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

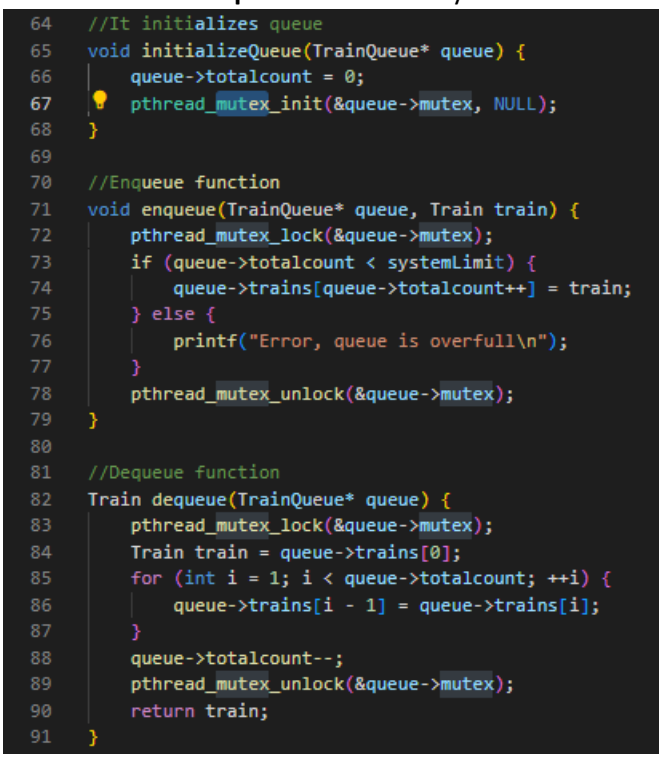
After creating the executable command, add simulation time as first argument and arrival probability as second argument. Example usage:
"gcc main.c -o a.out -lpthread"
"./a.out -s 60 0.25"

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.



```
64  //It initializes queue
65  void initializeQueue(TrainQueue* queue) {
66      queue->totalcount = 0;
67      pthread_mutex_init(&queue->mutex, NULL);
68  }
69
70  //Enqueue function
71  void enqueue(TrainQueue* queue, Train train) {
72      pthread_mutex_lock(&queue->mutex);
73      if (queue->totalcount < systemLimit) {
74          queue->trains[queue->totalcount++] = train;
75      } else {
76          printf("Error, queue is overfull\n");
77      }
78      pthread_mutex_unlock(&queue->mutex);
79  }
80
81  //Dequeue function
82  Train dequeue(TrainQueue* queue) {
83      pthread_mutex_lock(&queue->mutex);
84      Train train = queue->trains[0];
85      for (int i = 1; i < queue->totalcount; ++i) {
86          queue->trains[i - 1] = queue->trains[i];
87      }
88      queue->totalcount--;
89      pthread_mutex_unlock(&queue->mutex);
90      return train;
91  }
```

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

```
1139  //It executes A to C line
1140  //It dequeue from toA queue and enqueue to A to C line(means that it is in C point)
1141  void* A_CThread(void* arg) {
1142      Direction* direction = (Direction*)arg;
1143      TrainQueue* queue = direction->queueFrom;
1144      TrainQueue* queueTo = direction->queueTo;
1145      while (1) {
1146          if (queue->totalcount > 0) {
1147              Train passengerTrain = dequeue(queue);
1148              totalACounter++;
1149
1150              sleep(1);
1151              enqueue(queueTo, passengerTrain);
1152              totalACounter--;
1153
1154
1155          }
1156      }
1157      return NULL;
1158  }
1159  //It executes C to A line
1160  //It dequeue from fromC to A queue (means that it is departed)
1161  void* C_AThread(void* arg) {
1162      Direction* direction = (Direction*)arg;
1163      TrainQueue* queue = direction->queueFrom;
1164      TrainQueue* queueTo = direction->queueTo;
1165      while (1) {
1166          if (queue->totalcount > 0) {
1167              sem_wait(&pass_recordedA);
1168              Train passengerTrain = dequeue(queue);
1169              totalACounter++;
1170
1171              trainPass_A = passengerTrain;
1172              sleep(1);
1173              sem_post(&dep_recordedA);
1174              totalACounter--;
1175
1176
1177          }
1178      }
1179      return NULL;
1180  }
```

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.

```
1478        float prob = atof(argv[3]);
1479        float p = (float)rand() / RAND_MAX;
1480        int train_length = (rand() % 10 < 7) ? 100 : 200;
1481        if (overload != 1)
1482        {
1483            if (p < prob) {
1484                Train new_train = {train_length, 100, 0, ID};  // A
1485
1486                enqueue(&queue_to_A, new_train);
1487                //printf("A is created \n");
1488                createdACounter++;
1489                FILE *file = fopen("trainlog.txt", "a");
1490                time_t rawtime;
1491                struct tm *timeinfo;
1492                char buffer[800];
1493                char buffer2[80];
1494                //char buffer3[80] = "Arrival Time: ";
1495                char buffer4[80];
1496                time(&rawtime);
1497                timeinfo = localtime(&rawtime);
1498                int j = snprintf(buffer, 20,
1499                 " %d,  \t \t A   \t", ID);
1500                int k = snprintf(buffer4, 30,
1501                 "  \t %d   ", new_train.length);
1502                strftime(buffer2, sizeof(buffer), " \t %H:%M:%S ", timeinfo);
1503                strcat(buffer, buffer2);
1504                strcat(buffer, buffer4);
1505                fprintf(file, "[%s W ] \n", buffer);
1506                fclose(file);
1507                ID++;
1508            }else if (p < prob*2) {
1509                Train new_train = {train_length, 100, 0, ID};  // E
1510                enqueue(&queue_to_E, new_train);
1511                createdECounter++;
1512                FILE *file = fopen("trainlog.txt", "a");
1513                time_t rawtime;
1514                struct tm *timeinfo;
1515                char buffer[800];
1516                char buffer2[80];
1517                //char buffer3[80] = "Arrival Time: ";
1518                char buffer4[80];
1519                time(&rawtime);
1520                timeinfo = localtime(&rawtime);
1521                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.

```
117  //It checks(look for a message from A line if there is departure and if there is departure it write it to train log
118  void departureTimeA()
119  {
120      while (1) {
121      sem_wait(&dep_recordedA);
122      //printf("Dep Recorded A online");
123      sem_wait(&tunnel_log);
124      //printf("Dep Recorded A aktif");
125      int id = trainPass_A.ID;
126      char departure_time[80];
127      char destpoint = 'A';
128      time_t rawtime;
129      struct tm *timeinfo;
130      time(&rawtime);
131      timeinfo = localtime(&rawtime);
132      char inputsec[400];
133      int k = snprintf(inputsec, 70,
134                       "    \t    %c, \t \t ", destpoint);
135
136      strftime(departure_time, sizeof(departure_time), "%H:%M:%S ", timeinfo);
137      strcat(inputsec, departure_time);
138      strcat(inputsec, " ]");
139      //printf("INPUTSEC: %s \n", inputsec);
140      FILE* ptr;
141      FILE* ptr2;
142      char str[4096];
143      char strtop[4096];
144      char buffer[4096];
145      char input[4096];
146      int j = snprintf(input, 40,
147                       " %d,", id);
148
149
150      char* pointer;
151      char total[4096] = "";
152      strcat(input, " "); //e.g. 1 'space'
153      char *aliases[100];
154      char *repair[100];
155          ptr = fopen("trainlog.txt", "r");
156      if (NULL == ptr) {
157          perror("Cannot open the file!!! \n");
158      }
159      while (fgets(str, 4096, ptr) != NULL) {
160          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.

```
1455    while (difftime(time(NULL), start_time) < simulation_time) {
1456        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
1457            if(overload == 0)
1458            {
1459            //printf("System Overloaded\n");
1460            overload = 1;
1461            sem_post(&ol_log);
1462            }
1463        }
1464        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
1465        {
1466            if(overload == 1)
1467            {
1468                overload = 0;
1469                //printf("Overload is solved\n");
1470                sem_post(&tc_log);
1471            }
1472
1473        }
```

```
909     //It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when overload happened
910     void overloadRecorder(void* arg)
911     {
912         WaitingQueues* waitQue = (WaitingQueues*)arg;
913         TrainQueue* queue1 = waitQue->queue1;
914         TrainQueue* queue2 = waitQue->queue2;
915         TrainQueue* queue3 = waitQue->queue3;
916         TrainQueue* queue4 = waitQue->queue4;
917         while (1) {
918         sem_wait(&ol_log);
919         sem_wait(&controllog);
920         //printf("Biri overload oldu");
921         //printf("  ID : %d, ", trainPassing.ID);
922         fflush(stdout);
923         char result[1000] = "";
924         char result1[1000] = "";
925         char result2[1000] = "";
926         char result3[1000] = "";
927         for (int i = 0; i < queue1->totalcount; i++) {
928         char id[100];
929         sprintf(id, "%d", queue1->trains[i].ID);
930         strcat(result, id);
931             if (i != queue1->totalcount - 1)
932             {
933                 strcat(result, ", ");
934             }
935         }
936         for (int i = 0; i < queue2->totalcount; i++) {
937         char id[100];
938         sprintf(id, "%d", queue2->trains[i].ID);
939         strcat(result1, id);
940             if (i != queue2->totalcount - 1)
941             {
942                 strcat(result1, ", ");
943             }
944         }
945         for (int i = 0; i < queue3->totalcount; i++) {
946         char id[100];
947         sprintf(id, "%d", queue3->trains[i].ID);
948         strcat(result2, id);
949             if (i != queue3->totalcount - 1)
950             {
951                 strcat(result2, ", ");
```

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

```
791  //It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when breakdown happened
792  void breakDownRecorder(void* arg)
793  {
794      WaitingQueues* waitQue = (WaitingQueues*)arg;
795      TrainQueue* queue1 = waitQue->queue1;
796      TrainQueue* queue2 = waitQue->queue2;
797      TrainQueue* queue3 = waitQue->queue3;
798      TrainQueue* queue4 = waitQue->queue4;
799      while (1) {
800      sem_wait(&bd_log);
801      sem_wait(&controllog);
802      //printf("Biri breakdown oldu");
803      //printf("  ID : %d, ", trainPassing.ID);
804      fflush(stdout);
805      char result[1000] = "";
806      char result1[1000] = "";
807      char result2[1000] = "";
808      char result3[1000] = "";
809      for (int i = 0; i < queue1->totalcount; i++) {
810      char id[100];
811      sprintf(id, "%d", queue1->trains[i].ID);
812      strcat(result, id);
813          if (i != queue1->totalcount - 1)
814          {
815              strcat(result, ", ");
816          }
817      }
818      for (int i = 0; i < queue2->totalcount; i++) {
819      char id[100];
820      sprintf(id, "%d", queue2->trains[i].ID);
821      strcat(result1, id);
822          if (i != queue2->totalcount - 1)
823          {
824              strcat(result1, ", ");
825          }
826      }
827      for (int i = 0; i < queue3->totalcount; i++) {
828      char id[100];
829      sprintf(id, "%d", queue3->trains[i].ID);
830      strcat(result2, id);
831          if (i != queue3->totalcount - 1)
832          {
833              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.

```
669  //It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when a train is passed in tunnel
670  void centerLogRecorder(void* arg)
671  {
672      WaitingQueues* waitQue = (WaitingQueues*)arg;
673      TrainQueue* queue1 = waitQue->queue1;
674      TrainQueue* queue2 = waitQue->queue2;
675      TrainQueue* queue3 = waitQue->queue3;
676      TrainQueue* queue4 = waitQue->queue4;
677      while (1) {
678      sem_wait(&sem_log);
679      sem_wait(&controllog);
680      //printf("Biri tunel'den gecti sonunda");
681      //printf("  ID : %d, ", trainPassing.ID);
682      fflush(stdout);
683      char result[1000] = "";
684      char result1[1000] = "";
685      char result2[1000] = "";
686      char result3[1000] = "";
687      for (int i = 0; i < queue1->totalcount; i++) {
688      char id[100];
689      sprintf(id, "%d", queue1->trains[i].ID);
690      strcat(result, id);
691          if (i != queue1->totalcount - 1)
692          {
693              strcat(result, ", ");
694          }
695      }
696      for (int i = 0; i < queue2->totalcount; i++) {
697      char id[100];
698      sprintf(id, "%d", queue2->trains[i].ID);
699      strcat(result1, id);
700          if (i != queue2->totalcount - 1)
701          {
702              strcat(result1, ", ");
703          }
704      }
705      for (int i = 0; i < queue3->totalcount; i++) {
706      char id[100];
707      sprintf(id, "%d", queue3->trains[i].ID);
708      strcat(result2, id);
709          if (i != queue3->totalcount - 1)
```

```
1026 //It takes event time, passing train ID, and trains waiting passage, and record(append) it to controllog when tunnel is cleared
1027 void tunelClearRecorder(void* arg)
1028 {
1029     WaitingQueues* waitQue = (WaitingQueues*)arg;
1030     TrainQueue* queue1 = waitQue->queue1;
1031     TrainQueue* queue2 = waitQue->queue2;
1032     TrainQueue* queue3 = waitQue->queue3;
1033     TrainQueue* queue4 = waitQue->queue4;
1034     while (1) {
1035     sem_wait(&tc_log);
1036     sem_wait(&controllog);
1037     //printf("Tunel clear oldu");
1038     //printf("  ID : %d, ", trainPassing.ID);
1039     fflush(stdout);
1040     char result[1000] = "";
1041     char result1[1000] = "";
1042     char result2[1000] = "";
1043     char result3[1000] = "";
1044     for (int i = 0; i < queue1->totalcount; i++) {
1045     char id[100];
1046     sprintf(id, "%d", queue1->trains[i].ID);
1047     strcat(result, id);
1048         if (i != queue1->totalcount - 1)
1049         {
1050             strcat(result, ", ");
1051         }
1052     }
1053     for (int i = 0; i < queue2->totalcount; i++) {
1054     char id[100];
1055     sprintf(id, "%d", queue2->trains[i].ID);
1056     strcat(result1, id);
1057         if (i != queue2->totalcount - 1)
1058         {
1059             strcat(result1, ", ");
1060         }
1061     }
1062     for (int i = 0; i < queue3->totalcount; i++) {
1063     char id[100];
1064     sprintf(id, "%d", queue3->trains[i].ID);
1065     strcat(result2, id);
1066         if (i != queue3->totalcount - 1)
1067         {
```