Adhesh Reghu Kumar
COE18B001

Operating Systems Lab

Lab Assignment - 8

(A)

Dining Philosopher Problem

Logic Used

In this problem, we have to test-drive the semaphore based solution to the Dining Philosopher problem. Let us take a look at some of the important points about the problem.

- There are 5 philosophers and 5 forks in a circular table.
- Each philosopher needs two forks to eat, so a hungry philosopher might have to wait for a neighbor to put down a fork.
- Must prevent deadlock.
- Each philosopher spends some time to eat and then will definitely put down the forks.
- Each philosopher will think for some time and will change state to hungry after some time.

The threads running are:

- 5 philosopher() threads

Let us take the semaphores used

```
// Define sempahores
sem_t mutex;
sem_t S[N];
```

Let us take a look at the shared variables

```
// Define shared variables
int state[N];
int phil_num[N] = {0,1,2,3,4};
```

Some problem macros are

```
#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (phnum + 4)%N
#define RIGHT (phnum + 1)%N
```

Code

```
/****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the solution for the dining philosopher problem using semaphores.*
****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX 100

// Define problem macros
#define N 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2

#define LEFT (phnum + 4)%N
#define RIGHT (phnum + 1)%N

// Define sempahores
sem_t mutex;
sem_t S[N];

// Define shared variables
int state[N];
int phil_num[N] = {0,1,2,3,4};
```

```c
void *philosopher(void *num);
void take_fork(int phnum);
void put_fork(int phnum);
void test(int phnum);

int main()
{
    int i;

    pthread_t thread_id[N];

    // Init the semaphore
    sem_init(&mutex, 0,1);

    for(i=0;i<N;++i)
       sem_init(&S[i],0,0);

    // create and call thread runners
    for(i=0;i<N;++i)
    {
       pthread_create(&thread_id[i], NULL, philosopher, &phil_num[i]);
       printf("Philosopher %d is thinking.\n",i+1);
    }

    for(i=0;i<N;++i)
       pthread_join(thread_id[i],NULL);
}

void *philosopher(void *num)
{
    while(1)
    {
       int *i = num;
       sleep(1);
       take_fork(*i);
       sleep(0);
       put_fork(*i);
    }
}

// Take chopsticks up
void take_fork(int phnum)
```

```
{
    sem_wait(&mutex);

    // move to HUNGRY state
    state[phnum] = HUNGRY;
    printf("Philosopher %d is hungry.\n",phnum);

    // Eait if neighbors arent eating
    test(phnum);
    sem_post(&mutex);
    // If unable to eat wait
    sem_wait(&S[phnum]);
    sleep(1);
}

// Put down chopstick
void put_fork(int phnum)
{
    sem_wait(&mutex);

    // move to state of THINKING
    state[phnum] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",phnum + 1, LEFT + 1,phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

      test(LEFT);
      test(RIGHT);
      sem_post(&mutex);
}

void test(int phnum)
{
      if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
      {
      // state that eating
      state[phnum] = EATING;
      sleep(2);
      printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
      printf("Philosopher %d is Eating\n", phnum + 1);

      sem_post(&S[phnum]);
      }
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ gcc -o out diningPhil.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ ./out
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 5 is thinking.
Philosopher 0 is hungry.
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 1 is hungry.
Philosopher 2 is hungry.
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 3 is hungry.
Philosopher 4 is hungry.
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 0 is hungry.
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is hungry.
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 is hungry.
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 is hungry.
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 is hungry.
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 0 is hungry.
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is hungry.
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
```

Reader Writer Problem

Logic Used

In this problem, we have to test-drive the semaphore based solution to the Reader Writer problem. Let us take a look at some of the important points about the problem.

- Any number of readers can be in the critical section simultaneously.
- Writers must have exclusive access to the critical section.

In other words, a writer cannot enter the critical section while any other thread (reader or writer) is there, and while the writer is there, no other thread may enter.

The threads running are:

- 5 `reader()` threads
- 5 `writer()` threads

The semaphores used in the problem are:

```
// Define semaphores
sem_t mutex;
sem_t writeblock;
```

The `mutex` protects the shared counter and the `writeblock` is 1 if there are no readers accessing the critical section.

The shared variables are:

```
// Define shared variables
int data = 0, rcount = 0;
```

The `data` is the data being read and the `rcount` keeps track of the number of readers in the critical section.

Code

```c
/****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the solution for the reader writer using
semaphores.*
****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX 100

// Define semaphores
sem_t mutex;
sem_t writeblock;

// Define shared variables
int data = 0, rcount = 0;

void *reader(void *arg);
void *writer(void *arg);

int main()
{
    int i;

    pthread_t rtid[5], wtid[5];

    // init the semaphore
    sem_init(&mutex,0,1);
    sem_init(&writeblock,0,1);

    for(i=0;i< 5;++i)
    {
       int *arg = malloc(sizeof(*arg));
       *arg = i;
       pthread_create(&wtid[i], NULL, writer, arg);
       pthread_create(&rtid[i], NULL, reader, arg);
    }
```

```
    for(i=0;i< 5;++i)
    {
       pthread_join(wtid[i],NULL);
       pthread_join(rtid[i],NULL);
    }

    return 0;
}

// Define the reader thread runner
void *reader(void *arg)
{
    int f;
    f = *((int*) arg);

    sem_wait(&mutex);
    rcount = rcount + 1;

    if(rcount == 1)
       sem_wait(&writeblock);

    sem_post(&mutex);
    printf("Data read by reader %d is %d\n",f,data);
    sleep(1);

    sem_wait(&mutex);
    rcount = rcount - 1;

    if(rcount == 0)
       sem_post(&writeblock);

    sem_post(&mutex);
}

// Define writer runner
void *writer(void *arg)
{
    int f;
    f = *((int*) arg);

    sem_wait(&writeblock);
    data++;
```

```
    printf("Data written by writer %d is %d\n",f,data);
    sleep(1);

    sem_post(&writeblock);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ gcc -o out readerWriter.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ ./out
Data written by writer 0 is 1
Data read by reader 0 is 1
Data read by reader 1 is 1
Data read by reader 2 is 1
Data written by writer 1 is 2
Data written by writer 2 is 3
```

(B)

Santa Claus Problem

Logic Used

In this problem, we have to implement the semaphore based solution to the Santa Claus problem. Let us take a look at some of the important points about the problem.

- Initially: Santa Claus is asleep. There are 9 reindeers at vacation and elves at work making toys.
- The elves can wake Santa when only three of them have some difficulty. When three elves are having their problems solved, any other elf should wait for those elves to return.
- If Santa wakes up and finds that the last reindeer has returned, he will give higher priority to setting up the sleigh and distributing gifts, so the elves if any will have to wait.
- Until the last reindeer arrives, the other reindeers will get hitched to the sleigh and wait.

There are three main thread runners in this program.

- 1 Santa - `santaRunner()`
- 9 Reindeer runner threads - `reindeerRunner()`
- `N_ELVES` Elf runner threads - `elfRunner()`

Let us take a look at the share variables and semaphores used.

```
// Define shared variables
int elves = 0;
int reindeer = 0;
// Define semaphores
sem_t santaSem;
sem_t reindeerSem;
sem_t elfTex;
sem_t mutex;
```

```
// Init the semaphore variables
sem_init(&santaSem,0,0);
sem_init(&reindeerSem,0,0);
```

```
sem_init(&elfTex,0,1);
sem_init(&mutex,0,1);
```

Elves and reindeer are counters protected by mutex. Elves and reindeer get mutex to modify the counters; Santa gets it to check them. Santa waits on `santaSem` until either an elf or a reindeer signals him. The reindeer wait on `reindeerSem` until Santa signals them to enter the paddock and get hitched. The elves use `elfTex` to prevent additional elves from entering while three elves are being helped.

Now let us take a look at the Santa thread runner:

```
// Define the santa runner thread
void *santaRunner(void *args)
{
    int i = 0;
    printf("[Santa] Santa is Here !!!\n");
    while(1)
    {
       sem_wait(&santaSem);
       sem_wait(&mutex);

       if(reindeer == 9)
       {
             prepareSleigh();
             for(i=0;i<9;++i)
                   sem_post(&reindeerSem);
             printf("[Santa] Make kids happy\n");
             reindeer = 0;
       }
       else if(elves == 3)
             helpElves();

       sem_post(&mutex);
    }
}
```

When Santa wakes up, he checks which of the two conditions holds and either deals with the reindeer or the waiting elves. If there are nine reindeer waiting, Santa invokes `prepareSleigh()`, then signals `reindeerSem` nine times, allowing the reindeer to invoke

getHitched(). If there are elves waiting, Santa just invokes helpElves(). There is no need for the elves to wait for Santa; once they signal santaSem, they can invoke getHelp() immediately.

Let us take a look at the Elf thread runners:

```
// Define the elf runner threads
void *elfRunner(void *args)
{
    int elf_num = *(int*)args;
    while(1)
    {
       sem_wait(&elfTex);
       sem_wait(&mutex);

       elves++;
       if(elves == 3)
             sem_post(&santaSem);
       else
             sem_post(&elfTex);

       sem_post(&mutex);

       getHelp(elf_num);

       sem_wait(&mutex);
       elves--;
       if(elves == 0)
             sem_post(&elfTex);
       sem_post(&mutex);
    }

    // Work a while
    printf("[Elf] Elf %d is working.\n",elf_num);
    sleep(3);
}
```

The first two elves release elfTex at the same time they release the mutex, but the last elf holds elfTex, barring other elves from entering until all three elves have invoked getHelp(). The last elf to leave releases elfTex, allowing the next batch of elves to enter.

Finally let us take a look at the reindeer runner threads:

```
// Define the reindeer runner thread
void *reindeerRunner(void *args)
{
    int reindeer_num = *(int*)args;
    while(1)
    {
       sem_wait(&mutex);
       reindeer++;
       if(reindeer == 9)
             sem_post(&santaSem);
       sem_post(&mutex);

       sem_wait(&reindeerSem);
       getHitched(reindeer_num);
       sleep(5);
    }
}
```

The ninth reindeer signals Santa and then joins the other reindeer waiting on reindeerSem. When Santa signals, the reindeer all execute getHitched(). The elf code is similar, except that when the third elf arrives it has to bar subsequent arrivals until the first three have executed getHelp().

Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: Implement the Santa Claus Problem using semaphores*
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <semaphore.h>

// Define const parameters
```

```
static const int N_REINDEERS = 9;
static const int N_ELVES = 10;

// Define shared variables
int elves = 0;
int reindeer = 0;

// Define semaphores
sem_t santaSem;
sem_t reindeerSem;
sem_t elfTex;
sem_t mutex;

// Pre-declare functions
void *elfRunner(void *args);
void *reindeerRunner(void *args);
void *santaRunner(void *args);

void prepareSleigh();
void helpElves();
void getHitched();
void getHelp();

int main()
{
    // Loop variable
    int i=0;


    // Init the semaphore variables
    sem_init(&santaSem,0,0);
    sem_init(&reindeerSem,0,0);
    sem_init(&elfTex,0,1);
    sem_init(&mutex,0,1);

    // Create and run a Santa thread
    pthread_t santa_tid;
    pthread_create(&santa_tid,NULL,santaRunner,NULL);

    // Create and run N_ELVES thread
    pthread_t elf_tid[N_ELVES];
    for(i=0;i<N_ELVES;++i)
    {
```

```
        int *arg = malloc(sizeof(*arg));
        *arg = i + 1;
        pthread_create(&elf_tid[i],NULL,elfRunner,arg);
    }

    // Create and run the reindeer threads
    pthread_t reindeer_tid[N_REINDEERS];
    for(i=0;i<N_REINDEERS;++i)
    {
        int *arg = malloc(sizeof(*arg));
        *arg = i + 1;
        pthread_create(&reindeer_tid[i],NULL,reindeerRunner,arg);
    }

    // Join all threads
    pthread_join(santa_tid,NULL);

    for(i=0;i<N_ELVES;++i)
        pthread_join(elf_tid[i],NULL);

    for(i=0;i<N_REINDEERS;++i)
        pthread_join(reindeer_tid[i],NULL);

    return 0;
}

// Define the santa runner thread
void *santaRunner(void *args)
{
    int i = 0;
    printf("[Santa] Santa is Here !!!\n");
    while(1)
    {
        sem_wait(&santaSem);
        sem_wait(&mutex);

        if(reindeer == 9)
        {
                prepareSleigh();
                for(i=0;i<9;++i)
                        sem_post(&reindeerSem);
                printf("[Santa] Make kids happy\n");
                reindeer = 0;
```

```
        }
        else if(elves == 3)
                helpElves();

        sem_post(&mutex);
    }
}

// Define the reindeer runner thread
void *reindeerRunner(void *args)
{
    int reindeer_num = *(int*)args;
    while(1)
    {
        sem_wait(&mutex);
        reindeer++;
        if(reindeer == 9)
                sem_post(&santaSem);
        sem_post(&mutex);

        sem_wait(&reindeerSem);
        getHitched(reindeer_num);
        sleep(5);
    }
}

// Define the elf runner threads
void *elfRunner(void *args)
{
    int elf_num = *(int*)args;
    while(1)
    {
        sem_wait(&elfTex);
        sem_wait(&mutex);

        elves++;
        if(elves == 3)
                sem_post(&santaSem);
        else
                sem_post(&elfTex);

        sem_post(&mutex);
```

```
        getHelp(elf_num);

        sem_wait(&mutex);
        elves--;
        if(elves == 0)
              sem_post(&elfTex);
        sem_post(&mutex);
    }

    // Work a while
    printf("[Elf] Elf %d is working.\n",elf_num);
    sleep(3);
}

// Simple utility functions
void helpElves()
{
    printf("[Santa] Helping the elves.\n");
}

void prepareSleigh()
{
    printf("[Santa] Preparing Sleigh\n");
}

void getHelp(int elf_num)
{
    printf("[Elf] Elf %d needs help from Santa\n",elf_num);
    sleep(5);
}

void getHitched(int reindeer_num)
{
    printf("[Reindeer] Reindeer %d getting hitched.\n",reindeer_num);
    sleep(10);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ gcc -o out santaClaus.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ ./out
[Santa] Santa is Here !!!
[Santa] Helping the elves.
[Elf] Elf 2 needs help from Santa
[Elf] Elf 3 needs help from Santa
[Elf] Elf 1 needs help from Santa
[Santa] Preparing Sleigh
[Reindeer] Reindeer 1 getting hitched.
[Reindeer] Reindeer 5 getting hitched.
[Reindeer] Reindeer 2 getting hitched.
[Reindeer] Reindeer 4 getting hitched.
[Reindeer] Reindeer 7 getting hitched.
[Reindeer] Reindeer 8 getting hitched.
[Santa] Make kids happy
[Reindeer] Reindeer 3 getting hitched.
[Reindeer] Reindeer 9 getting hitched.
[Reindeer] Reindeer 6 getting hitched.
[Elf] Elf 3 needs help from Santa
[Elf] Elf 4 needs help from Santa
[Santa] Helping the elves.
[Elf] Elf 5 needs help from Santa
[Elf] Elf 5 needs help from Santa
[Elf] Elf 6 needs help from Santa
[Elf] Elf 9 needs help from Santa
[Santa] Helping the elves.
[Elf] Elf 8 needs help from Santa
[Elf] Elf 9 needs help from Santa
[Santa] Preparing Sleigh
[Reindeer] Reindeer 8 getting hitched.
[Reindeer] Reindeer 5 getting hitched.
[Reindeer] Reindeer 4 getting hitched.
[Reindeer] Reindeer 7 getting hitched.
[Reindeer] Reindeer 3 getting hitched.
[Reindeer] Reindeer 2 getting hitched.
[Santa] Make kids happy
[Reindeer] Reindeer 1 getting hitched.
[Reindeer] Reindeer 6 getting hitched.
[Santa] Helping the elves.
[Elf] Elf 2 needs help from Santa
[Reindeer] Reindeer 9 getting hitched.
[Elf] Elf 2 needs help from Santa
[Elf] Elf 7 needs help from Santa
[Santa] Helping the elves.
[Elf] Elf 4 needs help from Santa
[Elf] Elf 4 needs help from Santa
[Elf] Elf 10 needs help from Santa
[Elf] Elf 6 needs help from Santa
[Santa] Helping the elves.
[Santa] Preparing Sleigh
[Reindeer] Reindeer 5 getting hitched.
[Reindeer] Reindeer 7 getting hitched.
[Reindeer] Reindeer 1 getting hitched.
[Reindeer] Reindeer 4 getting hitched.
[Reindeer] Reindeer 3 getting hitched.
[Santa] Make kids happy
```

Senate Bus Problem

Logic Used

In this problem, we have to implement the semaphore based solution to the Senate Bus problem. Let us take a look at some of the important points of the problem:

- Riders come and wait for a bus.
- When the bus arrives all the waiting riders board the bus but anyone who arrives while the bus is boarding has to wait for the next bus.
- The capacity of the bus is 50, so if there are more than 50 riders waiting some will have to wait for the next bus.
- When all the waiting riders have boarded, the bus can depart.
- If the bus arrives when there are no riders, it should depart immediately.

There are two main thread runners in this program.

- 1 Bus - `bus_runner()`
- N Riders - `rider_runner()`

Let us take a look at the share variables and semaphores used.

```
// Define shared variables
int riders = 0;

// Define the semaphores
sem_t mutex;
sem_t multiplex;
sem_t bus;
sem_t allAboard;
```

```
// Init the semaphore variables
sem_init(&mutex,0,1);
sem_init(&multiplex,0,50);
sem_init(&bus,0,0);
sem_init(&allAboard,0,0);
```

mutex protects riders, which keeps track of how many riders are waiting; multiplex makes sure there are no more than 50 riders in the boarding area. Riders wait on the bus, which gets signaled when the bus arrives. The bus waits on allAboard, which gets signaled by the last student to board.

Let us take a look at the bus runner:

```
// Define the bus runner
void *bus_runner(void *args)
{
    while(1)
    {
       sem_wait(&mutex);
       if(riders > 0)
       {
             sem_post(&bus);
             sem_wait(&allAboard);
       }
       sem_post(&mutex);

       depart();
    }
}
```

When the bus arrives, it gets mutex, which prevents late arrivals from entering the boarding area. If there are no riders, it departs immediately. Otherwise, it signals the bus and waits for the riders to board.

Let us take a look at the rider runner:

```
// Define the rider runner
void *rider_runner(void *args)
{
    int rider_num = *(int*)args;
    while(1)
    {
       sem_wait(&multiplex);
       sem_wait(&mutex);
       ++riders;
       printf("[RIDER] No of waiting riders = %d\n",riders);
```

```
        sem_post(&mutex);

        sem_wait(&bus);
        sem_post(&multiplex);

        boardBus(rider_num);

        --riders;
        if(riders == 0)
              sem_post(&allAboard);
        else
              sem_post(&bus);
    }
}
```

The multiplex controls the number of riders in the waiting area, although strictly speaking, a rider doesn't enter the waiting area until she increments riders. Riders wait on the bus until the bus arrives. When a rider wakes up, it is understood that she has the mutex. After boarding, each rider decrements riders. If there are more riders waiting, the boarding rider signals the bus and passes the mutex to the next rider. The last rider signals allAboard and passes the mutex back to the bus.

Code

```
/*****************************************
* AUTHOR : AdheshR *
* Problem Description: *
*****************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <semaphore.h>
#define MAX 100

// Define shared variables
int riders = 0;
```

```c
// Define the semaphores
sem_t mutex;
sem_t multiplex;
sem_t bus;
sem_t allAboard;

// Define global const parameters
static const int N = 60;

// Declare functions
void *bus_runner(void *args);
void *rider_runner(void *args);

void boardBus(int rider_num);
void depart();

int main()
{
    // Loop variable
    int i = 0;

    // Init the semaphore variables
    sem_init(&mutex,0,1);
    sem_init(&multiplex,0,50);
    sem_init(&bus,0,0);
    sem_init(&allAboard,0,0);

    // Create and run the bus thread
    pthread_t bus_tid;
    pthread_create(&bus_tid,NULL,bus_runner,NULL);

    // Create and run the rider threads
    pthread_t rider_tid[N];
    for(i=0;i<N;++i)
    {
       int *arg = malloc(sizeof(*arg));
       *arg = i + 1;
       pthread_create(&rider_tid[i],NULL,rider_runner,arg);
    }

    // Join all threads
    pthread_join(bus_tid,NULL);
```

```
    for(i=0;i<N;++i)
       pthread_join(rider_tid[i],NULL);

    return 0;
}

// Define the bus runner
void *bus_runner(void *args)
{
    while(1)
    {
       sem_wait(&mutex);
       if(riders > 0)
       {
             sem_post(&bus);
             sem_wait(&allAboard);
       }
       sem_post(&mutex);

       depart();
    }
}

// Define the rider runner
void *rider_runner(void *args)
{
    int rider_num = *(int*)args;
    while(1)
    {
       sem_wait(&multiplex);
       sem_wait(&mutex);
       ++riders;
       printf("[RIDER] No of waiting riders = %d\n",riders);
       sem_post(&mutex);

       sem_wait(&bus);
       sem_post(&multiplex);

       boardBus(rider_num);

       --riders;
       if(riders == 0)
             sem_post(&allAboard);
```

```
        else
                sem_post(&bus);
    }
}

// Define utility functions
void boardBus(int rider_num)
{
    printf("[RIDER] Bus has arrived. Rider %d has boarded.\nNo of riders waiting
= %d\n",rider_num,riders);
    sleep(1);
}

void depart()
{
    printf("[BUS] Departing. Riders waiting = %d\n",riders);
    sleep(1);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ gcc -o out senateBus.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week8$ ./out
[BUS] Departing. Riders waiting = 0
[RIDER] No of waiting riders = 1
[RIDER] No of waiting riders = 2
[RIDER] No of waiting riders = 3
[RIDER] No of waiting riders = 4
[RIDER] No of waiting riders = 5
[RIDER] No of waiting riders = 6
[RIDER] No of waiting riders = 7
[RIDER] No of waiting riders = 8
[RIDER] No of waiting riders = 9
[RIDER] No of waiting riders = 10
[RIDER] No of waiting riders = 11
[RIDER] No of waiting riders = 12
[RIDER] No of waiting riders = 13
[RIDER] No of waiting riders = 14
[RIDER] No of waiting riders = 15
[RIDER] No of waiting riders = 16
[RIDER] No of waiting riders = 17
[RIDER] No of waiting riders = 18
[RIDER] No of waiting riders = 19
[RIDER] No of waiting riders = 20
[RIDER] No of waiting riders = 21
[RIDER] No of waiting riders = 22
[RIDER] No of waiting riders = 23
[RIDER] No of waiting riders = 24
[RIDER] No of waiting riders = 25
[RIDER] No of waiting riders = 26
[RIDER] No of waiting riders = 27
[RIDER] No of waiting riders = 28
[RIDER] No of waiting riders = 29
[RIDER] No of waiting riders = 30
[RIDER] No of waiting riders = 31
[RIDER] No of waiting riders = 32
[RIDER] No of waiting riders = 33
[RIDER] No of waiting riders = 34
[RIDER] No of waiting riders = 35
[RIDER] No of waiting riders = 36
[RIDER] No of waiting riders = 37
[RIDER] No of waiting riders = 38
[RIDER] No of waiting riders = 39
[RIDER] No of waiting riders = 40
[RIDER] No of waiting riders = 41
[RIDER] No of waiting riders = 42
[RIDER] No of waiting riders = 43
[RIDER] No of waiting riders = 44
[RIDER] No of waiting riders = 45
[RIDER] No of waiting riders = 46
[RIDER] No of waiting riders = 47
[RIDER] No of waiting riders = 48
[RIDER] No of waiting riders = 49
[RIDER] No of waiting riders = 50
[RIDER] Bus has arrived. Rider 1 has boarded.
No of riders waiting = 50
[RIDER] Bus has arrived. Rider 2 has boarded.
No of riders waiting = 49
[RIDER] Bus has arrived. Rider 6 has boarded.
No of riders waiting = 48
[RIDER] Bus has arrived. Rider 4 has boarded.
No of riders waiting = 47
[RIDER] Bus has arrived. Rider 5 has boarded.
No of riders waiting = 46
[RIDER] Bus has arrived. Rider 3 has boarded.
No of riders waiting = 45
```

```
[RIDER] Bus has arrived. Rider 8 has boarded.
No of riders waiting = 44
[RIDER] Bus has arrived. Rider 7 has boarded.
No of riders waiting = 43
[RIDER] Bus has arrived. Rider 10 has boarded.
No of riders waiting = 42
[RIDER] Bus has arrived. Rider 9 has boarded.
No of riders waiting = 41
[RIDER] Bus has arrived. Rider 11 has boarded.
No of riders waiting = 40
[RIDER] Bus has arrived. Rider 12 has boarded.
No of riders waiting = 39
[RIDER] Bus has arrived. Rider 13 has boarded.
No of riders waiting = 38
[RIDER] Bus has arrived. Rider 14 has boarded.
No of riders waiting = 37
[RIDER] Bus has arrived. Rider 15 has boarded.
No of riders waiting = 36
[RIDER] Bus has arrived. Rider 17 has boarded.
No of riders waiting = 35
[RIDER] Bus has arrived. Rider 16 has boarded.
No of riders waiting = 34
[RIDER] Bus has arrived. Rider 18 has boarded.
No of riders waiting = 33
[RIDER] Bus has arrived. Rider 19 has boarded.
No of riders waiting = 32
[RIDER] Bus has arrived. Rider 20 has boarded.
No of riders waiting = 31
[RIDER] Bus has arrived. Rider 21 has boarded.
No of riders waiting = 30
[RIDER] Bus has arrived. Rider 22 has boarded.
No of riders waiting = 29
[RIDER] Bus has arrived. Rider 23 has boarded.
No of riders waiting = 28
[RIDER] Bus has arrived. Rider 24 has boarded.
No of riders waiting = 27
[RIDER] Bus has arrived. Rider 26 has boarded.
No of riders waiting = 26
[RIDER] Bus has arrived. Rider 27 has boarded.
No of riders waiting = 25
[RIDER] Bus has arrived. Rider 25 has boarded.
No of riders waiting = 24
[RIDER] Bus has arrived. Rider 28 has boarded.
No of riders waiting = 23
[RIDER] Bus has arrived. Rider 29 has boarded.
No of riders waiting = 22
[RIDER] Bus has arrived. Rider 30 has boarded.
No of riders waiting = 21
[RIDER] Bus has arrived. Rider 31 has boarded.
No of riders waiting = 20
[RIDER] Bus has arrived. Rider 32 has boarded.
No of riders waiting = 19
[RIDER] Bus has arrived. Rider 33 has boarded.
No of riders waiting = 18
[RIDER] Bus has arrived. Rider 34 has boarded.
No of riders waiting = 17
[RIDER] Bus has arrived. Rider 35 has boarded.
No of riders waiting = 16
[RIDER] Bus has arrived. Rider 36 has boarded.
No of riders waiting = 15
[RIDER] Bus has arrived. Rider 37 has boarded.
No of riders waiting = 14
[RIDER] Bus has arrived. Rider 38 has boarded.
No of riders waiting = 13
```

```
[RIDER] Bus has arrived. Rider 39 has boarded.
No of riders waiting = 12
[RIDER] Bus has arrived. Rider 40 has boarded.
No of riders waiting = 11
[RIDER] Bus has arrived. Rider 41 has boarded.
No of riders waiting = 10
[RIDER] Bus has arrived. Rider 42 has boarded.
No of riders waiting = 9
[RIDER] Bus has arrived. Rider 43 has boarded.
No of riders waiting = 8
[RIDER] Bus has arrived. Rider 44 has boarded.
No of riders waiting = 7
[RIDER] Bus has arrived. Rider 45 has boarded.
No of riders waiting = 6
[RIDER] Bus has arrived. Rider 46 has boarded.
No of riders waiting = 5
[RIDER] Bus has arrived. Rider 48 has boarded.
No of riders waiting = 4
[RIDER] Bus has arrived. Rider 50 has boarded.
No of riders waiting = 3
[RIDER] Bus has arrived. Rider 52 has boarded.
No of riders waiting = 2
[RIDER] Bus has arrived. Rider 54 has boarded.
No of riders waiting = 1
[BUS] Departing. Riders waiting = 0
[RIDER] No of waiting riders = 1
[RIDER] No of waiting riders = 2
[RIDER] No of waiting riders = 3
[RIDER] No of waiting riders = 4
[RIDER] No of waiting riders = 5
[RIDER] No of waiting riders = 6
[RIDER] No of waiting riders = 7
[RIDER] No of waiting riders = 8
[RIDER] No of waiting riders = 9
[RIDER] No of waiting riders = 10
[RIDER] No of waiting riders = 11
[RIDER] No of waiting riders = 12
[RIDER] No of waiting riders = 13
[RIDER] No of waiting riders = 14
[RIDER] No of waiting riders = 15
[RIDER] No of waiting riders = 16
[RIDER] No of waiting riders = 17
[RIDER] No of waiting riders = 18
[RIDER] No of waiting riders = 19
[RIDER] No of waiting riders = 20
[RIDER] No of waiting riders = 21
[RIDER] No of waiting riders = 22
[RIDER] No of waiting riders = 23
[RIDER] No of waiting riders = 24
[RIDER] No of waiting riders = 25
[RIDER] No of waiting riders = 26
[RIDER] No of waiting riders = 27
[RIDER] No of waiting riders = 28
[RIDER] No of waiting riders = 29
[RIDER] No of waiting riders = 30
[RIDER] No of waiting riders = 31
[RIDER] No of waiting riders = 32
[RIDER] No of waiting riders = 33
[RIDER] No of waiting riders = 34
[RIDER] No of waiting riders = 35
[RIDER] No of waiting riders = 36
[RIDER] No of waiting riders = 37
[RIDER] No of waiting riders = 38
```