**Operating System Project Report**

Submitted By – Tushar Tak

Registration Number – 11713679

Section – K17CE

Roll Number - 07

Question Allotted – 07

1. **Description**

According to the question provided, IndianRail has decided to improve its efficiency by automating not just its trains but also its passengers. Each passenger and each train are controlled by a thread.

We have to write a synchronization functions that will guarantee orderly loading of trains.

1. **Algorithm**

According to the question

Peterson’s Algorithm((Mutex algorithm)) is used in this code along with the concept of deadlock avoidance

Mutex algorithm  allows two or more processes to share a single-use resource without conflict, using only shared memory for communication.

We can also use concept of semaphore in this code for purpose instead of mutex

Algorithm.

1. **Code Snippet**

![A screenshot of text

Description automatically generated]()

![A screenshot of a social media post

Description automatically generated]() // Leave critical region

pthread\_mutex\_unlock(&(station->lock));

}

![A screenshot of a social media post

Description automatically generated]()

1. **Test Cases**
2. **Code**

#include <pthread.h>

struct station {

int out\_passengers; // in station waiting passengers

int in\_passengers; // in train passengers

pthread\_mutex\_t lock;

pthread\_cond\_t train\_arrived\_cond;

pthread\_cond\_t passengers\_seated\_cond;

pthread\_cond\_t train\_is\_full\_cond;

};

void station\_init(struct station \*station);

void station\_load\_train(struct station \*station, int count);

void station\_wait\_for\_train(struct station \*station);

void station\_on\_board(struct station \*station);

IndianRailways.c

#include <pthread.h>

#include "IndianRailways.h"

/\*\*

Initializes all the mutexes and condition-variables.

\*/

void station\_init(struct station \*station)

{

station->out\_passengers = 0;

station->in\_passengers = 0;

pthread\_mutex\_init(&(station->lock), NULL);

pthread\_cond\_init(&(station->train\_arrived\_cond), NULL);

pthread\_cond\_init(&(station->passengers\_seated\_cond), NULL);

pthread\_cond\_init(&(station->train\_is\_full\_cond), NULL);

}

/\*\*

Loads the train with passengers. When a passenger robot arrives in a station, it first invokes this function.

The function must not return until the train is satisfactorily loaded.

Params:

stattion: current station pointer

count: indicates how many seats are available on the train

\*/

void station\_load\_train(struct station \*station, int count)

{

// Enter critical region

pthread\_mutex\_lock(&(station->lock));

while ((station->out\_passengers > 0) && (count > 0)){

pthread\_cond\_signal(&(station->train\_arrived\_cond));

count--;

pthread\_cond\_wait(&(station->passengers\_seated\_cond), &(station->lock));

}

if (station->in\_passengers > 0)

pthread\_cond\_wait(&(station->train\_is\_full\_cond), &(station->lock));

// Leave critical region

pthread\_mutex\_unlock(&(station->lock));

}

/\*\*

This function must not return until a train is in the station and there are enough free seats on

the train for this passenger. Once this function returns, the passenger robot will move the

passenger on board the train and into a seat.

Once the passenger is seated, it will call the function: station\_on\_board

Params:

stattion: current station pointe

\*/

void station\_wait\_for\_train(struct station \*station)

{

pthread\_mutex\_lock(&(station->lock));

station->out\_passengers++;

pthread\_cond\_wait(&(station->train\_arrived\_cond), &(station->lock));

station->out\_passengers--;

station->in\_passengers++;

pthread\_mutex\_unlock(&(station->lock));

pthread\_cond\_signal(&(station->passengers\_seated\_cond));

}

/\*\*

Use this function to let the train know that itâ€™s on board.

Params:

stattion: current station pointer

\*/

void station\_on\_board(struct station \*station)

{

pthread\_mutex\_lock(&(station->lock));

station->in\_passengers--;

pthread\_mutex\_unlock(&(station->lock));

if (station->in\_passengers == 0)

pthread\_cond\_broadcast(&(station->train\_is\_full\_cond));

}

IndianRailways-runner.c

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <time.h>

#include <unistd.h>

#include <pthread.h>

#include "IndianRailways.h"

// Count of passenger threads that have completed (i.e. station\_wait\_for\_train

// has returned) and are awaiting a station\_on\_board() invocation.

volatile int threads\_completed = 0;

void\*passenger\_thread(void \*arg)

{

struct station \*station = (struct station\*)arg;

station\_wait\_for\_train(station);

\_\_sync\_add\_and\_fetch(&threads\_completed, 1);

return NULL;

}

struct load\_train\_args {

struct station \*station;

int free\_seats;

};

volatile int load\_train\_returned = 0;

void\*load\_train\_thread(void \*args)

{

struct load\_train\_args \*ltargs = (struct load\_train\_args\*)args;

station\_load\_train(ltargs->station, ltargs->free\_seats);

load\_train\_returned = 1;

return NULL;

}

const char\* alarm\_error\_str;

int alarm\_timeout;

void

\_alarm(int seconds, const char \*error\_str)

{

alarm\_timeout = seconds;

alarm\_error\_str = error\_str;

alarm(seconds);

}

void alarm\_handler(int foo)

{

fprintf(stderr, "Error: Failed to complete after %d seconds. Something's "

"wrong, or your system is terribly slow. Possible error hint: [%s]\n",

alarm\_timeout, alarm\_error\_str);

exit(1);

}

#ifndef MIN

#define MIN(\_x,\_y) ((\_x) < (\_y)) ? (\_x) : (\_y)

#endif

/\*

\* This creates a bunch of threads to simulate arriving trains and passengers.

\*/

int main()

{

struct station;

station\_init(&station);

srandom(getpid() ^ time(NULL));

signal(SIGALRM, alarm\_handler);

// Make sure station\_load\_train() returns immediately if no waiting passengers.

\_alarm(1, "station\_load\_train() did not return immediately when no waiting passengers");

station\_load\_train(&station, 0);

station\_load\_train(&station, 10);

\_alarm(0, NULL);

// Create a bunch of 'passengers', each in their own thread.

int i;

const int total\_passengers = 100;

int passengers\_left = total\_passengers;

for (i = 0; i < total\_passengers; i++) {

pthread\_t tid;

int ret = pthread\_create(&tid, NULL, passenger\_thread, &station);

if (ret != 0) {

// If this fails, perhaps we exceeded some system limit.

// Try reducing 'total\_passengers'.

perror("pthread\_create");

exit(1);

}

}

// Make sure station\_load\_train() returns immediately if no free seats.

\_alarm(2, "station\_load\_train() did not return immediately when no free seats");

station\_load\_train(&station, 0);

\_alarm(0, NULL);

// Tons of random tests.

int total\_passengers\_boarded = 0;

const int max\_free\_seats\_per\_train = 50;

int pass = 0;

while (passengers\_left > 0) {

\_alarm(2, "Some more complicated issue appears to have caused passengers "

"not to board when given the opportunity");

int free\_seats = random() % max\_free\_seats\_per\_train;

printf("Train entering station with %d free seats\n", free\_seats);

load\_train\_returned = 0;

struct load\_train\_args args = { &station, free\_seats };

pthread\_t lt\_tid;

int ret = pthread\_create(&lt\_tid, NULL, load\_train\_thread, &args);

if (ret != 0) {

perror("pthread\_create");

exit(1);

}

int threads\_to\_reap = MIN(passengers\_left, free\_seats);

int threads\_reaped = 0;

while (threads\_reaped < threads\_to\_reap) {

if (load\_train\_returned) {

fprintf(stderr, "Error: station\_load\_train returned early!\n");

exit(1);

}

if (threads\_completed > 0) {

if ((pass % 2) == 0)

usleep(random() % 2);

threads\_reaped++;

station\_on\_board(&station);

\_\_sync\_sub\_and\_fetch(&threads\_completed, 1);

}

}

// Wait a little bit longer. Give station\_load\_train() a chance to return

// and ensure that no additional passengers board the train. One second

// should be tons of time, but if you're on a horribly overloaded system,

// this may need to be tweaked.

for (i = 0; i < 1000; i++) {

if (i > 50 && load\_train\_returned)

break;

usleep(1000);

}

if (!load\_train\_returned) {

fprintf(stderr, "Error: station\_load\_train failed to return\n");

exit(1);

}

while (threads\_completed > 0) {

threads\_reaped++;

\_\_sync\_sub\_and\_fetch(&threads\_completed, 1);

}

passengers\_left -= threads\_reaped;

total\_passengers\_boarded += threads\_reaped;

printf("Train departed station with %d new passenger(s) (expected %d)%s\n",

threads\_to\_reap, threads\_reaped,

(threads\_to\_reap != threads\_reaped) ? " \*\*\*\*\*" : "");

if (threads\_to\_reap != threads\_reaped) {

fprintf(stderr, "Error: Too many passengers on this train!\n");

exit(1);

}

pass++;

}

if (total\_passengers\_boarded == total\_passengers) {

printf("Looks good!\n");

return 0;

} else {

// I don't think this is reachable, but just in case.

fprintf(stderr, "Error: expected %d total boarded passengers, but got %d!\n",

total\_passengers, total\_passengers\_boarded);

return 1;

}

}

1. **Complexity**

The Complexity for while loop is 1.

Complexity : Ω (log n).

1. **Git hub**