LAB Evaluation -3

<u>Assignment for Set-B</u>

CSE325 (Operating Systems Laboratory)

Name :- Ashwani Kumar   Roll No. :- B-38   Section :- K18PG   Reg No :- 11806645

Question 1.) Write a program to create a scenario where child process should return its exit status to parent and parent should suspend its execution till child is not terminated. (25)

Answer :- The program to create a scenario where child process should return its exit status to parent and parent should suspend its execution till child is not terminated is given below :-

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>
int main()
{
     pid_t  p;
     p = fork()
     printf("Process execution started");
     switch(p)
     {
            case -1 :
                       printf("\n Error, No Process can  Execute");
                        break;
           case 0 : for(i=0;i<10;i++)
                      {
                      Printf("%d",i);
                      Sleep(1);
                       }
                     printf("\n Child Process is Created"); // Child is executing till that parent has to wait.
                     printf("\n Child process terminated and The Child Process ID has : %d",getpid()); // ID of the
                                                                                                   //Child Process
                       printf("\n I am Parent Process: %d",getppid()); //    ID of the Parent  Process
                 break;
          default:
                       wait(NULL);
                      for(j=20;i<30;i++)
                      {
                      Printf("%d",j);
                      Sleep(1);
                       }
                       printf("\n Parent process terminated and The Parent Process ID is :%d",getpid());
                       printf("\nMy Child is having ID :%d",p); //ID of the Child Process
                    break;
      }
}
```

Question 2.) Implement the Reader-Writer problem using semaphores. (25)

Answer :- The Data Representation of Reader-Writer problem using Semaphores is given below:-

Three variables are used: **mutex, wrt, readcnt to implement solution**

1. **semaphore mutex, wrt;** // semaphore mutex is used to ensure mutual exclusion when readcnt is updated i.e. when any reader enters or exit from the critical section and semaphore wrt is used by both readers and writers
2. **int readcnt;** // readcnt tells the number of processes performing read in the critical section, initially 0

Functions for sempahore :
– **wait() :** decrements the semaphore value.
– **signal() :** increments the semaphore value.

Reader Process :-

1. Reader requests the entry to critical section.
2. If allowed:
    - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.
    - It then, signals mutex as any other reader is allowed to enter while others are already reading.
    - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore **'wrt'** as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

```
do {
  // Reader wants to enter the critical section
  wait(mutex);
  // The number of readers has now increased by 1
  readcnt++;
  // there is atleast one reader in the critical section
  // this ensure no writer can enter if there is even one reader
  // thus we give preference to readers here
  if (readcnt==1)
    wait(wrt);
  // other readers can enter while this current reader is inside
  // the critical section
  signal(mutex);
  // current reader performs reading here
  wait(mutex);   // a reader wants to leave
  readcnt--;
  // that is, no reader is left in the critical section,
  if (readcnt == 0)
     signal(wrt);        // writers can enter
  signal(mutex); // reader leaves
} while(true);
```

Writer Process :-

1. Writer requests the entry to critical section.
2. If allowed i.e. **wait()** gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

```
do {
   // writer requests for critical section
   wait(wrt);
```

```
    // performs the write
    // leaves the critical section
    signal(wrt);
} while(true);
```

Thus, the semaphore **‘wrt’** is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

The Reader-Writer problem using semaphores in C Language is as given below :-

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
sem_t mutex,writeblock;
int data = 0,rcount = 0;

void *reader(void *arg)
{
  int f;
  f = ((int)arg);
  sem_wait(&mutex);
  rcount = rcount + 1;
  if(rcount==1)
   sem_wait(&writeblock);
  sem_post(&mutex);
  printf("Data read by the reader%d is %d\n",f,data);
  sleep(1);
  sem_wait(&mutex);
  rcount = rcount - 1;
  if(rcount==0)
   sem_post(&writeblock);
  sem_post(&mutex);
}
void *writer(void *arg)
{
       int f;
       f = ((int) arg);
       sem_wait(&writeblock);
       data++;
       printf("Data writen by the writer%d is %d\n",f,data);
       sleep(1);
       sem_post(&writeblock);
}
int main()
{
           int i,b;
          pthread_t rtid[5],wtid[5];
         sem_init(&mutex,0,1);
        sem_init(&writeblock,0,1);
        for(i=0;i<=2;i++)
       {
              pthread_create(&wtid[i],NULL,writer,(void *)i);
```

```
            pthread_create(&rtid[i],NULL,reader,(void *)i);
      }
      for(i=0;i<=2;i++)
     {
           pthread_join(wtid[i],NULL);
          pthread_join(rtid[i],NULL);
      }
}
```

Question 3.) Write a program to create a file and open it in read write mode. Take the input "Lovely Professional University" from the user. Also display the 10- 17 characters of the input on the terminal. **(25)**

Answer :- The Program to create a file and open it in read write mode and Take the input "Lovely Professional University" from the user and display the 10th - 17th characters of the input on the terminal is given below :-

```
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
int main()
{
int n, fd, off;
char buff[50];
printf("enter Lovely Professional University to write into the file\n");
n=read(0,buff,50);
fd=open("ABC",O_CREAT|O_RDWR|O_APPEND,0777); // The File Name “ABC” is created and opened in read-write mode.
write(fd,buff,n);
off=lseek(fd,10,SEEK_CUR);   // It is used for positioning a pointer i.e lseek(SEEK_CUR) a current  location.
read(fd,buff,8);
write(1,buff,8);// it will display the character from 10th – 17th Position.
int close(int fd);
return 0;
}
```

Question 4.) Write a C program to generate a resource allocation graph for deadlock detection. **(25)**

Answer :- The C Program to generate a resource allocation graph for deadlock detection is given below :-

```
#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("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);
printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
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)
{
   for(j=0;j<nr;j++)
    {
      if(request[i][j]<=w[j])
        canbeprocessed=1;
      else
         {
         canbeprocessed=0;
         break;
          }
     }
if(canbeprocessed)
{
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 possible");
}
```