Enter yo
consumer consumes item-2
Enter your choice : 2
consumer choice item 2
Buffer is empty

6. write a C program to simulate the concept of dining-philosophers problem

```
#incolude <stdio.h>
# include <stdlib.h>
# incltude <pthread.h>
# include < unistd.h>
# include < semaphore.h>

# define  NUM_PHIL 3
sem_t  forks [NUM_PHIL)
pthread_t  p[NUM_PHIL);
void * philosopher (void *arg) {
        int id = *((int*) arg);
        int left_fork = id;
        int right_fork = (id + 1) % NUM_PHIL ;
      while (1) {
          printf("philosopher %d is thinking\n",
                    id);
          sleep (rand() %3 + 1);
          printf("philosopher %d is hunger and
                   trying to pick forks\n",
                   id);
        sem_wait (& forks [left_fork]);
```

```
printf("philosopher %d picked up left fork %d\n",
          id, left-fork);
sem-wait(&fork[right-fork]);
printf("philosopher %d picked up right fork %d\n",
          id, right-fork);
printf("philosopher %d is eating\n", philosopher);
sem-post(&fork[left-fork]);
sem-post(&fork[right-fork]);
printf("philosopher %d finished eating and
          released fork\n", id);

}
return null;

}

int main(){
    int i;
    int ids[NUM-PHIL];
    for(int i=0; i< NUM-PHIL; i++){
        if(sem-init(&fork[i], 0, 1) != 0){
            perror("semaphore initialization
                      failed");
            exit(EXIT-FAILURE);
        }
    }
    for(int i=0; i< NUM-PHIL; i++){
        ids[i]=i;
        if(pthread-create(&p[i], NULL, P,
                          &id[i]) != 0){
            perror("Thread creation failed");
            exit(EXIT-FAILURE);
for (int i=0; i< NUM_PHIL; i++){
        if (pthread_join (p[i], NULL) !=0){
                perror("Thread join failed");
                exit(EXIT_FAILURE);
        }
}
for (int i=0; i< NUM_PHIL; i++) {
        if (sem_destroy (&forks[i]) !=0){
                perror("semaphore destruction
                        failed");
                exit(EXIT_FAILURE);
        }
}
return 0;
}
```

Output:

```
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 3 is hungry
philosopher 2 is hungry
philosopher 1 is hungry.
philosopher 1 takes fork 3 and 1
philosopher 1 is eating.
philosopher 1 putting fork 3 and 1 ~~then~~ down
philosopher 1 is thinking
```

Write a C program to simulate Banker Algorithm for the purpose of deadlock avoidance

```
#include <stdio.h>
int main()
{
    int n,m,i,j,k;
    n=5;
    m=3;
    int alloc[5][3] = { { 0, 1, 0 },
                        { 2, 0, 0 },
                        { 3, 0, 2 },
                        { 2, 1, 1 },
                        { 0, 0, 2 } };
    int max[5][3] = { { 7, 5, 3 },
                      { 3, 2, 2 },
                      { 9, 0, 2 },
                      { 2, 2, 2 }
                      { 4, 3, 3 } };
    int avail[3] = {3, 3, 2}
    int f[n], ans[n], ind=0;
    for (k=0; k<n ; k++){
        f[k]=0;
    }
    int need[n][m];
    for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            need[i][j] = max[i][j] -
                         alloc[i][j];
        }
    }
    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]
                }
        }
    }
  }
int flag =1 ;
for (int i =0; i<n ;i++)
{
    if (f[i] ==0)
    {
        flag=0;
        printf("The following system is not
        ~~break;~~ safe");
    ~~}~~
}
if(flag ==1)
{
    printf ("Following is the safe sequence\n");
    for(i=0; i<n-1 ;i++)
        printf("p%d →", ans[i]);
    ttf("p%d", ans[n-1]);
    }
    return 0;
}
```

Output

```
following is the safe sequence
P1 → P3 → P4 → P0 → P2
```

③ write a C program to simulate deadlock detection

```
#include <stdio.h>
void main()
{
    int n,m,i,j;
    printf("Enter the number of process and
            number of types of resource \n");
    scanf("%d %d", &n, &m);
    int max[n][m], need[n][m], all[n][m],
    ava[m], flag=1, finish[n], dead[n], c=0;
    printf("Enter the max of each type of resource
           needed by each proce \n");
    for(int i=0; i<n; i++)
    {
          for(int j=0; j<m; j++)
          {
                scanf("%d", &max[i][j]);
          }
    }
    printf("Enter the allocated no. of each
            type of resource needed by each
            proces \n");
        for(j=0;j<m; j++){
              scanf("%d", &ava[j]);
        }
```

```
for (i=0 ; i<n; i++)
{  for (j=0; j<m; j++)
     {    need[i][i] = max (i)[i] - all(i][i];

     }.

}
for (i=0 ; i<n ; i++)
{    finich[i]=0;

}
while (flag)
{
       flag =0;
       for (i=0; i<n ; i++)
       {
              c=0;
              for(j=0; j<m ; j++)
              {
                     if (finich[i]==0 &&
                        need[i](j] <=ava(i]){

                     c++;
                     if (c==m)
                     {
                        for (j=0;j<m; j++)
                        {
                           ava[i] += all[i][i];
                             finich[i] =1;
                              flag =1;
                        }
                     if (finich[i]==1)
                     {  i=n;
                     }
j=0;
flag=0;
for(i=0;i<n;i++)
{   if(finish[i]==0)
    {     dead[j]=i;
          j++;
          flag=1;
    }
}
if(flag==1)
{
    printf("Deadlock has occurred:\n");
    printf("the deadlock process are:\n");
    for(i=0;i<n;i++)
    {     printf("P%d", dead[i]);
    }
}
else
 printf("No deadlock has occurred\n");
}.
```

Output:

```
Enter the no. of process and number of types
of resource
5  4.
Enter max number of each type of resource needed
by each process:
 0  0  1  2
 1  7  5  0
```

```
2 3 5 8
0 6 5 2
0 6 5 6
Enter the allocated no. of each type of resource
needed by each process:
0 0 1 2
1 4 2 0
1 3 6 4
0 5 3 2
0 0 1 4
Enter the available no. of each type of resource
1 1 0 0.

Deadlock has occurred:
The deadlock process are:
P1 P2 P3 P4
```

20/6/2024