README File for Metro Simulation Project domestOS

Project Overview

This project demonstrates scheduling, synchronization, multi-threading, and deadlock prevention using POSIX threads while simulates a metro system. The aim of this project is preventing deadlocks and accidents in interconnected tunnel lane without breakdowns and overloads.

Working Parts

All parts are working correctly, but "controllog.txt and trainlog.txt" files should be deleted before running the code. Otherwise creates segmentation fault.
Log files successfully record the events and be found as "controllog.txt" and "trainlog.txt".
Inline commenting for all functions.

How to Run

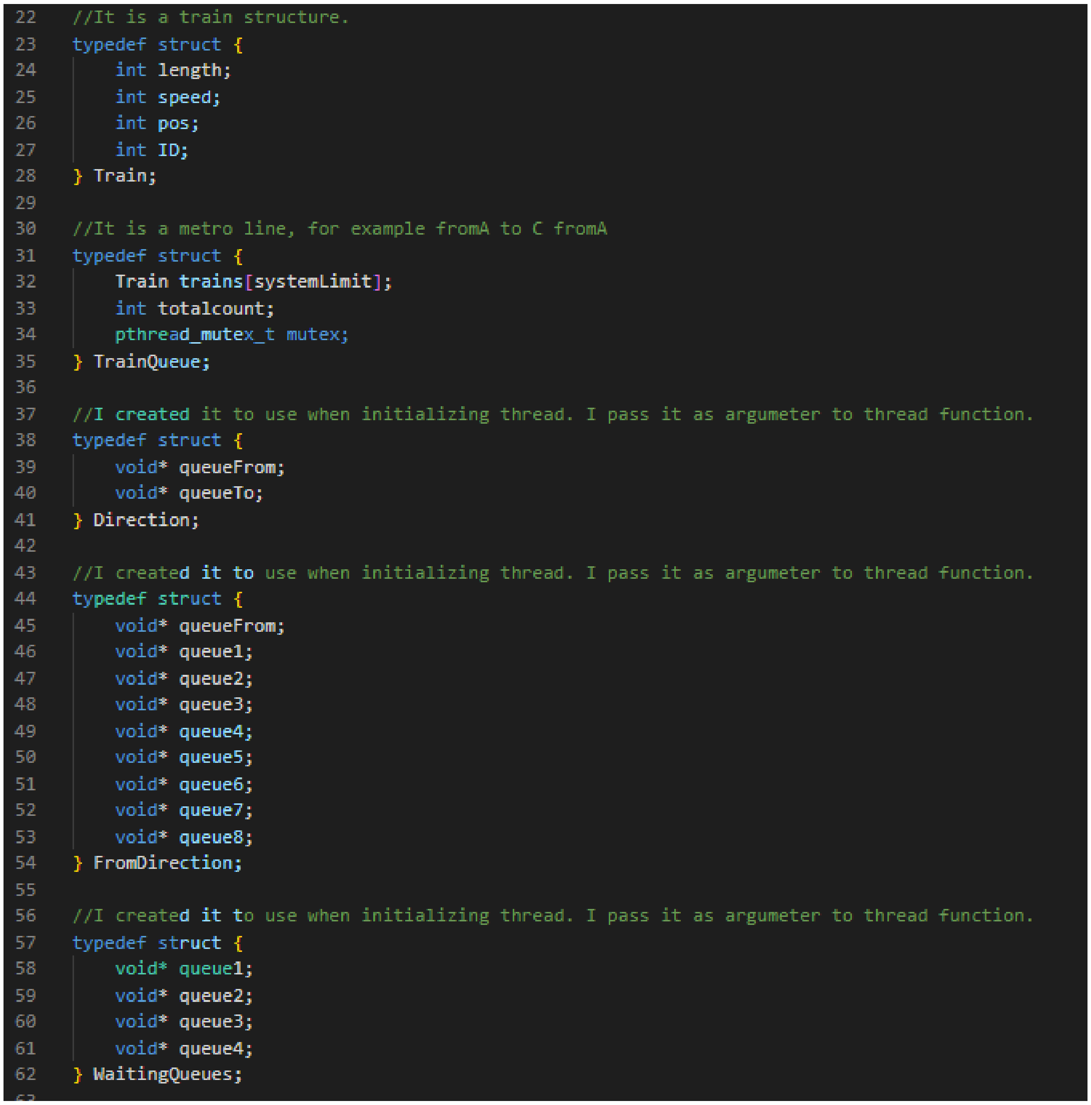
After creating the executable command, add simulation time as first argument and arrival probability as second argument. Example usage:

```
"gcc main.c -o a.out -lpthread"
"./a.out -s 60 0.25"
```

Implementation

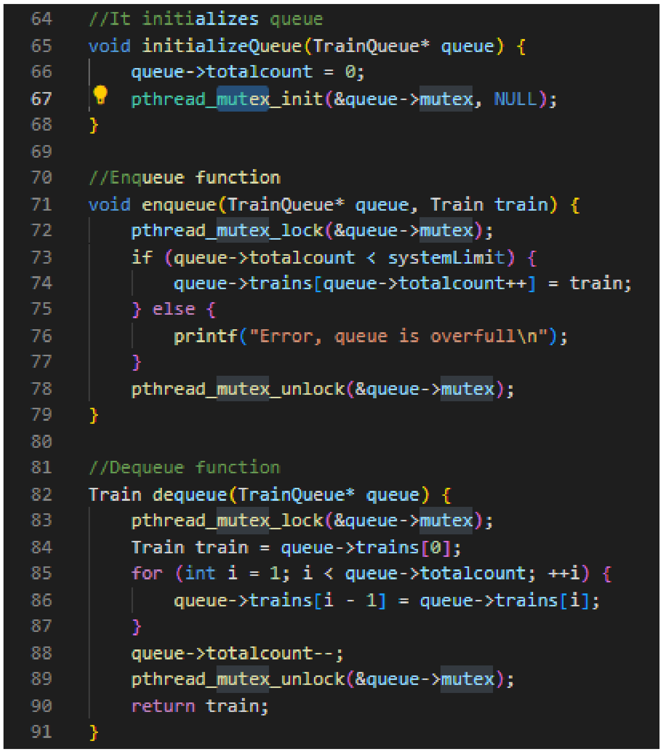
Train and Queue Structures: Defined to represent the trains and the queues at each section.

Direction Structures: To pass multiple arguments to thread functions.



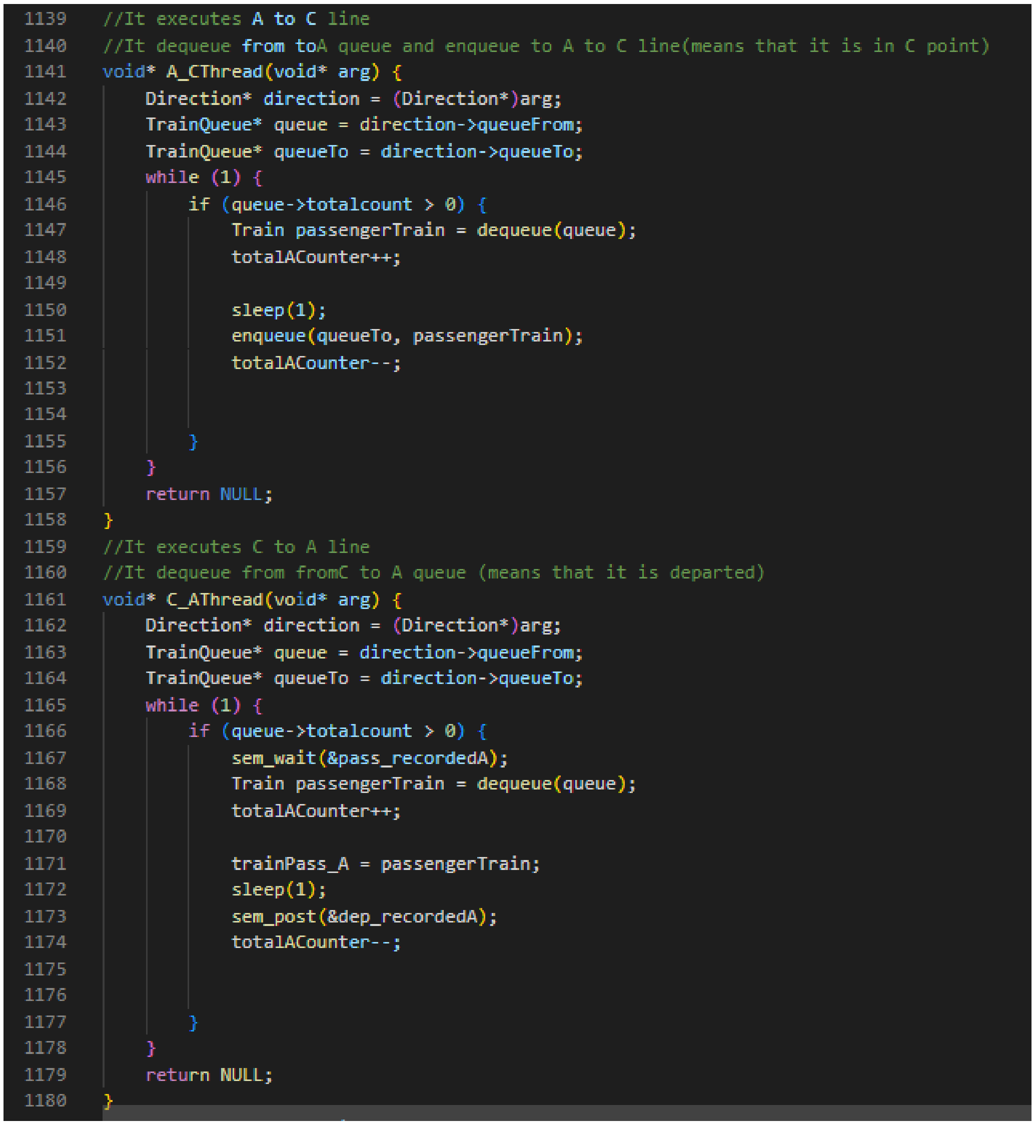
```c
//It is a train structure.
typedef struct {
    int length;
    int speed;
    int pos;
    int ID;
} Train;

//It is a metro line, for example fromA to C fromA
typedef struct {
    Train trains[systemLimit];
    int totalcount;
    pthread_mutex_t mutex;
} TrainQueue;

//I created it to use when initializing thread. I pass it as argumeter to thread function.
typedef struct {
    void* queueFrom;
    void* queueTo;
} Direction;

//I created it to use when initializing thread. I pass it as argumeter to thread function.
typedef struct {
    void* queueFrom;
    void* queue1;
    void* queue2;
    void* queue3;
    void* queue4;
    void* queue5;
    void* queue6;
    void* queue7;
    void* queue8;
} FromDirection;

//I created it to use when initializing thread. I pass it as argumeter to thread function.
typedef struct {
    void* queue1;
    void* queue2;
    void* queue3;
    void* queue4;
} WaitingQueues;
```

Mutexes and Semaphores: Used for synchronization and to prevent race conditions.



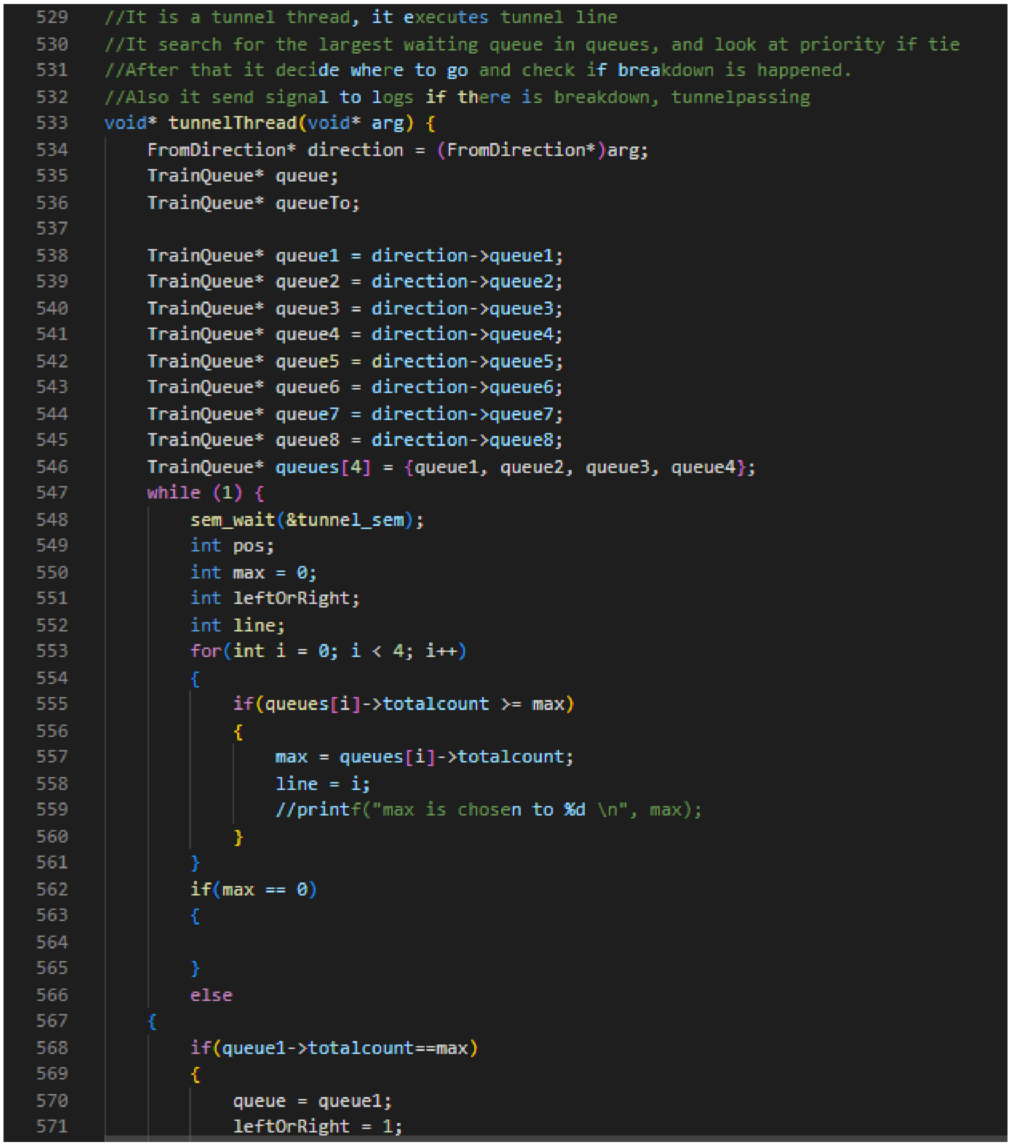
```c
//It initializes queue
void initializeQueue(TrainQueue* queue) {
    queue->totalcount = 0;
    pthread_mutex_init(&queue->mutex, NULL);
}

//Enqueue function
void enqueue(TrainQueue* queue, Train train) {
    pthread_mutex_lock(&queue->mutex);
    if (queue->totalcount < systemLimit) {
        queue->trains[queue->totalcount++] = train;
    } else {
        printf("Error, queue is overfull\n");
    }
    pthread_mutex_unlock(&queue->mutex);
}

//Dequeue function
Train dequeue(TrainQueue* queue) {
    pthread_mutex_lock(&queue->mutex);
    Train train = queue->trains[0];
    for (int i = 1; i < queue->totalcount; ++i) {
        queue->trains[i - 1] = queue->trains[i];
    }
    queue->totalcount--;
    pthread_mutex_unlock(&queue->mutex);
    return train;
}
```

Threads: Different threads represent different sections of the metro system (A-C, B-C, D-E, D-F, etc.).



```
//It executes A to C line
//It dequeue from toA queue and enqueue to A to C line(means that it is in C point)
void* A_CThread(void* arg) {
    Direction* direction = (Direction*)arg;
    TrainQueue* queue = direction->queueFrom;
    TrainQueue* queueTo = direction->queueTo;
    while (1) {
        if (queue->totalcount > 0) {
            Train passengerTrain = dequeue(queue);
            totalACounter++;

            sleep(1);
            enqueue(queueTo, passengerTrain);
            totalACounter--;


        }
    }
    return NULL;
}
//It executes C to A line
//It dequeue from fromC to A queue (means that it is departed)
void* C_AThread(void* arg) {
    Direction* direction = (Direction*)arg;
    TrainQueue* queue = direction->queueFrom;
    TrainQueue* queueTo = direction->queueTo;
    while (1) {
        if (queue->totalcount > 0) {
            sem_wait(&pass_recordedA);
            Train passengerTrain = dequeue(queue);
            totalACounter++;

            trainPass_A = passengerTrain;
            sleep(1);
            sem_post(&dep_recordedA);
            totalACounter--;


        }
    }
    return NULL;
}
```

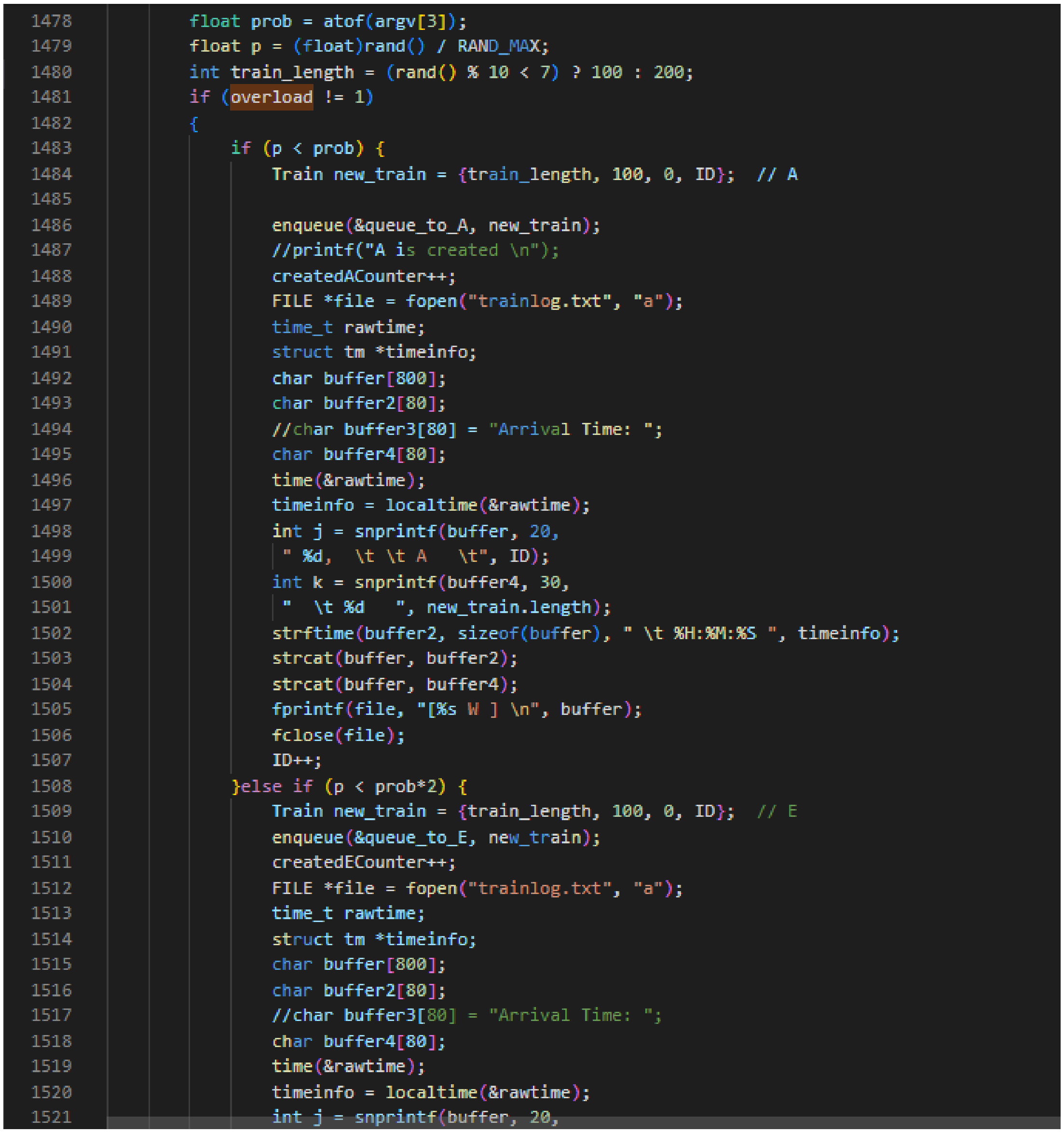
Additional threads are created for the metro control center and various logging purposes.



```
//It is a tunnel thread, it executes tunnel line
//It search for the largest waiting queue in queues, and look at priority if tie
//After that it decide where to go and check if breakdown is happened.
//Also it send signal to logs if there is breakdown, tunnelpassing
void* tunnelThread(void* arg) {
    FromDirection* direction = (FromDirection*)arg;
    TrainQueue* queue;
    TrainQueue* queueTo;

    TrainQueue* queue1 = direction->queue1;
    TrainQueue* queue2 = direction->queue2;
    TrainQueue* queue3 = direction->queue3;
    TrainQueue* queue4 = direction->queue4;
    TrainQueue* queue5 = direction->queue5;
    TrainQueue* queue6 = direction->queue6;
    TrainQueue* queue7 = direction->queue7;
    TrainQueue* queue8 = direction->queue8;
    TrainQueue* queues[4] = {queue1, queue2, queue3, queue4};
    while (1) {
        sem_wait(&tunnel_sem);
        int pos;
        int max = 0;
        int leftOrRight;
        int line;
        for(int i = 0; i < 4; i++)
        {
            if(queues[i]->totalcount >= max)
            {
                max = queues[i]->totalcount;
                line = i;
                //printf("max is chosen to %d \n", max);
            }
        }
        if(max == 0)
        {

        }
        else
    {
        if(queue1->totalcount==max)
        {
            queue = queue1;
            leftOrRight = 1;
```

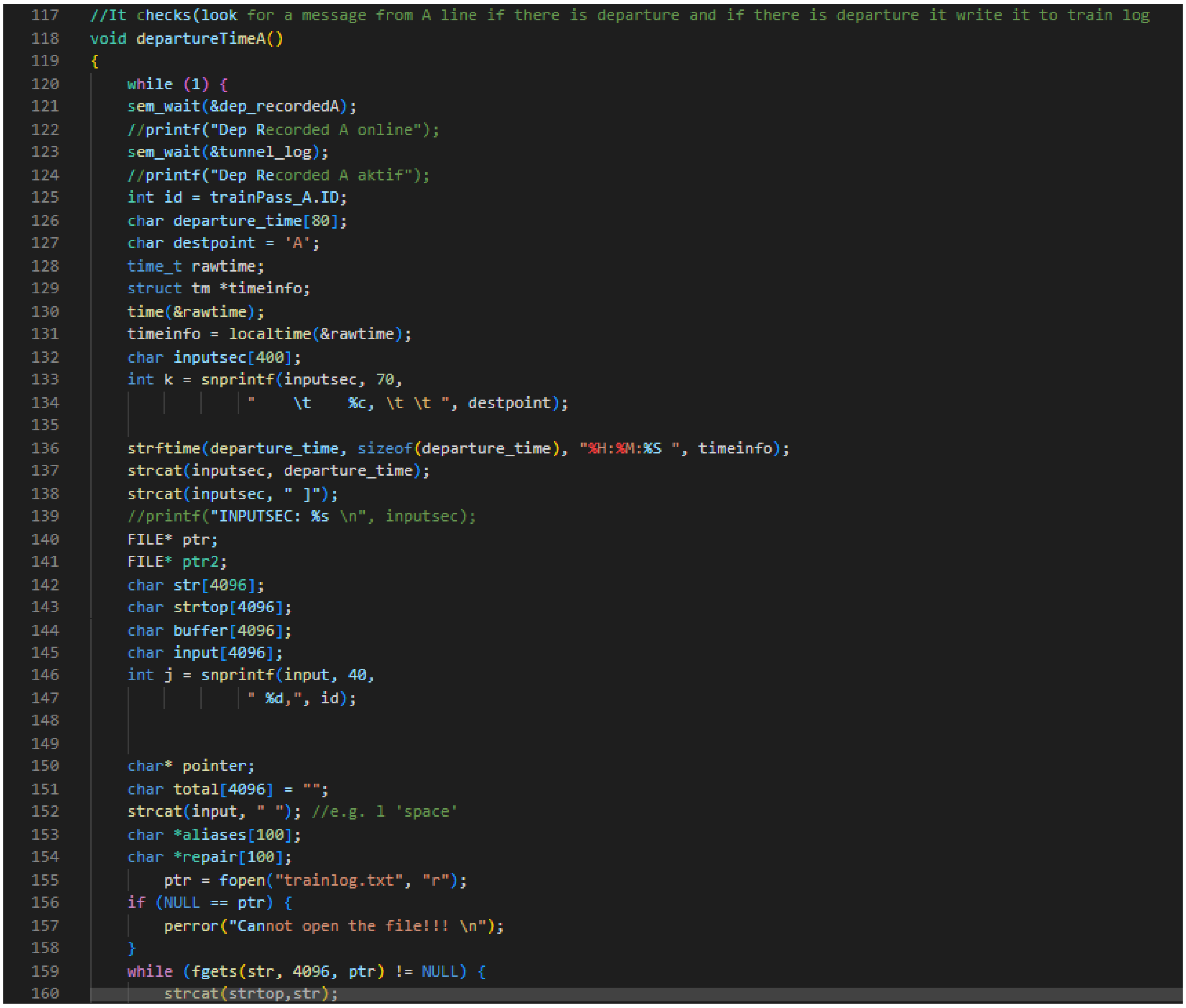
Key Functionalities

Train Movement: Trains move from one section to another, with their journey through the tunnel being the critical section.



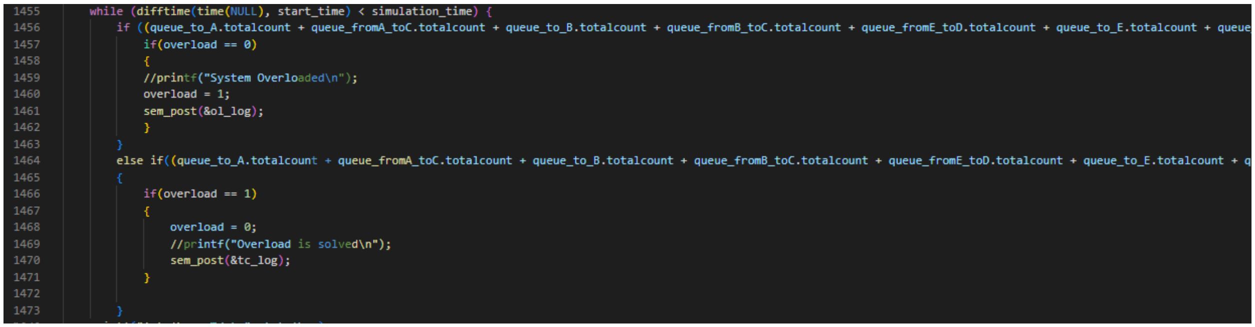
```c
float prob = atof(argv[3]);
float p = (float)rand() / RAND_MAX;
int train_length = (rand() % 10 < 7) ? 100 : 200;
if (overload != 1)
{
    if (p < prob) {
        Train new_train = {train_length, 100, 0, ID};  // A

        enqueue(&queue_to_A, new_train);
        //printf("A is created \n");
        createdACounter++;
        FILE *file = fopen("trainlog.txt", "a");
        time_t rawtime;
        struct tm *timeinfo;
        char buffer[800];
        char buffer2[80];
        //char buffer3[80] = "Arrival Time: ";
        char buffer4[80];
        time(&rawtime);
        timeinfo = localtime(&rawtime);
        int j = snprintf(buffer, 20,
         " %d,  \t \t A   \t", ID);
        int k = snprintf(buffer4, 30,
         "  \t %d   ", new_train.length);
        strftime(buffer2, sizeof(buffer), " \t %H:%M:%S ", timeinfo);
        strcat(buffer, buffer2);
        strcat(buffer, buffer4);
        fprintf(file, "[%s W ] \n", buffer);
        fclose(file);
        ID++;
    }else if (p < prob*2) {
        Train new_train = {train_length, 100, 0, ID};  // E
        enqueue(&queue_to_E, new_train);
        createdECounter++;
        FILE *file = fopen("trainlog.txt", "a");
        time_t rawtime;
        struct tm *timeinfo;
        char buffer[800];
        char buffer2[80];
        //char buffer3[80] = "Arrival Time: ";
        char buffer4[80];
        time(&rawtime);
        timeinfo = localtime(&rawtime);
        int j = snprintf(buffer, 20,
```

Scheduling Algorithm: The metro control center prioritizes trains from the busiest section, with a predefined priority order in case of a tie.

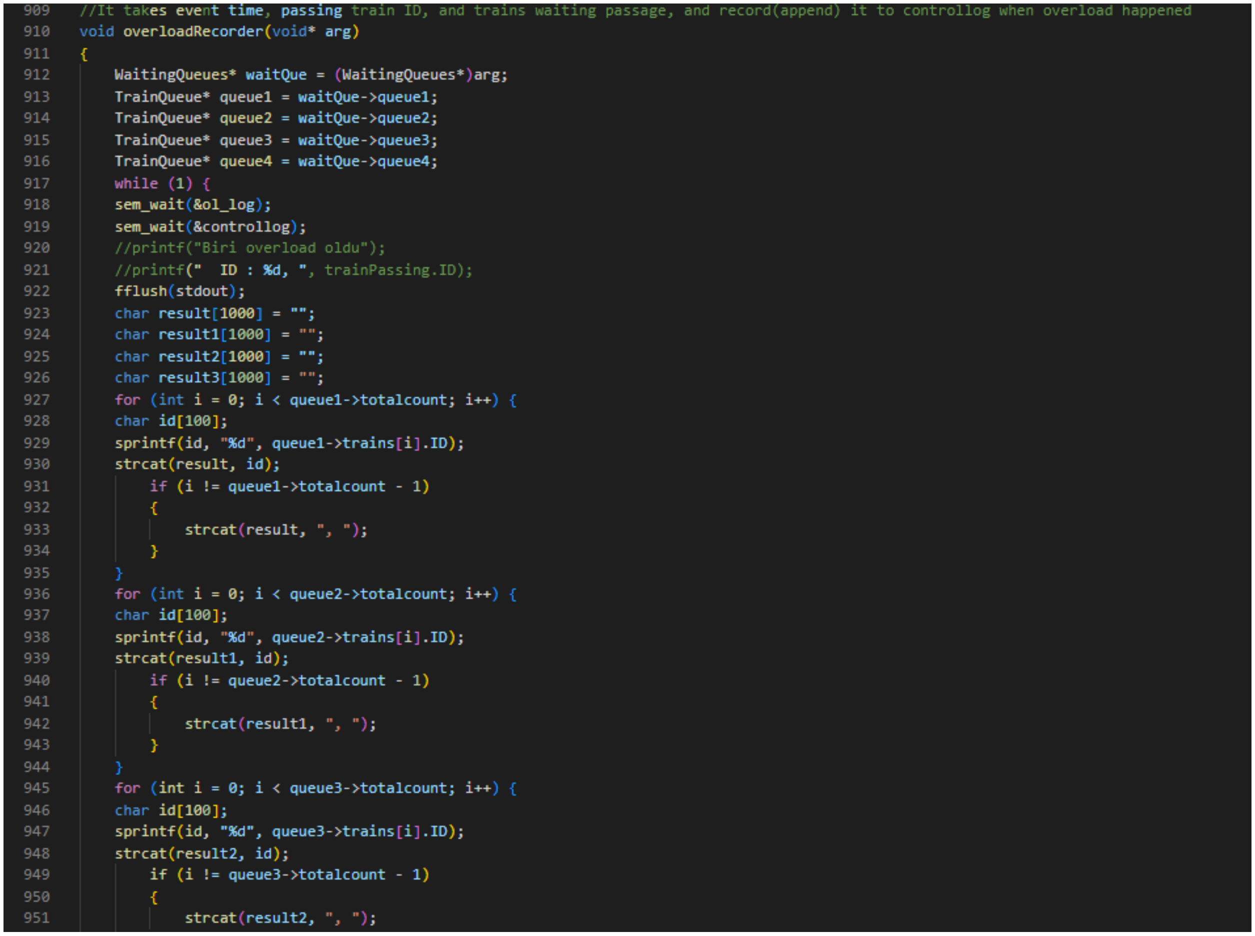


```c
//It checks(look for a message from A line if there is departure and if there is departure it write it to train log
void departureTimeA()
{
    while (1) {
    sem_wait(&dep_recordedA);
    //printf("Dep Recorded A online");
    sem_wait(&tunnel_log);
    //printf("Dep Recorded A aktif");
    int id = trainPass_A.ID;
    char departure_time[80];
    char destpoint = 'A';
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    char inputsec[400];
    int k = snprintf(inputsec, 70,
                    "    \t    %c, \t \t ", destpoint);

    strftime(departure_time, sizeof(departure_time), "%H:%M:%S ", timeinfo);
    strcat(inputsec, departure_time);
    strcat(inputsec, " ]");
    //printf("INPUTSEC: %s \n", inputsec);
    FILE* ptr;
    FILE* ptr2;
    char str[4096];
    char strtop[4096];
    char buffer[4096];
    char input[4096];
    int j = snprintf(input, 40,
                    " %d,", id);


    char* pointer;
    char total[4096] = "";
    strcat(input, " "); //e.g. 1 'space'
    char *aliases[100];
    char *repair[100];
        ptr = fopen("trainlog.txt", "r");
    if (NULL == ptr) {
        perror("Cannot open the file!!! \n");
    }
    while (fgets(str, 4096, ptr) != NULL) {
        strcat(strtop,str);
```

Overload Handling: When the system is overloaded (more than 10 trains outside the tunnel), new train arrivals are halted until the situation normalizes.

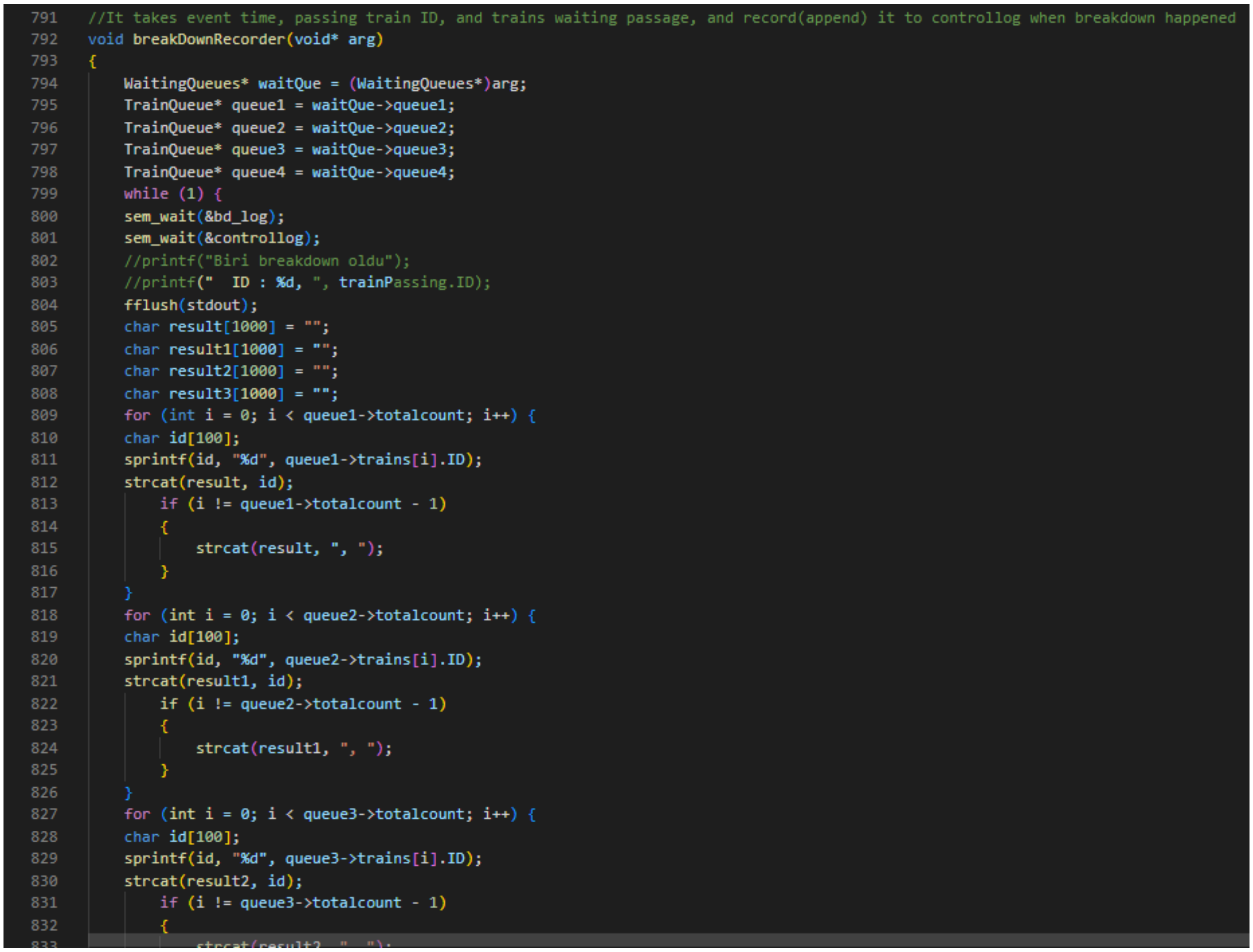


```
    while (difftime(time(NULL), start_time) < simulation_time) {
        if ((queue_to_A.totalcount + queue_fromA_toC.totalcount + queue_to_B.totalcount + queue_fromB_toC.totalcount + queue_fromE_toD.totalcount + queue_to_E.totalcount + queue
            if(overload == 0)
            {
            //printf("System Overloaded\n");
            overload = 1;
            sem_post(&ol_log);
            }
        }
        else if((queue_to_A.totalcount + queue_fromA_toC.totalcount + queue_to_B.totalcount + queue_fromB_toC.totalcount + queue_fromE_toD.totalcount + queue_to_E.totalcount + q
        {
            if(overload == 1)
            {
                overload = 0;
                //printf("Overload is solved\n");
                sem_post(&tc_log);
            }

        }
```



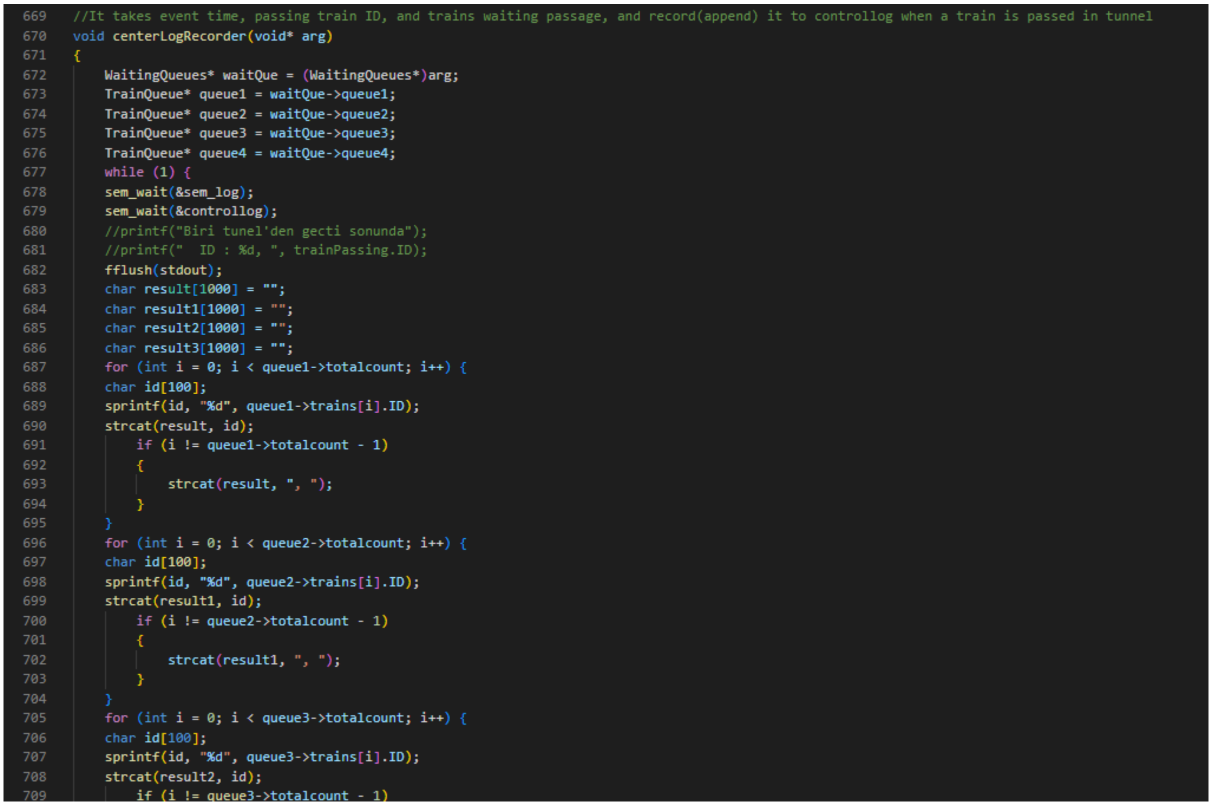
```
//It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when overload happened
void overloadRecorder(void* arg)
{
    WaitingQueues* waitQue = (WaitingQueues*)arg;
    TrainQueue* queue1 = waitQue->queue1;
    TrainQueue* queue2 = waitQue->queue2;
    TrainQueue* queue3 = waitQue->queue3;
    TrainQueue* queue4 = waitQue->queue4;
    while (1) {
    sem_wait(&ol_log);
    sem_wait(&controllog);
    //printf("Biri overload oldu");
    //printf("  ID : %d, ", trainPassing.ID);
    fflush(stdout);
    char result[1000] = "";
    char result1[1000] = "";
    char result2[1000] = "";
    char result3[1000] = "";
    for (int i = 0; i < queue1->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue1->trains[i].ID);
    strcat(result, id);
        if (i != queue1->totalcount - 1)
        {
            strcat(result, ", ");
        }
    }
    for (int i = 0; i < queue2->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue2->trains[i].ID);
    strcat(result1, id);
        if (i != queue2->totalcount - 1)
        {
            strcat(result1, ", ");
        }
    }
    for (int i = 0; i < queue3->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue3->trains[i].ID);
    strcat(result2, id);
        if (i != queue3->totalcount - 1)
        {
            strcat(result2, ", ");
```

Breakdowns in Tunnel: Implemented with a 0.1 probability, causing a delay in tunnel passage.

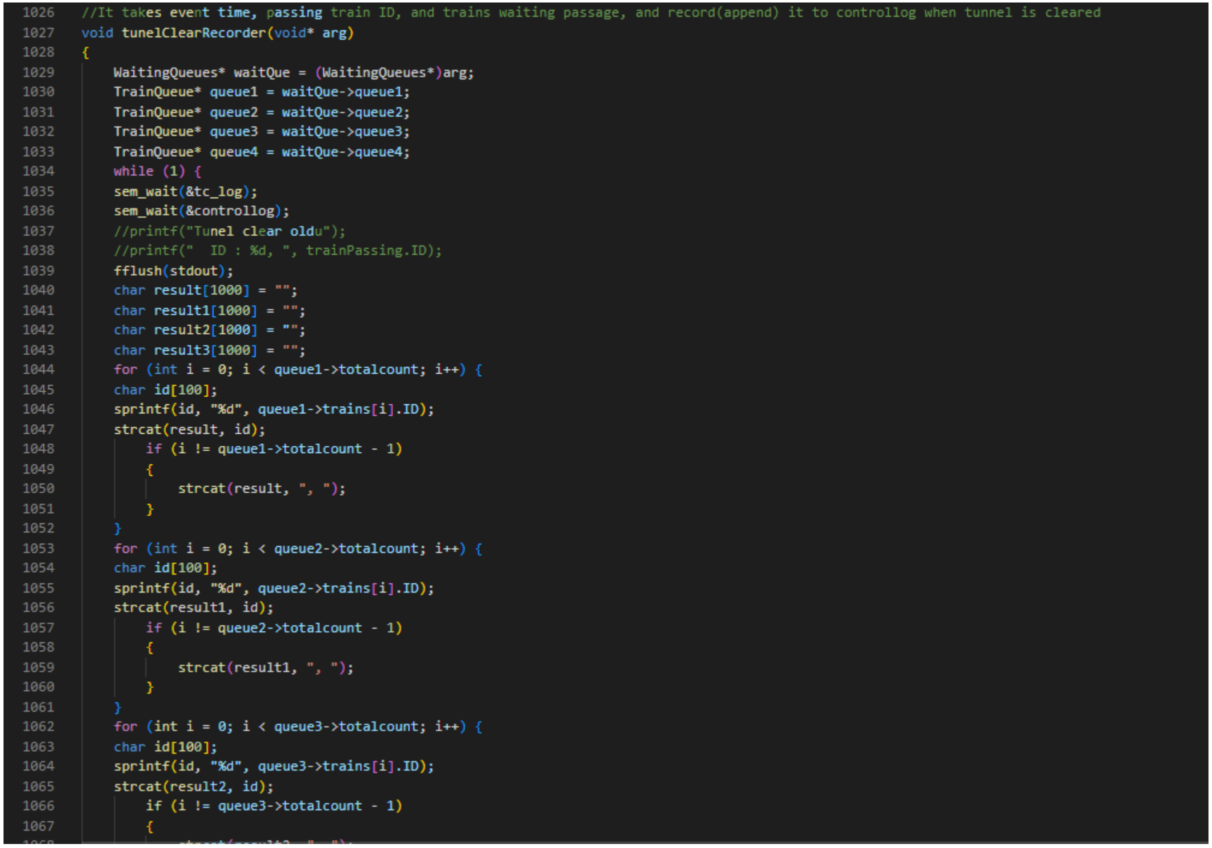


```c
//It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when breakdown happened
void breakDownRecorder(void* arg)
{
    WaitingQueues* waitQue = (WaitingQueues*)arg;
    TrainQueue* queue1 = waitQue->queue1;
    TrainQueue* queue2 = waitQue->queue2;
    TrainQueue* queue3 = waitQue->queue3;
    TrainQueue* queue4 = waitQue->queue4;
    while (1) {
    sem_wait(&bd_log);
    sem_wait(&controllog);
    //printf("Biri breakdown oldu");
    //printf("  ID : %d, ", trainPassing.ID);
    fflush(stdout);
    char result[1000] = "";
    char result1[1000] = "";
    char result2[1000] = "";
    char result3[1000] = "";
    for (int i = 0; i < queue1->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue1->trains[i].ID);
    strcat(result, id);
        if (i != queue1->totalcount - 1)
        {
            strcat(result, ", ");
        }
    }
    for (int i = 0; i < queue2->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue2->trains[i].ID);
    strcat(result1, id);
        if (i != queue2->totalcount - 1)
        {
            strcat(result1, ", ");
        }
    }
    for (int i = 0; i < queue3->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue3->trains[i].ID);
    strcat(result2, id);
        if (i != queue3->totalcount - 1)
        {
            strcat(result2, ", ");
```

Logging: “trainlog.txt” records the train information, destination, and arrival/departure time. “controllog.txt” record detailed information about train movements and system information.



```
//It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when a train is passed in tunnel
void centerLogRecorder(void* arg)
{
    WaitingQueues* waitQue = (WaitingQueues*)arg;
    TrainQueue* queue1 = waitQue->queue1;
    TrainQueue* queue2 = waitQue->queue2;
    TrainQueue* queue3 = waitQue->queue3;
    TrainQueue* queue4 = waitQue->queue4;
    while (1) {
    sem_wait(&sem_log);
    sem_wait(&controllog);
    //printf("Biri tunel'den gecti sonunda");
    //printf("  ID : %d, ", trainPassing.ID);
    fflush(stdout);
    char result[1000] = "";
    char result1[1000] = "";
    char result2[1000] = "";
    char result3[1000] = "";
    for (int i = 0; i < queue1->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue1->trains[i].ID);
    strcat(result, id);
        if (i != queue1->totalcount - 1)
        {
            strcat(result, ", ");
        }
    }
    for (int i = 0; i < queue2->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue2->trains[i].ID);
    strcat(result1, id);
        if (i != queue2->totalcount - 1)
        {
            strcat(result1, ", ");
        }
    }
    for (int i = 0; i < queue3->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue3->trains[i].ID);
    strcat(result2, id);
        if (i != queue3->totalcount - 1)
```



```
//It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when tunnel is cleared
void tunelClearRecorder(void* arg)
{
    WaitingQueues* waitQue = (WaitingQueues*)arg;
    TrainQueue* queue1 = waitQue->queue1;
    TrainQueue* queue2 = waitQue->queue2;
    TrainQueue* queue3 = waitQue->queue3;
    TrainQueue* queue4 = waitQue->queue4;
    while (1) {
    sem_wait(&tc_log);
    sem_wait(&controllog);
    //printf("Tunel clear oldu");
    //printf("  ID : %d, ", trainPassing.ID);
    fflush(stdout);
    char result[1000] = "";
    char result1[1000] = "";
    char result2[1000] = "";
    char result3[1000] = "";
    for (int i = 0; i < queue1->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue1->trains[i].ID);
    strcat(result, id);
        if (i != queue1->totalcount - 1)
        {
            strcat(result, ", ");
        }
    }
    for (int i = 0; i < queue2->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue2->trains[i].ID);
    strcat(result1, id);
        if (i != queue2->totalcount - 1)
        {
            strcat(result1, ", ");
        }
    }
    for (int i = 0; i < queue3->totalcount; i++) {
    char id[100];
    sprintf(id, "%d", queue3->trains[i].ID);
    strcat(result2, id);
        if (i != queue3->totalcount - 1)
        {
```