12/6/24

Q Write a C program to stimulate producer-consumer problem using semaphores

→
```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1, full = 0, empty = 3, x = 0;
int main()
{ int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);
  printf("\n1.Producer\n2.Consumer\n3.Exit");
  while(1)
  {
    printf("\nEnter your choice: ");
    scanf("%d", &n);
    switch(n)
    {
      case 1: if((mutex==1)&&(empty!=0))
              producer();
              else
              printf("Buffer is full!!");
              break;
      case 2: if((mutex==1)&&(full!=0))
              consumer();
              else
              print("Buffer is empty!!");
              break;
```

```
        case 3 : exit(0);
                 break;
        }
    } return 0;
}
int wait (int s)
{   return (--s); }

int signal (int s)
{   return (++s); }

void producer()
{   mutex = wait (mutex);
    full = signal (full);
    empty = wait (empty);
    x++;
    printf ("\n Producer produces item %d", x);
    mutex = signal (mutex);
}
void consumer()
{   mutex = wait (mutex);
    full = wait (full);
    empty = signal (empty);
    printf ("\n Consumer consumes item %d", x);
    x--;
    mutex = signal (mutex);
}
```

OUTPUT

1. Producer
2. Consumer
3. Exit

Enter your choice: 1
Producer produces the item 1
Enter your choice: 1
Producer produces the item 2
Enter your choice: 2
consumer consumes item 2
Enter your choice: 2
consumer consumes item 1
Enter your choice: 2
Buffer is empty !!
Enter your choice: 1
~~Consumer~~ ~~on~~ Producer produces item 1
Enter your choice: 3

8 Write a C program to stimulate the concept of Dining - Philosophers problem

→
```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i+4)%N
#define RIGHT (i+1)%N
int state[N];
int phil[N] = {0,1,2,3,4}
Sem_t mutex;
Sem_t S[N];
void test (int i)
{   if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {   state[i] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n", i+1, LEFT+1, i+1);
        printf("Philosopher %d is Eating\n", i+1);
        Sem_post(&S[i]);
    }
}
```

```
void take_fork (int i)
{  Sem_wait (&mutex);
   state [i] = HUNGRY;
   printf ("Philosopher %d is Hungry\n", i+1);
   test (i);
   Sem_post (&mutex);
   Sem_wait (&S[i]);
   Sleep (1);
}
void put_fork(int i)
{  Sem_wait(&mutex);
   State[i] = THINKING;
   printf("Philosopher %d putting fork %d & %d
down\n", i+1, LEFT, i+1);
   printf("Philosopher %d is thinking\n", i+1);
   test(LEFT);
   test(RIGHT);
   Sem_post(&mutex);
}
void * philosopher(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];
  Sem_init (&mutex, 0, 1);
  for (i=0; i<N; i++)
      sem_init (&S[i], 0, 0);
  for (i=0; i<N; i++)
  { pthread_create(&thread_id[i], NULL,
    philosopher, &phil[i]);
    printf ("Philosopher %d is thinking", i+1);
  }
  for (i=0; i<N; i++)
  {
      pthread_join (thread_id[i], NULL);
  }
}
```

OUTPUT

Philosopher 1 is thinking
P2 is thinking
P3 " "
P4 " "
P5 " "
P5 is Hungry
P1 " "
P2 " "
P3 " "
P4 " "

P4 takes fork 3 & 4

P4 is eating

P4 is putting fork 3 & 4 down

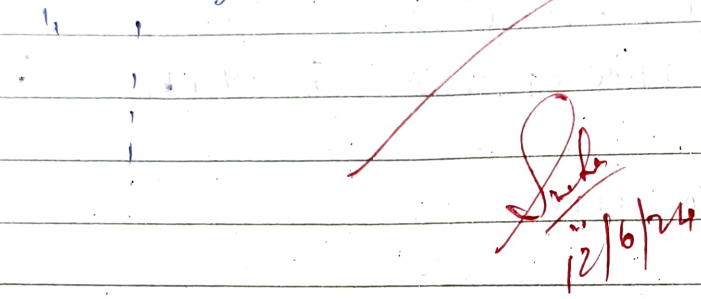
P4 is thinking

P3 takes fork 2 and 3

P3 is eating

P3 is putting fork 2 & 3 down

P3 is thinking



12/6/24

8 Write a C program to stimulate Banker's algorithm for the purpose of deadlock avoidance

→
```
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    printf("Enter number of processes:");
    scanf("%d", &n);
    printf("Enter number of resources:");
    scanf("%d", &m);
    int allocation[n][m];
    printf("Enter the Allocation Matrix:\n");
    for (i=0; i<n; i++)
    { for (j=0; j<m; j++)
        { scanf("%d", &allocation[i][j]); }
    }
    int max[n][m];
    printf("Enter the max matrix:\n");
    for (i=0; i<n; i++)
    { for (j=0; j<m; j++)
        { scanf("%d", &max[i][j]);
        }
    }
    int available[m];
    printf("Enter the Available resources:\n");
    for (i=0; i<m; i++)
    { scanf("%d", &available[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] - allocation[i][j];
    }
}
int y=0
for(k=0; k<n; k++)
{   for (i=0; i<n; i++)
    {   if (f[i] == 0)
        {   int flag = 0;
            for (j=0; j<m; j++)
            {
                if (need[i][j] > available[j])
                {   flag = 1;
                    break;
                }
            }
            if (flag == 0)
            {
                ans[ind++] = i;
                for (y=0; y<m; y++)
                {   available[y] += allocation[i][y];
                }
                f[i] = 1;
            }
        }
    }
```

```
int flag = 1;
for( i=0; i<n; i++)
{   if ( f[i] == 0)
    {
        flag = 0;
        printf("The following System is not safe");
        break;
    }
}
if ( flag == 1)
{   printf("Following is the safe sequence\n");
    for(i=0; i<n; i++)
    {   printf("P%d -> ", ans[i]);
    }
    printf("P%d\n", ans[n-1]);
}
return 0;
}
```

Output

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the Allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

Enter the MAX matrix:

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the Available Resources:

3 3 2

Following is the Safe Sequence

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

8 Write a C program to stimulate deadlock detection.

→
```
#include<stdio.h>
Static int mark[20];
int i,j,np,nr;
int main()
{
    int alloc[10][10], request[10][10],
    avail[10], r[10], w[10];
    printf("Enter the no of process");
    scanf("%d",&nr);
    for(i=0; i<nr; i++)
    {
        printf("\n Total Amount of the
Resource R %d: ", i+1);
        scanf("%d", &r[i]);
    }
    printf("\n Enter the request matrix:");
    for(i=0; i<np; i++)
    for(j=0; j<nr; j++)
    scanf("%d", &request[i][j]);
    printf("Enter the allocation matrix");
    for(i=0; i<np; i++)
    for(j=0; j<nr; j++)
    scanf("%d", &alloc[i][j]);
```

```
for(j=0; j<nr; j++)
{   avail[j] = r[j];
    for(i=0; i<np; i++)
    {
        avail[j] -= alloc[i][j];
    }
}
for(i=0; i<np; i++)
{   int count=0;
    for(j=0; j<nr; j++)
    {   if(alloc[i][j]==0)
            count++;
        else
            break;
    }
    if(count==nr)
    mark[i]=1;
}
for(j=0; j<nr; j++)
    w[j] = avail[j];
for(i=0; i<np; i++)
{ int canbeprocessed=0;
  if[mark[i]!=1)
  {   if(request[i][j] <= w[j])
          canbeprocessed=1;
      else
      {   canbeprocessed=0;
          break;
      }
}
```

```
if ( canbe processed )
{  mark[i] = 1
    for ( j = 0; j < nr; j++)
        w[j] += alloc[i][j];
    }
}
int deadlock = 0;
for (i=0; i<np; i++)
 if (mark[i] != 1)
     deadlock = 1;
 if (deadlock)
    printf("\n Deadlock detected");
 else
    printf("\n No deadlock ~~det~~ possible");
}
```

OUTPUT :

Enter no of processors : 5
Enter no of resources : 3
Total amount of the resource R_1 : 5
Total amount of the resource R_2 : 3
Total amount of the resource R_3 : 3
Enter the request matrix :

1	0	1
2	2	2
3	3	3
5	2	1
1	4	2

Enter the allocation matrix

2 1 2
3 3 3
4 4 4
6 3 2
2 5 3

Deadlock detected

19/6/24

8 Write a C program to stimulate the following contiguous memory allocation techniques

a) worst fit

b) Best fit

c) First-fit

→

```
#include <stdio.h>
#define max 25
void firstfirst(int b[], int nb, int f[], int nf);
void worstfirst(int b[], int nb, int f[], int nf);
void bestfirst(int b[], int nb, int f[], int nf);
int main()
{ int b[max], f[max], nb, nf;
  printf("Memory Management Schemes\n");
  printf("\n Enter the number of blocks:");
  scanf("%d", &nb);
  printf("\n Enter the number of files:");
  scanf("%d", &nf);
  printf("\n Enter the size of the blocks:\n");
  for(int i=1; i<=nb; i++)
  { printf("Block %d:", i);
    scanf("%d", &b[i]);
  } printf("Enter the size of the files:\n");
  for(int i=1; i<=nf; i++)
  { printf("File %d:", i);
    scanf("%d", &f[i]);
  } printf("\n Memory Management Scheme -
                First Fit");
```

```
firstfit(b,nb,f,nf);
printf("\nMemory Management Scheme - Worst Fit");
worstfit(b,nb,f,nf);
return 0;
}
voidfirstfit(int b[], int nb, int f[], int nf)
{
    int bf[max]={0};
    int ff[max]={0};
    int frag[max], i, j;
    for(i=1; i<=nf; i++)
    {
        for(j=1; j<=nb; j++)
        {
            if(bf[j]!=1 && b[j]>=f[i])
            {
                ff[i]=j;
                bf[j]=1;
                frag[i]=b[j]-f[i];
                break;
            }
        }
    }
    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for(i=1; i<=nf; i++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    }
}
```

```
void worstfit (int b[], int nb, int f[], int nf)
{ int bf[max] = {0};
  int ff[max] = {0};
  int frag[max], i, j, temp, highest = 0;
  for (i=1; i<=nf; i++)
  { for (j=1; j<=nb; j++)
    { if (bf[j] != 1)
      { temp = b[j] - f[i];
        if (temp >= 0 && highest < temp)
        { ff[i] = j;
          highest = temp;
        }
      }
    }
    frag[i] = highest;
    bf[ff[i]] = 1;
    highest = 0;
  }
  printf("\n File_no:\t File_size:\t Block_no:\t
Block_size:\t Fragment");
  for (i=1; i<=nf; i++)
  {   printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",
    i, f[i], ff[i], b[ff[i]], frag[i]);
  }
}

void bestfit (int b[], int nb, int f[], int nf)
{
```

```
int bf[max] = {0}
int ff[max] = {0}
int frag[max], i, j, temp, lowest = 10000;
for (i=1; i<=nf; i++)
{
    for (j=1; j<=nb; j++)
    {
        if (bf[j] != 1)
        {
            temp = b[j] - f[i];
            if (temp>=0 && lowest > temp)
            {
                ff[i] = j;
                lowest = temp;
            }
        }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
}
printf("\nfile_no:\tfile_size:\tBlock_no:\tBlock_size:\tFragment");
for (i=1; i<=nf && ff[i]!=0; i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT

Enter the number of blocks

5

Enter the number of processes

5

Enter the block size

100 500 900 300 600 10 12 15 14 13

Enter the ~~process~~ size of the files:

~~File~~ 1: 52 53 54 56 58

Memory Management Scheme - First fit

File no:	File_size	Block no.	Block size	Fragment
1	52	0	0	0
2	53	0	0	0
3	54	1	15	10
4	56	0	0	0
5	58	0	0	0

Memory — Best fit / Worst fit

File No	File size	Block no	Block Size	Fragment
1	52	0	0	0
2	53	0	0	0
3	5	1/5	10/15	5/10
4	56	0	0	0
5	58	0	0	0

Q Write a C program to stimulate page Deplacement algorithm:

a) FIFO

b) LRU

c) optimal

→ #include <stdio.h>

```
int n, f, i, j, k;
int in[100];
int p[50];
int hit = 0;
int pgfaultcnt = 0;
void getData()
{ printf("\n Enter length of page reference sequence");
  scanf("%d", &n);
  printf("\n Enter the page reference Sequence:");
  for(i=0; i<n; i++)
  scanf("%d", &in[i]);
  printf("\n Enter no of frames:");
  scanf("%d", &f);
}
void initialize()
{ pgfaultcnt = 0;
  for(i=0; i<f; i++)
    p[i] = 9999;
}
```

```
int isHit(int data)
{ hit = 0;
  for (j=0; j<f; j++
  { if (p[j] == data)
    { hit=1;
      break;
    }
  } return hit;
}
int getHitIndex(int data)
{ int hitind;
  for (k=0; k<f; k++)
  { if (p[k] == data)
    { hitind = k;
      break;
    }
  } return hitind;
}
void dispPages()
{ for (k=0; k<f; k++)
  { if (p[k] != 9999)
      printf("%d", p[k]);
  }
}
void dispPgFaultCnt()
{ printf("\nTotal no of page faults: %d", pgfaultcnt++);
}
```

```
void fifo()
{   getdata();
    initialize();
    for (int i=0; i<n; i++)
    {   printf("\nfor %d: ", in[i]);
        if (isHit(in[i]) == 0)
        {   for (k=0; k<f-1; k++)
                p[k] = p[k+1];
            p[k] = in[i];
            pgfaultcnt++;
            dispPages();
        } else
            printf("No page fault");
    } dispPgFaultCnt();
}

void optimal()
{   initialize();
    int near[50];
    for (i=0; i<n; i++)
    {   printf("\nfor %d: ", in[i]);
        if (isHit(in[i]) == 0)
        {   for (j=0; j<f; j++)
            {   int pg = p[j];
                int found = 0;
                for (k=i; k<n; k++)
                {   if (pg == in[k])
```

```
        { near [j] = k;
          found = 1;
          break;
        } else
          found = 0
    }
    if (!found)
        near [j] = 9999;
}
int max = -9999;
int repindex;
for (j=0; j<nf; j++)
{   if (near [j] > max)
        {   max = near [j];
            repindex = j;
        }
}
p[repindex] = in[i];
pgfaultcnt++
dispPages();
} else
  printf("No page fault");
} dispPgFaultCnt();
}
```

```
void lru()
{  initialize();
   int last[50];
   for(i=0; i<n; i++)
   {  printf("\nFor %d: ", in[i]);
      if(isHit(in[i])==0)
      {  for(j=0; j<nf; j++)
         {  int pg=p[j];
            int found=0;
            for(k=i-1; k>=0; k--)
            {  if(pg==in[k])
               {
                  last[j]=k;
                  found=1;
                  break;
               }else
               found=0;
            }
            if(!found)
               last[j]=-9999;
         }
      int min=9999;
      int repindex;
      for(j=0; j<nf; j++)
      {  if(last[j]<min)
         {  min=last[j];
            repindex=j;
         }
      }
}
```