



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GENERAL SEMESTER 2023-24

B.Tech - CSE

BCSE303P: Operating Systems Lab

DIGITAL LAB ASSIGNMENT -3

NAME-SHREYAS NEHE

REG.NO-21BCE3359

SLOT-L49+L50

Q3A Implement process synchronization using semaphores.

```
#include <stdio.h>

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

#define BUFFER_SIZE 5

sem_t empty;
sem_t full;
int buffer[BUFFER_SIZE];
int index = 0;

void* producer(void* arg) {
    int item = *((int*)arg);
    sem_wait(&empty);
```

```
// Critical section
buffer[index] = item;
printf("Produced item: %d\n", item);
index++;

sem_post(&full);
pthread_exit(NULL);
}

void* consumer(void* arg) {
    sem_wait(&full);

    // Critical section
    int item = buffer[index - 1];
    printf("Consumed item: %d\n", item);
    index--;

    sem_post(&empty);
    pthread_exit(NULL);
}

int main() {
    int numItems;
    printf("Enter the number of items to produce and consume: ");
    scanf("%d", &numItems);

    sem_init(&empty, 0, BUFFER_SIZE); // Initialize empty semaphore
    with BUFFER_SIZE
    sem_init(&full, 0, 0); // Initialize full semaphore with 0

    pthread_t producerThread, consumerThread;
```

```
for (int i = 1; i <= numItems; i++) {  
    pthread_create(&producerThread, NULL, producer, &i);  
    pthread_create(&consumerThread, NULL, consumer, NULL);  
    pthread_join(producerThread, NULL);  
    pthread_join(consumerThread, NULL);  
}  
  
sem_destroy(&empty);  
sem_destroy(&full);  
  
return 0;  
}
```

```
#include <stdio.h>  
#include <pthread.h>  
#include <semaphore.h>  
  
#define BUFFER_SIZE 5  
  
sem_t empty;  
sem_t full;  
int buffer[BUFFER_SIZE];  
int index = 0;  
  
void* producer(void* arg) {  
    int item = *((int*)arg);  
    sem_wait(&empty);  
  
    // Critical section  
    buffer[index] = item;  
    printf("Produced item: %d\n", item);  
    index++;  
  
    sem_post(&full);  
    pthread_exit(NULL);  
}  
  
void* consumer(void* arg) {  
    sem_wait(&full);  
  
    // Critical section  
    int item = buffer[index - 1];  
    printf("Consumed item: %d\n", item);  
    index--;  
  
    sem_post(&empty);  
    pthread_exit(NULL);  
}
```

```
void* threadFunction(void* threadId) {  
    int tid = *((int*)threadId);  
  
    printf("Thread %d is waiting.\n", tid);  
    sem_wait(&semaphore);  
    printf("Thread %d has acquired the semaphore.\n", tid);  
  
    // Critical section  
    printf("Thread %d is in the critical section.\n", tid);  
  
    printf("Thread %d is releasing the semaphore.\n", tid);  
    sem_post(&semaphore);  
  
    pthread_exit(NULL);  
}
```

```
int main() {  
    int numItems;  
    printf("Enter the number of items to produce and consume: ");  
    scanf("%d", &numItems);  
  
    sem_init(&empty, 0, BUFFER_SIZE); // Initialize empty semaphore with BUFFER_SIZE  
    sem_init(&full, 0, 0); // Initialize full semaphore with 0  
  
    pthread_t producerThread, consumerThread;  
  
    for (int i = 1; i <= numItems; i++) {  
        pthread_create(&producerThread, NULL, producer, &i);  
        pthread_create(&consumerThread, NULL, consumer, NULL);  
        pthread_join(producerThread, NULL);  
        pthread_join(consumerThread, NULL);  
    }  
  
    sem_destroy(&empty);  
    sem_destroy(&full);  
  
    return 0;  
}
```

OUTPUT:

```
Enter the number of items to produce and consume: 2 2
Produced item: 1
Consumed item: 1
Produced item: 2
Consumed item: 2
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

```
Thread 0 is waiting.
Thread 0 has acquired the semaphore.
Thread 0 is in the critical section.
Thread 0 is releasing the semaphore.
Thread 1 is waiting.
Thread 1 has acquired the semaphore.
Thread 1 is in the critical section.
Thread 1 is releasing the semaphore.
Thread 2 is waiting.
Thread 2 has acquired the semaphore.
Thread 2 is in the critical section.
Thread 2 is releasing the semaphore.
Thread 4 is waiting.
Thread 4 has acquired the semaphore.
Thread 4 is in the critical section.
Thread 4 is releasing the semaphore.
Thread 3 is waiting.
Thread 3 has acquired the semaphore.
Thread 3 is in the critical section.
Thread 3 is releasing the semaphore.

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Input
Enter the number of items to produce and consume: 3 4
Produced item: 1
Consumed item: 1
Produced item: 2
Consumed item: 2
Produced item: 3
Consumed item: 3

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Input
Thread 0 is waiting.
Thread 0 has acquired the semaphore.
Thread 0 is in the critical section.
Thread 0 is releasing the semaphore.
Thread 2 is waiting.
Thread 2 has acquired the semaphore.
Thread 2 is in the critical section.
Thread 2 is releasing the semaphore.
Thread 1 is waiting.
Thread 1 has acquired the semaphore.
Thread 1 is in the critical section.
Thread 1 is releasing the semaphore.
Thread 3 is waiting.
Thread 3 has acquired the semaphore.
Thread 3 is in the critical section.
Thread 3 is releasing the semaphore.
Thread 4 is waiting.
Thread 4 has acquired the semaphore.
Thread 4 is in the critical section.
Thread 4 is releasing the semaphore.

...Program finished with exit code 0
Press ENTER to exit console.
```

Q3B Simulation of Banker s algorithm to check whether the given system is in safe state or not. Also check whether addition resource requested can be granted immediately.

CODE:

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

```
int n,m;
int i,j;
int a[100][100];
int highest[100][100];
int whichisavailable[100];
int needd[100][100];
int w[100];
int f[100];
int safseq[100];

void init()
{
    int i,j;
    for(i=0;i<m;i++)
        w[i]=whichisavailable[i];

    for(i=0;i<n;i++)
        f[i]=0;

    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            needd[i][j]=highest[i][j]-a[i][j];
    }
}
```

```
int sequence_whichisSAFE()
{
    int i,j,k;
    int found=0;
    int count=0;
    while(count<n){
        int find=0;
        mark:for(i=0;i<n;i++)
        {
            if(!f[i]){
                int exe=1;
                for(j=0;j<m;j++){
                    if(needd[i][j]>w[j]){
                        exe=0;
                        break;
                    }
                }
                if(exe)
                {
                    for(k=0;k<m;k++)
                    {
                        w[k]+=a[i][k];
                    }
                    f[i]=1;
                    safseq[count]=i;
                    count++;
                    found=1;
                    goto mark;
                }
            }
        }
    }
}
```



```
        if(!found){
            return 0;
        }
    }
    return 1;
}

int st()
{
    init();
    if(sequence_whichissafeUENCE())
    {

        printf("safe\n");
        for(int i=0;i<n;i++){
            printf("%d",safseq[i]);
            printf(" ");
        }
        printf("\n");
        return 1;

    }
    else
    {
        printf("System is in unsafe state\n");
        return 0;
    }
}

int reqq(int process,int req[])
```

```
{
    int i;
    for(i=0;i<m;i++)
    {
        if(req[i]>needd[process][i])
            return 0;
        if(req[i]>whichisavailablee[i])
            return 0;
    }
    return 1;
}

void grant(int process,int req[])
{
    int i=0;
    for(i=0;i<m;i++)
    {
        whichisavailablee[i]-=req[i];
        a[process][i]+=req[i];
        needd[process][i]-=req[i];
    }
}

void print(int matrix[100][100])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++){
            printf("%d",matrix[i][j]);
        }
    }
}
```

```
        printf("\n");
    }
}

int main()
{
    printf("Number of processes:")
    scanf("%d",&n);
    printf("Number of resource types:")
    scanf("%d",&m);
    printf("enter allocation matrix:")
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            scanf("%d",&a[i][j]);
    }
    printf("enter max matrix:")
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
            scanf("%d",&highest[i][j]);
    }
    printf("enter available matrix:")
    for(i=0;i<m;i++)
        scanf("%d",&whichisavailable[i]);
    int process;
    scanf("%d",&process);
    int reqqqq[100];
    for(i=0;i<m;i++)
        scanf("%d",&reqqqq[i]);

    int ISSAFE=st();
```

```
printf("\n");  
if(ISSAFE && reqq(process,reqqqq)){  
  
    grant(process,reqqqq);  
    printf("safe\n");  
    for(i=0;i<n;i++){  
        printf("%d",safseq[i]);  
        printf(" ");  
    }  
    printf("\n");  
    printf("granted\n");  
}  
else  
{  
    printf("not granted\n");  
}  
return 0;  
  
}
```

OUTPUT WITH CODE:

```
1  #include<stdio.h>
2  #include<iostream>
3  using namespace std;
4      int n,m;
5      int i,j;
6      int a[100][100];
7      int highest[100][100];
8      int whichisavallablee[100];
9      int needd[100][100];
10     int w[100];
11     int f[100];
12     int safseq[100];
13
14     void init()
15     {
16         int i,j;
17         for(i=0;i<m;i++)
18             w[i]=whichisavallablee[i];
19
20         for(i=0;i<n;i++)
21             f[i]=0;
22
23         for(i=0;i<n;i++)
24         {
25             for(j=0;j<m;j++)
26                 needd[i][j]=highest[i][j]-a[i][j];
27         }
28     }
29
30     int sequence_whichissafeUENCE()
31     {
32         int i,j,k;
33         int found=0;
34         int count=0;
35         while(count<n){
36             int find=0;
37             mark:for(i=0;i<n;i++)
38             {
39                 if(!f[i]){
40                     int exe=1;
41                     for(j=0;j<m;j++){
42                         if(needd[i][j]>w[j]){
43                             exe=0;
44                             break;
45                         }
46                     }
47                 }
48             }
49             if(exe){
50                 f[i]=1;
51                 safseq[count]=i;
52                 count++;
53             }
54         }
55         return count;
56     }
57 }
```

Ln 8, Col 32 Spaces: 4 UTF-8 CRLF {} C

```
46     }
47     if(exe)
48     {
49         for(k=0;k<m;k++)
50         {
51             w[k]+=a[i][k];
52         }
53         f[i]=1;
54         safseq[count]=i;
55         count++;
56         found=1;
57         goto mark;
58     }
59     }
60 }
61
62 if(!found){
63     return 0;
64 }
65 }
66 return 1;
67 }
68
69
70 int st()
71 {
72     init();
73     if(sequence_whichissafeUENCE())
74     {
75
76         printf("safe\n");
77         for(int i=0;i<n;i++){
78             printf("%d",safseq[i]);
79             printf(" ");
80         }
81         printf("\n");
82         return 1;
83     }
84 }
85 else
86 {
87     printf("System is in unsafe state\n");
88     return 0;
89 }
90 }
```

Ln 8, Col 33

```
}

int reqq(int process,int req[])
{
    int i;
    for(i=0;i<m;i++)
    {
        if(req[i]>needd[process][i])
            return 0;
        if(req[i]>whichisavablee[i])
            return 0;
    }
    return 1;
}

void grant(int process,int req[])
{
    int i=0;
    for(i=0;i<m;i++)
    {
        whichisavablee[i]-=req[i];
        a[process][i]+=req[i];
        needd[process][i]-=req[i];
    }
}

void print(int matrix[100][100])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++){
            printf("%d",matrix[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    printf("Number of processes:")
    scanf("%d",&n);
    printf("Number of resource types:")
```

```
Number of processes:5
Number of resource types:3
enter allocation matrix:0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
enter max matrix:7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
enter available matrix:3 3 2
1
1 0 2
safe
1 3 0 2 4

safe
1 3 0 2 4
granted

...Program finished with exit code 0
Press ENTER to exit console.
```



```
133     printf("Number of resource types:");
134     scanf("%d",&m);
135     printf("enter allocation matrix:");
136     for(i=0;i<n;i++)
137     {
138         for(j=0;j<m;j++)
139             scanf("%d",&a[i][j]);
140     }
141     printf("enter max matrix:")
142     for(i=0;i<n;i++)
143     {
144         for(j=0;j<m;j++)
145             scanf("%d",&highest[i][j]);
146     }
147     printf("enter available matrix:")
148     for(i=0;i<m;i++)
149         scanf("%d",&whichisavailable[i]);
150     int process;
151     scanf("%d",&process);
152     int reqqqq[100];
153     for(i=0;i<m;i++)
154         scanf("%d",&reqqqq[i]);
155
156     int ISSAFE=st();
157     printf("\n");
158     if(ISSAFE && reqq(process,reqqqq)){
159
160         grant(process,reqqqq);
161         printf("safe\n");
162         for(i=0;i<n;i++){
163             printf("%d",safseq[i]);
164             printf(" ");
165         }
166         printf("\n");
167         printf("granted\n");
168     }
169     else
170     {
171         printf("not granted\n");
172     }
173     return 0;
174
175 }
```