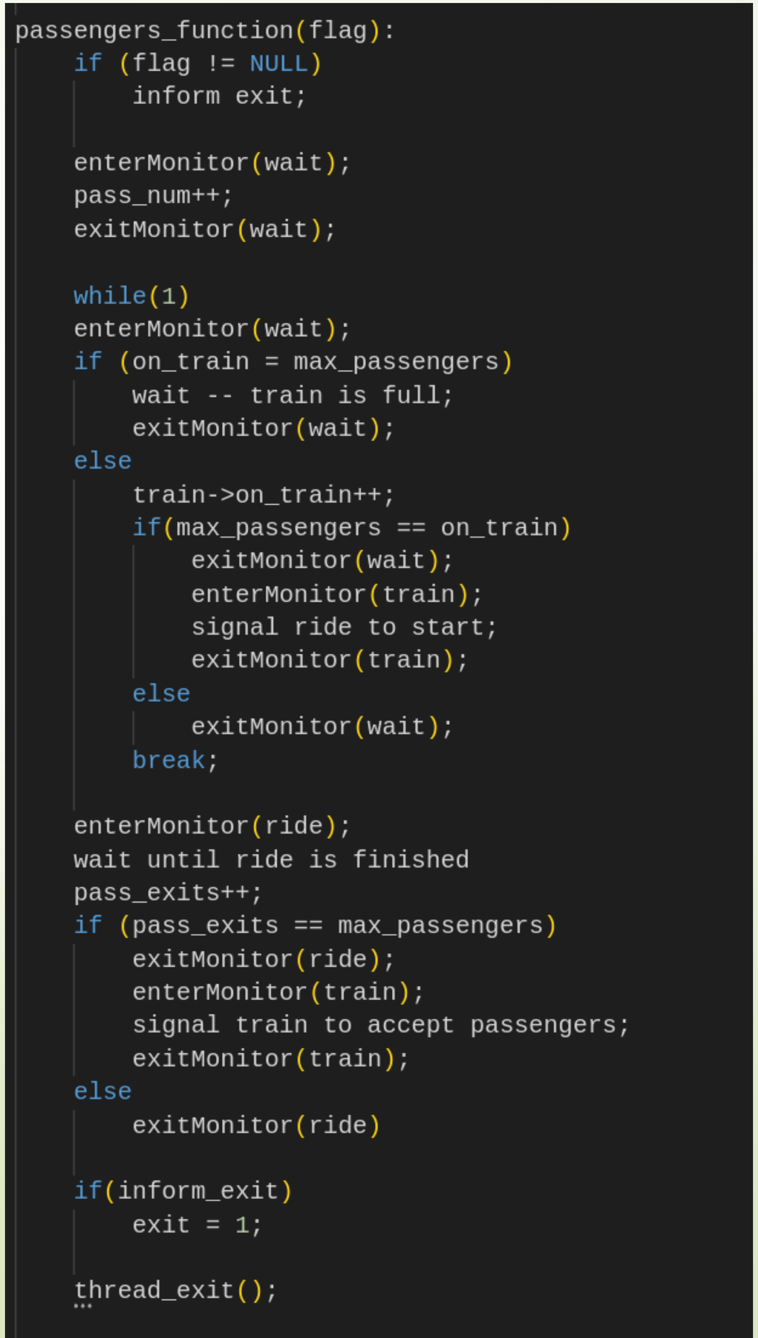
- **Train_mon**: Monitor between train and coming passengers
- **Pass_wait**: blocks waiting passengers outside of train
- **Pass_ride**: blocks passengers already on train
- **Print**: Monitor that synchronize prints between threads

Train function:



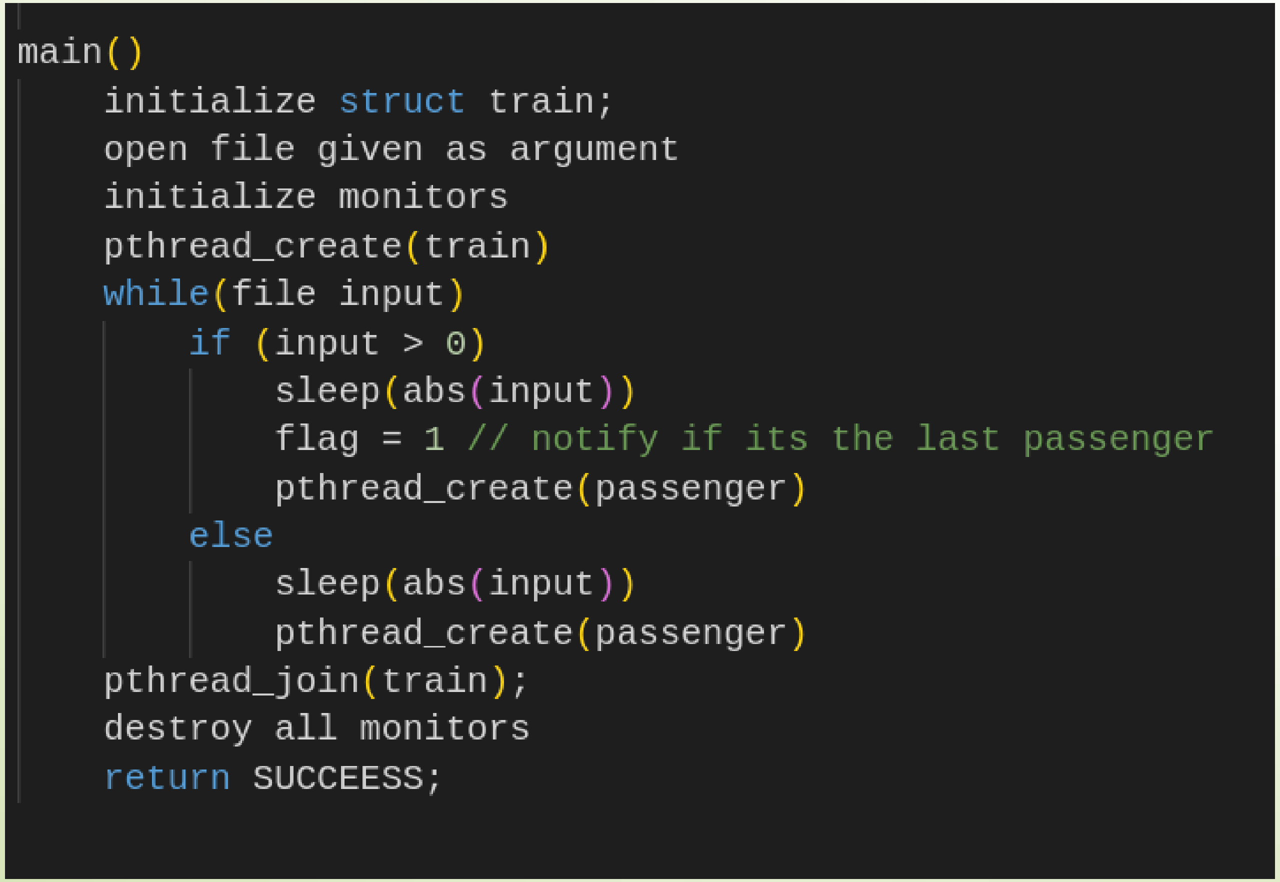
```
train_function():
    while(1)
        enterMonitor(train);
        wait for passengers to get on traIN ;
        exitMonitor(train);

        sleep(time_of_ride);

        enterMonitor(ride);
        signal all to get off;
        exitMonitor(ride);

        enterMonitor(train);
        wait so that passengers get off train
        exitMonitor(train);

        enterMonitor(wait);
        on_train = 0;
        pass_exits = 0;
        signal all to get on train;
        exitMonitor(wait);

        if(exit == 1)
            break;

    thread_exit();
```

Passenger function:



```
passengers_function(flag):
    if (flag != NULL)
        inform exit;

    enterMonitor(wait);
    pass_num++;
    exitMonitor(wait);

    while(1)
    enterMonitor(wait);
    if (on_train = max_passengers)
        wait -- train is full;
        exitMonitor(wait);
    else
        train->on_train++;
        if(max_passengers == on_train)
            exitMonitor(wait);
            enterMonitor(train);
            signal ride to start;
            exitMonitor(train);
        else
            exitMonitor(wait);
        break;

    enterMonitor(ride);
    wait until ride is finished
    pass_exits++;
    if (pass_exits == max_passengers)
        exitMonitor(ride);
        enterMonitor(train);
        signal train to accept passengers;
        exitMonitor(train);
    else
        exitMonitor(ride)

    if(inform_exit)
        exit = 1;

    thread_exit();
```

Main:



```
main()
    initialize struct train;
    open file given as argument
    initialize monitors
    pthread_create(train)
    while(file input)
        if (input > 0)
            sleep(abs(input))
            flag = 1 // notify if its the last passenger
            pthread_create(passenger)
        else
            sleep(abs(input))
            pthread_create(passenger)
    pthread_join(train);
    destroy all monitors
    return SUCCEESS;
```

Thanks for your time!

Team 4