| Ex.No .9 | INTERPROCESS COMMUNICATION USING PIPES AND SHARED MEMORY |
|---|---|
| 24.04.2024 | |

**AIM:**

To write a C program to implement the interprocess communication using 16shared memory and pipes.

**PROGRAMS:**

**USING PIPES :**

```
#include<stdio.h>
int main()
{
   int fd[2],child;
   char a[10];
   printf("Enter the string to enter into the pipe:");
   scanf("%s",a);
   pipe(fd);
   child=fork();
   if(!child)
   {
      close(fd[0]);
      write(fd[1],a,strlen(a));
      wait(0);
   }
   else
   {
      close(fd[1]);
      read(fd[0],a,10);
      printf("\n The String retrieved from the pipe is: %s\n",a);
   }
   return 0;
}
```

**OUTPUT :**

```
itii@UGIII:~$ ./a.out pipes.c
Enter the string to enter into the pipe: IPC In Linux

 The String retrieved from the pipe is: IPC
itii@UGIII:~$
```

**USING SHARED MEMORY :**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#define SHM_SIZE 1024

int main() {
   int shmid;
   char *shmaddr;
   char buffer[SHM_SIZE];
   pid_t pid;
   shmid = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
   if (shmid < 0) {
      perror("shmget");
      exit(1);
   }
   shmaddr = (char*) shmat(shmid, NULL, 0);
   if (shmaddr == (char*) -1) {
      perror("shmat");
      exit(1);
   }
   printf("Enter a message: ");
   fgets(buffer, SHM_SIZE, stdin);
   strcpy(shmaddr, buffer);
  if (shmdt(shmaddr) < 0) {
      perror("shmdt");
```

```
exit(1);
   }
   pid = fork();

 if (pid < 0) {
      perror("fork");
      exit(1);
   }
   else if (pid == 0) {
      char *shmaddr_child;
      shmaddr_child = (char*) shmat(shmid, NULL, 0);
      if (shmaddr_child == (char*) -1) {
         perror("shmat");
         exit(1);
      }
      printf("Message received by child process: %s", shmaddr_child)
      if (shmdt(shmaddr_child) < 0) {
         perror("shmdt");
         exit(1);
      }
      exit(0);
   } else {
      wait(NULL);
      if (shmctl(shmid, IPC_RMID, NULL) < 0) {
         perror("shmctl");
         exit(1);
      }
   }

   return 0;
}
```

**OUTPUT:**

```
g2215036@cloudshell:~/x$ gcc sh.c
g2215036@cloudshell:~/x$ ./a.out sh.c
Enter some data to write to shared memory
hello everyone
You wrote : hello everyone

Data read from shared memory is : hello everyone

g2215036@cloudshell:~/x$
```

**RESULT :**

Thus the program to implement the interprocess communication using sharmemory was written , executed and verified.

| Ex.No .8 | IMPLEMENTATION OF DEADLOCK AVOIDANCE USING SEMAPHORES |
|---|---|
| 17.04.2024 | |

**AIM:**

To write a C program to implement the detection of deadlocks.

**PROGRAM:**

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int request[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];
int n_processes, n_resources;
int detect_cycle(int graph[][MAX_RESOURCES], int visited[], int start, int parent) {
   int i;
   visited[start] = 1;
   for (i = 0; i < n_processes; i++) {
      if (graph[start][i]) {
         if (!visited[i]) {
            if (detect_cycle(graph, visited, i, start)) {
               return 1;
            }
         } else if (i != parent) {
            return 1;
         }
      }
   }
   return 0;
}
void detect_deadlock() {
   int i, j;
   int graph[MAX_PROCESSES][MAX_PROCESSES] = {0};
```

```
    int visited[MAX_PROCESSES] = {0};
    for (i = 0; i < n_processes; i++) {

        for (j = 0; j < n_resources; j++) {
            graph[i][j] = request[i][j] && available[j];
        }
    }
    if (detect_cycle(graph, visited, 0, -1)) {
        printf("Deadlock detected.\n");
    } else {
        printf("No deadlock detected.\n");
    }
}

int main() {
    int i, j;
    printf("Enter the number of processes: ");
    scanf("%d", &n_processes);
    printf("Enter the number of resources: ");
    scanf("%d", &n_resources);
    printf("Enter the allocation matrix:\n");
    for (i = 0; i < n_processes; i++) {
        for (j = 0; j < n_resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Enter the request matrix:\n");
    for (i = 0; i < n_processes; i++) {
        for (j = 0; j < n_resources; j++) {
            scanf("%d", &request[i][j]);
        }
    }
    printf("Enter the available vector:\n");
    for (i = 0; i < n_resources; i++) {
        scanf("%d", &available[i]);
    }
    detect_deadlock();
    return 0;
    }
```

**OUTPUT:**

```
g2215036@cloudshell:~/karthicka$ gcc dl.c
g2215036@cloudshell:~/karthicka$ ./a.out dl.c
Enter the number of processes: 4
Enter the number of resources: 2
Enter the allocation matrix:
1 0
0 1
1 1
0 0
Enter the request matrix:
0 1
1 0
0 0
1 1
Enter the available vector:
1 1
No deadlock detected.
g2215036@cloudshell:~/karthicka$
```

**RESULT :**

Thus the program to implement the detection of deadlocks in C was written,executed and verified

| Ex.No :7 | IMPLEMENTATION OF PRODUCER CONSUMER PROBLEM USING SEMAPHORES |
|---|---|
| 10.04.2024 | |

**AIM :**

To write a C program to solve the producer consumer problem.

**PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  printf("PROBLEMS OF THE CRITICAL SECTION - PRODUCER CONSUMER
  \n"); int full=0,empty=3,mutex=1;
  int count=0;
  int choice;
  if(empty!=0)
  {
  while(1)
  {
    printf("\nPRODUCER - 1 || CONSUMER - 2 || EXIT - 3\n");
    printf("\nEnter your choice: \n\n");
    scanf("%d",&choice);
    if(choice==1)
    {
      if((mutex==1)&&(empty!=0))
      {
        mutex-=1,full+=1,empty-=1,count++;
        printf("\nThe producer can produce...\n");
        mutex+=1;
        printf("\nThe number of items in the buffer is %d \n",count);
      }
      else
      {
```

```
            printf("BUFFER IS FULL!!!\n");
          }
        }
        if(choice==2)
        {
          if((mutex==1)&&(full!=0))
          {
            mutex-=1,full-=1,empty+=1,count--;
            printf("\nThe Consumer can consume...\n");
            mutex+=1;
            printf("\nThe number of items in the buffer is %d \n",count);
          }
          else
          {
            printf("BUFFER IS EMPTY!!!\n");
          }
        }
        if(choice==3)
        {
          printf("\nTHANK YOU!\n");
          exit(1);
        }
    }
}
```

**OUTPUT:**

```
g2215036@cloudshell:~/karthicka$ ./a.out procon.c
PROBLEMS OF THE CRITICAL SECTION - PRODUCER CONSUMER

PRODUCER - 1 || CONSUMER - 2 || EXIT - 3

Enter your choice:

1

The producer can produce...

The number of items in the buffer is 1

PRODUCER - 1 || CONSUMER - 2 || EXIT - 3

Enter your choice:

2

The Consumer can consume...

The number of items in the buffer is 0

PRODUCER - 1 || CONSUMER - 2 || EXIT - 3

Enter your choice:

2
BUFFER IS EMPTY!!!

PRODUCER - 1 || CONSUMER - 2 || EXIT - 3

Enter your choice:

3

THANK YOU!
```

**RESULT:**

Thus the program to solve the producer consumer problem was executed and verified.

| Ex.No : 10 | IMPLEMENTATION OF MEMORY MANAGEMENT SCHEMES |
|---|---|
| 08.05.2024 | |

**AIM:**

To write a C program to implement the memory management schemes such as first fit,best fit and worst first.

**PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
int bestfit(int b, int sb[], int p, int ps[])
{
   printf("\nBEST-FIT\n");
   printf("Block
   number\tBlocksize\tProcessID\tPsize\tFragmentation\n"); int all[p],
   allocated[b];
   for(int i = 0; i < b; i++) {
      allocated[i] = 0;
   }
   for(int j = 0; j < p; j++) {
      int min = 99999, pos = -1;
      for(int i = 0; i < b; i++) {
         if(sb[i] >= ps[j] && sb[i] < min && !allocated[i]) {
            if(pos == -1 || sb[i] < sb[pos]) {
               min = sb[i];
               pos = i;
            }
         }
      }
      if(pos == -1) {
         all[j] = 0;
      } else {
         all[j] = ps[j];
```

```
            printf(" %d\t\t%d\t\t%d\t\t%d\t\t%d\n", pos, sb[pos], j, all[j], sb[pos] - all[j]);
            sb[pos] -= all[j];
            allocated[pos] = 1;
        }
    }
}
int firstfit(int b,int sb[],int p,int ps[])
{
    printf("\nFIRST-FIT\n");
    printf("Block
    number\tBlocksize\tProcessID\tPsize\tFragmentation\n"); int all[b];
    for(int j=0;j<p;j++)
    {
        int max=0,pos=0;
        for(int i=0;i<b;i++)
        {
            if(sb[i]>max)
            {
                max=sb[i];
                pos=i;
                break;
            }
        }
        if(ps[j]<=max)
        {
            all[j]=ps[j];
        }
        else if(ps[j]>max)
        {
            continue;
        }
        printf(" %d\t\t%d\t\t%d\t\t%d\t\t%d\n",pos,sb[pos],j,all[j],sb[pos]-all[j]);
        sb[pos]=0;
    }
}
int worstfit(int b,int sb[],int p,int ps[])
{
    printf("\nWORST-FIT\n");

    printf("Block
```

```
number\tBlocksize\tProcessID\tPsize\tFragmentation\n"); int all[b];
for(int j=0;j<p;j++)
{
  int max=0,pos=0;
  for(int i=0;i<b;i++)
  {
    if(sb[i]>max)
    {
      max=sb[i];
      pos=i;
    }
  }
  if(ps[j]<=max)
    all[j]=ps[j];
  else if(ps[j]>max)
    continue;
  printf(" %d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",pos,sb[pos],j,all[j],sb[pos]-all[j]);
  sb[pos]=0;
 }
}
int main()
{
 int b,p;
 printf("Enter the number of blocks:\n");
 scanf("%d",&b);
 int sb[b];
 printf("Enter the size of each block : \n");
 for(int i=0;i<b;i++)
 {
  scanf("%d",&sb[i]);
 }
 printf("Enter the number of processes:\n");
 scanf("%d",&p);
 int ps[b];
 printf("Enter the size of each process: \n");

 for(int i=0;i<p;i++)
  scanf("%d",&ps[i]);
 int choice;
 printf("Enter your choice :\n1. Worst Fit\n2. First Fit\n3. Best
 Fit\n"); scanf("%d",&choice);
```

```
    switch(choice)
    {
       case 1:
          worstfit(b,sb,p,ps);
          break;
       case 2:
          firstfit(b,sb,p,ps);
          break;
       case 3:
          bestfit(b,sb,p,ps);
          break;
       default:
          printf("Invalid choice");
    }
    return 0;
}
```

**OUTPUT:**

**BEST FIT :**

```
itii@UGIII:~/mm$ ./a.out mm2.c
Enter the number of blocks:
5
Enter the size of each block :
200 600 100 25 10
Enter the number of processes:
4
Enter the size of each process:
5 300 550 4
Enter your choice :
1. Worst Fit
2. First Fit
3. Best Fit
3

BEST-FIT
Block number     Blocksize        ProcessID        Psize    Fragmentation
 4               10               0                5                5
 1               600              1                300              300
 3               25               3                4                21
itii@UGIII:~/mm$
```

**FIRST FIT :**

```
itii@UGIII:~/mm$ ./a.out mm2.c
Enter the number of blocks:
5
Enter the size of each block :
200 600 100 25 10
Enter the number of processes:
4
Enter the size of each process:
5 300 550 4
Enter your choice :
1. Worst Fit
2. First Fit
3. Best Fit
2

FIRST-FIT
Block number     Blocksize        ProcessID                Psize    Fragmentation
 0               200              0                5               195
 1               600              1                300             300
 2               100              3                4               96
itii@UGIII:~/mm$
```

**WORST FIT :**

```
Enter the size of each block :
200 600 100 25 10
Enter the number of processes:
4
Enter the size of each process:
5 300 550 4
Enter your choice :
1. Worst Fit
2. First Fit
3. Best Fit
1

WORST-FIT
Block number    Blocksize       ProcessID       Psize   Fragmentation
 1              600             0               5               595
 0              200             3               4               196
```

**RESULT:**

Thus the program to implement the memory management schemes was written,executed and verified.