OS LAB DIGITAL ASSIGNMENT 3

PRITHISH SAMANTA 19BCE2261

1. Producer and Consumer

```
#include<stdio.h>
#include<sys/types.h>
#include<pthread.h>
#include<semaphore.h>
sem_t mutex;
sem_t full;
sem_t empty;
char buffer[10];

void *producer()
{
int i,index=0;
for(i=0;i<26;i++)
{
sem_wait(&empty);
sem_wait(&mutex);
buffer[index]=i+64;
printf("Producer added %c to buffer\t with thread id %ld \n",buffer[index],pthread_self());
sem_post(&full);
sem_post(&mutex);
if(++index==10)
index=0;
```

```
/*if(rand()%3==0)
sleep(5);*/
}
}
void *consumer()
{
int i,index=0;
for(i=0;i<26;i++)
{
sem_wait(&full);
sem_wait(&mutex);
printf("Consumer consumed %c\n ",buffer[index]);
sem_post(&empty);
sem_post(&mutex);
if(++index==10)
index=0;
/*if(rand()%3==0)
sleep(5);*/
}
}

int main()
{
int i;
pthread_t tid1[10],tid2[10],tid3,tid4;
sem_init(&mutex,0,1);
```

```
sem_init(&full,0,0);

sem_init(&empty,0,10);

for(i=0;i<26;i++)

{

pthread_create(&tid1[i],NULL,producer,NULL);

pthread_create(&tid2[i],NULL,consumer,NULL);

//pthread_create(&tid3,NULL,consumer,NULL);

//pthread_create(&tid4,NULL,consumer,NULL);

pthread_join(tid1[i],NULL);

pthread_join(tid2[i],NULL);

sem_destroy(&mutex);

sem_destroy(&full);

sem_destroy(&empty);

}

}
```

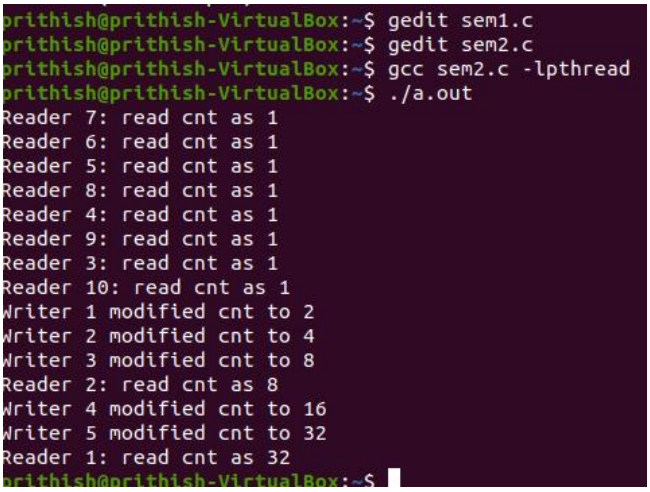
```
prithish@prithish-VirtualBox:~$ gedit sem1.c
prithish@prithish-VirtualBox:~$ gcc sem1.c -lpthread
prithish@prithish-VirtualBox:~$ ./a.out
Producer added @ to buffer       with thread id 140039957169920
Producer added A to buffer       with thread id 140039957169920
Producer added B to buffer       with thread id 140039957169920
Producer added C to buffer       with thread id 140039957169920
Producer added D to buffer       with thread id 140039957169920
Producer added E to buffer       with thread id 140039957169920
Producer added F to buffer       with thread id 140039957169920
Producer added G to buffer       with thread id 140039957169920
Producer added H to buffer       with thread id 140039957169920
Producer added I to buffer       with thread id 140039957169920
Consumer consumed @
 Consumer consumed A
 Consumer consumed B
 Consumer consumed C
 Consumer consumed D
 Consumer consumed E
 Consumer consumed F
 Consumer consumed G
 Consumer consumed H
 Consumer consumed I
 Producer added J to buffer      with thread id 140039957169920
Producer added K to buffer       with thread id 140039957169920
Producer added L to buffer       with thread id 140039957169920
Producer added M to buffer       with thread id 140039957169920
Producer added N to buffer       with thread id 140039957169920
Producer added O to buffer       with thread id 140039957169920
Producer added P to buffer       with thread id 140039957169920
Producer added Q to buffer       with thread id 140039957169920
Producer added R to buffer       with thread id 140039957169920
Producer added S to buffer       with thread id 140039957169920
Consumer consumed J
 Consumer consumed K
 Consumer consumed L
 Consumer consumed M
 Consumer consumed N
 Consumer consumed O
 Consumer consumed P
 Consumer consumed Q
 Consumer consumed R
 Consumer consumed S
```

2. Readers Writers

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
   sem_post(&wrt);

}
void *reader(void *rno)
{
   // Reader acquire the lock before modifying numreader
   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt); // If this id the first reader, then it will block the writer
   }
```

```
    pthread_mutex_unlock(&mutex);
    // Reading Section
    printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);

    // Reader acquire the lock before modifying numreader
    pthread_mutex_lock(&mutex);
    numreader--;
    if(numreader == 0) {
        sem_post(&wrt); // If this is the last reader, it will wake up the writer.
    }
    pthread_mutex_unlock(&mutex);
}

int main()
{

    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&wrt,0,1);

    int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer

    for(int i = 0; i < 10; i++) {
        pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
    }
    for(int i = 0; i < 5; i++) {
```

```
        pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
    }

    for(int i = 0; i < 10; i++) {
        pthread_join(read[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(write[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);
    return 0;
}
```



3. Dining Philosophers

```
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <stdio.h>
```

```
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
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]) has no effect
// during takefork
// used to wake up hungry philosophers
// during putfork
sem_post(&S[phnum]);
```

```
}
}
// take up chopsticks
void take_fork(int phnum)
{
sem_wait(&mutex);
// state that hungry
state[phnum] = HUNGRY;
printf("Philosopher %d is Hungry\n", phnum + 1);
// eat if neighbours are not eating
test(phnum);
sem_post(&mutex);
// if unable to eat wait to be signalled
sem_wait(&S[phnum]);
sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
sem_wait(&mutex);
// state that 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* philospher(void* num)
{
while (1) {
int* i = num;
sleep(1);
take_fork(*i);
sleep(0);
put_fork(*i);
}
}
int main()
{
int i;
pthread_t thread_id[N];
// initialize the semaphores
sem_init(&mutex, 0, 1);
for (i = 0; i < N; i++)
sem_init(&S[i], 0, 0);
for (i = 0; i < N; i++) {
// create philosopher processes
pthread_create(&thread_id[i],NULL,philospher, &phil[i]);
printf("Philosopher %d is thinking\n", i + 1);
```

}

for (i = 0; i < N; i++)

pthread_join(thread_id[i], NULL);

```
prithish@prithish-VirtualBox:~$ gedit sem3.c
prithish@prithish-VirtualBox:~$ gcc sem3.c -lpthread
prithish@prithish-VirtualBox:~$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 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 5 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 2 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 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
```

4. Banker's Algorithm

#include <stdio.h>

int main()

{

int n, m, i, j, k;

printf("Enter no of processes: \n");

scanf("%d", &n);

printf("Enter no of resources: \n");

scanf("%d", &m);

```
int alloc[n][m];
int max[n][m];
for(int i=0; i<n; i++){
for(int j=0; j<m; j++){
printf("Enter max resouces for P[%d]", i);
scanf("%d", &max[i][j]);
printf("\n");
}
}
for(int i=0; i<n; i++){
for(int j=0; j<m; j++)
{
printf("Enter allocated resources for P[%d]", i);
scanf("%d", &alloc[i][j]);
printf("\n");
}
}
int avail[m];
for(int i=0; i<m; i++){
printf("Enter available no of instances for Resource[%d]", i);
scanf("%d", &avail[i]);
}
int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
f[k] = 0;
}
```

```
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
printf("Allocated Matrix \n");
for(int i=0; i<n; i++)
{
printf("P[%d] ", i);
for(int j=0; j<m; j++)
{
printf("%d ", alloc[i][j]);
}
printf("\n");
}
printf("Needed Matrix \n");
for(int i=0; i<n; i++){
printf("P[%d] ", i);
for(int j=0; j<m; j++){
printf("%d ", need[i][j]);
}
printf("\n");
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
```

```
if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d\n", ans[n - 1]);
return (0);
}
```

```
prithish@prithish-VirtualBox:~$ gedit banker.c
prithish@prithish-VirtualBox:~$ gcc banker.c -lpthread
prithish@prithish-VirtualBox:~$ ./a.out
Enter no of processes:
4
Enter no of resources:
3
Enter max resouces for P[0]3

Enter max resouces for P[0]2

Enter max resouces for P[0]0

Enter max resouces for P[1]0

Enter max resouces for P[1]1

Enter max resouces for P[1]0

Enter max resouces for P[2]0

Enter max resouces for P[2]1
```

```
Enter max resouces for P[2]3

Enter max resouces for P[3]2

Enter max resouces for P[3]0

Enter max resouces for P[3]0

Enter allocated resources for P[0]1

Enter allocated resources for P[0]0

Enter allocated resources for P[0]0

Enter allocated resources for P[1]0

Enter allocated resources for P[1]0

Enter allocated resources for P[1]0

Enter allocated resources for P[2]0

Enter allocated resources for P[2]1

Enter allocated resources for P[2]2

Enter allocated resources for P[3]0

Enter allocated resources for P[3]0

Enter allocated resources for P[3]0

Enter available no of instances for Resource[0]5
Enter available no of instances for Resource[1]5
Enter available no of instances for Resource[2]3
Allocated Matrix
P[0] 1 0 0
P[1] 0 0 0
P[2] 0 1 2
P[3] 0 0 0
Needed Matrix
P[0] 2 2 0
P[1] 0 1 0
P[2] 0 0 1
P[3] 2 0 0
Following is the SAFE Sequence
 P0 -> P1 -> P2 -> P3
prithish@prithish-VirtualBox:~$
```