Dayananda Sagar University

School of Engineering
Department of Computer Science & Engineering

List of Experiments

Course: Operating System Lab

Exp. No	Division of Experiments	List of Experiments
1	System Calls	Write a C program to create a new process that exec a new program using system calls fork(), execlp() & wait()
2		Write a C program to display PID and PPID using system calls getpid () & getppid ()
3		Write a C program using I/O system calls open(), read() & write() to copy contents of one file to another file
4	Process Management	Write a C program to implement multithreaded program using pthreads
5		Write C program to simulate the following CPU scheduling algorithms a) FCFS b) SJF c) Priority d) Round Robin
6	Process synchronization	Write a C program to simulate producer-consumer problem using semaphores
7	Deadlock	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.
8		Write a C program to simulate deadlock detection.
9	Memory Management	Write a C program to simulate paging technique of memory management
10		Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) LFU
11	I/O System	Write a C program to simulate the following file organization techniques a) Single level directory b) Two level directory
12		Write a C program to simulate the following file allocation strategies. a) Sequential b) Indexed

1. Write a C program to create a new process that exec a new program using system calls fork(), execlp() & wait()

Program description:

C program (or, process, at run time) needs to create a process which would execute a program. Here, two system calls are of interest, fork and exec. The fork system call in Unix creates a new process. The new process (the ***child*** process) is an exact copy of the calling process (the ***parent*** process). however, you want the new process to run a new program. When, for example, you type "date" on the unix command line, the command line interpreter (the so-called "shell") forks so that momentarily 2 shells are running, then the code in the child process is replaced by the code of the "date" program by using one of the family of exec system calls **execlp().** The parent process waits for the child to exit.

System calls used

1. fork ( )

fork() is used to create new processes. The new process consists of a copy of the address space of the original process. Upon successful completion, **fork() returns** 0 to the child process and **returns** the process ID of the child process to the parent process.

Syntax:

```
#include <unistd.h>
int fork( );
```

2. execlp ( )

Used after the fork () system call by one of the two processes to replace the process memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution. The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.

Syntax:

```
#include <unistd.h>
int execlp(const char *file, const char *arg, ...);
```

The initial argument is the name of a file that is to be executed. The next arguments describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The list of arguments *must*be terminated by a NULL pointer,

3. wait ( )

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax:

```
#include <sys/wait.h>
pid_t wait( int* status );
```

4. exit( )

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).

Syntax:

```
#include <std1ib.h>
void exit(int status);
```

Program Code:

```
 #include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
{
int pid;
pid=fork();
if(pid<0)
{
printf("fork failed");
exit(1);
}
else if(pid==0)
{
printf("\nNow in Child Process and it's o/p is \n");
execlp("ls","ls",NULL);
exit(0);
}
else
{
printf("\nChild Process created successfully\n");
printf("\nIt's Process id is %d\n",getpid());
wait(NULL);
printf("\nReturn back to Parent process, now ready to exit\n");
exit(0);
}
}
```

2. Write a C program to display PID and PPID using system calls getpid () & getppid ()

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
int pid;
pid=fork();
if(!pid)
{
printf("Child process...");
printf("\n\nChild PID : %d",getpid());
printf("\nParent PID : %d",getppid());
printf("\n\nFinished with child\n");
}
else
{
wait(NULL);
printf("\nParent process");
printf("\nPARENT PID : %d",getpid());
printf("\nChild PID : %d",pid);
}
}
```

3.Write a C program using I/O system calls open(), read() & write() to copy contents of one file to another file

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<fcntl.h>
void main()
{
char buff;
int fd,fd1;
fd=open("one.txt",O_RDONLY);
fd1=open("two.txt",O_WRONLY|O_CREAT);
while(read(fd,&buff,1))
write(fd1,&buff,1);
printf("The copy of a file is successed");
close(fd);
close(fd1);
}
```

4. Write a C program to implement multithreaded program using pthreads

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

// Let us create a global variable to change it in threads
int g = 0;

// The function to be executed by all threads
void *myThreadFun(void * vargp)
{
   // Store the value argument passed to this thread
   int *myid = (int *)vargp;

   // Let us create a static variable to observe its changes
   static int s = 0;

   // Change static and global variables
   ++s; ++g;

   // Print the argument, static and global variables
   printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}

int main()
{
   int i;
   pthread_t tid;

   // Let us create three threads
   for (i = 0; i < 3; i++)
      pthread_create(&tid, NULL, myThreadFun, (void *)&tid);

   pthread_exit(NULL);
   return 0;
}
```

Output:

```
gcc multithread.c -lpthread
./a.out
Thread ID: 3, Static: 2, Global: 2
Thread ID: 3, Static: 4, Global: 4
Thread ID: 3, Static: 6, Global: 6
```

5.Write C program to simulate the Round robin CPU scheduling algorithms

```
#include<stdio.h>
void main()
{
   // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time quantum
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
```

```
        temp[i] = temp[i] - quant;
        sum = sum + quant;
    }
    if(temp[i]==0 && count==1)
    {
        y--; //decrement the process no.
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
        wt= wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_tat);
printf("\n Average Waiting Time: \t%f", avg_wt);
}
```

Output:

Total number of process in the system: 5

Enter the Arrival and Burst time of the Process[1]
Arrival time is: 0
Burst time is: 5

Enter the Arrival and Burst time of the Process[2]
Arrival time is: 1
Burst time is: 3

Enter the Arrival and Burst time of the Process[3]
Arrival time is: 2
Burst time is: 1

Enter the Arrival and Burst time of the Process[4]
Arrival time is: 3
Burst time is: 2

Enter the Arrival and Burst time of the Process[5]
Arrival time is: 4
Burst time is: 3
Enter the Time Quantum for the process: 2

Process No	Burst Time	TAT	Waiting Time
Process No[3]	1	3	2
Process No[4]	2	4	2
Process No[2]	3	11	8
Process No[5]	3	9	6
Process No[1]	5	14	9

Average Turn Around Time: 8.200000
Average Waiting Time: 5.400000

6.Write a C program to simulate producer-consumer problem using semaphores

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;  //Initialization of semaphores
                                 // mutex = 1
                                 //Full = 0 - Initially, all slots are empty. Thus full slots are 0
                                 //Empty = 3 // All slots are empty initially
void producer();
void consumer();
int wait(int);
int signal(int);
int main()
{
        int n;
        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
                                        printf("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);          //producer has placed the item and thus the value of "full" is
                                    increased by 1
        empty=wait(empty);  //producer produces an item then the value of "empty" is
                        reduced  by 1
                                  //because one slot will be filled now

        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);                  //consumer is removing an item from buffer, therefore the
                                            value of
                                            //"full" is reduced by 1
        empty=signal(empty);       //consumer has consumed the item, thus increasing the
                                    value of
                                          //"empty" by 1
        printf("\nConsumer 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:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
```

5.Write a C program to simulate producer-consumer problem using semaphores

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;  //Initialization of semaphores
                                 // mutex = 1
                                 //Full = 0 - Initially, all slots are empty. Thus full slots are 0
                                 //Empty = 3 // All slots are empty initially
void producer();
void consumer();
int wait(int);
int signal(int);
int main()
{
        int n;
        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
                                        printf("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=signall(full);          //producer has placed the item and thus the value of "full" is
                                increased by 1
        empty=wait(empty);        //producer produces an item then the value of "empty" is reduced
                                        by 1  //because one slot will be filled now
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer 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:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3

7. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10],
completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;
printf("Enter the no of processes : ");
scanf("%d", &p);
for(i = 0; i< p; i++)
completed[i] = 0;
printf("\n\nEnter the no of resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each process :
"); for(i = 0; i < p; i++)
{
printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &Max[i][j]);
}
printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ",i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]);
}
printf("\n\nEnter the Available Resources : ");
for(i= 0; i < r; i++)
scanf("%d", &avail[i]);
for(i = 0; i < p; i++)
for(j = 0; j < r; j++)
need[i][j] = Max[i][j] - alloc[i][j];
do
{
printf("\n Max matrix:\tAllocation matrix:\n");
for(i = 0; i < p; i++)
{
for( j = 0; j < r; j++)
printf("%d ", Max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");
}
process = -1;
for(i = 0; i < p; i++)
```

```
{
if(completed[i] == 0)//if not completed
{
process = i ;
for(j = 0; j < r; j++)
{
if(avail[j] < need[i][j])
{
process = -1;
break;
}
}
}
if(process != -1)
break;
}
if(process != -1)
{
printf("\nProcess %d runs to completion!", process + 1);
safeSequence[count] = process + 1; count++;
for(j = 0; j < r; j++)
{
avail[j] += alloc[process][j];
alloc[process][j] = 0;
Max[process][j] = 0;
completed[process] = 1;
}
}
}
while(count != p && process != -1);
if(count == p)
{
printf("\nThe system is in a safe state!!\n");
printf("Safe Sequence : < "); for( i = 0; i < p; i++)
printf("%d ", safeSequence[i]);
printf(">\n");
}
else
printf("\nThe system is in an unsafe state!!");
}
```

Output:

Enter the no of processes : 5
Enter the no of resources : 3
Enter the Max Matrix for each process :
For process 1 : 7
5
3
For process 2 : 3
2
2
For process 3 : 7
0
2
For process 4 : 2
2
2
For process 5 : 4
3
3
Enter the allocation for each process :
For process 1 : 0
1
0
For process 2 : 2
0
0
For process 3 : 3
0
2
For process 4 : 2
1
1
For process 5 : 0
0
2
Enter the Available Resources : 3
3
2
Max matrix: Allocation matrix:
7 5 3 0 1 0
3 2 2 2 0 0
7 0 2 3 0 2
2 2 2 2 1 1
4 3 3 0 0 2
Process 2 runs to completion!
Max matrix: Allocation matrix:
7 5 3 0 1 0
0 0 0 0 0 0
7 0 2 3 0 2
2 2 2 2 1 1
4 3 3 0 0 2
Process 3 runs to completion!
Max matrix: Allocation matrix:

7 5 3 0 1 0
0 0 0 0 0 0
0 0 0 0 0 0
2 2 2 2 1 1
4 3 3 0 0 2
Process 4 runs to completion!
Max matrix: Allocation matrix:
7 5 3 0 1 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
4 3 3 0 0 2
Process 1 runs to completion!
Max matrix: Allocation matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
4 3 3 0 0 2
Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence: < 2 3 4 1 5 >

8. Write a C program to simulate FIFO page replacement algorithm.

```
#include<stdio.h>
int main()
{
  int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
  for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
  for(i=0;i<no;i++)
     frame[i]= -1;
  j=0;
printf("\tref string\t page frames\n");
  for(i=1;i<=n;i++)
    {
printf("%d\t\t",a[i]);
       avail=0;
       for(k=0;k<no;k++)
           if(frame[k]==a[i])
               avail=1;
           if (avail==0)
             {
                frame[j]=a[i];
                j=(j+1)%no;
                count++;
                for(k=0;k<no;k++)
printf("%d\t",frame[k]);
             }
```

```
printf("\n");
    }
printf("Page Fault Is %d",count);
   return 0;
```

OUTPUT:

```
ENTER THE NUMBER OF PAGES:  20
ENTER THE PAGE NUMBER :       7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
ENTER THE NUMBER OF FRAMES :3

 ref string      page frames
         7           7     -1     -1
         0           7     0      -1
         1           7     0      1
         2           2     0      1
         0
         3           2     3      1
         0           2     3      0
         4           4     3      0
         2           4     2      0
         3           4     2      3
         0           0     2      3
         3
         2
         1           0     1      3
         2           0     1      2
         0
         1
         7           7     1      2
         0           7     0      2
         1           7     0      1

Page Fault Is 15
```

9. Write a C program to simulate the Single level directory file organization technique.

```
#include<stdio.h>
#include<string.h>
int main()
   {
      int nf=0,i=0,j=0,ch;
      char mdname[10],fname[10][10],name[10];
printf("\nEnter the directory name:");
scanf("%s",mdname);
printf("\nEnter the number of files:");
scanf("%d",&nf);
      do
      {
printf("\nEnter file name to be created:");
scanf("%s",name);
         for(i=0;i<nf;i++)
         {
            if(!strcmp(name,fname[i]))
            break;
         }
         if(i==nf)
         {
strcpy(fname[j++],name);
nf++;
         }
         else
printf("\nThere is already %s\n",name);
printf("\nDo you want to enter another file(yes - 1 or no - 0):");
scanf("%d",&ch);
      }
      while(ch==1);
```

```
printf("\n\nDirectory name is:%s\n",mdname);
printf("\nFiles names are:");
        for(i=0;i<j;i++)
printf("\n%s",fname[i]);
        return 0;
    }
```

OUTPUT:

```
Enter the directory name:SSS
Enter the number of files:3
Enter file name to be created:aaa
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:bbb
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:ccc
Do you want to enter another file(yes - 1 or no - 0):0

Directory name is:SSS

Files names are:
aaa
bbb
ccc
```

10. Write a C program to simulate the indexed file allocation strategy.

```
#include<stdio.h>
#include<stdlib.h>
int main()
 {
   int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
   for(i=0;i<50;i++)
      f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
   if(f[ind]!=1)
   {
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
   }
   else
   {
printf("%d index is already allocated \n",ind);
goto x;
   }
   y: count=0;
   for(i=0;i<n;i++)
   {
scanf("%d", &index[i]);
   if(f[index[i]]==0)
      count++;
   }
   if(count==n)
   {
      for(j=0;j<n;j++)
         f[index[j]]=1;
printf("Allocated\n");
```

```
printf("File Indexed\n");
        for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
    }
    else
    {
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
    }
printf("Do you want to enter more file(Yes - 1/No - 0): ");
scanf("%d", &c);
    if(c==1)
goto x;
    else
exit(0);
    return 0;
}
```

OUTPUT:

```
Enter the index block: 5
Enter no of blocks needed and no of files for the index 5 on the disk :
4
1 2 3 4
Allocated
File Indexed
5-------->1 : 1
5-------->2 : 1
5-------->3 : 1
5-------->4 : 1
Do you want to enter more file(Yes - 1/No - 0): 1
Enter the index block: 4
4 index is already allocated
Enter the index block: 6
Enter no of blocks needed and no of files for the index 6 on the disk :
2
7 8
Allocated
File Indexed
```

6-------->7 : 1
6-------->8 : 1
Do you want to enter more file(Yes - 1/No - 0): 0