FORK()

The Fork system call is used for creating a new process in Linux

Below are different values returned by fork().

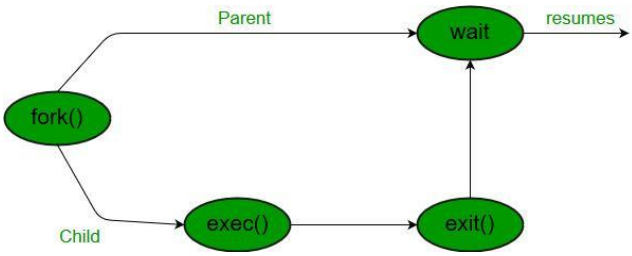
Negative Value: The creation of a child process was unsuccessful.

Zero: Returned to the newly created child process.

Positive value: Returned to parent or caller. The value contains the process ID of the newly created child process.

WAIT()

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.



EXECLP()

The execlp() function replaces the current process image with a new process image specified by file.

1)Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process,

terminate process)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
void main()
```

```
{
int pid;
pid=fork();
if(pid<0)
printf("error");
if(pid==0)
execlp("ls","ls","-l",NULL);
else
{
wait(NULL);
printf("child complete");
}
}
```

2) Simulate the following CPU scheduling algorithms to find turnaround time and waiting time
a) FCFS

```
#include<stdio.h>
int main()
{
   int  p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;
   float awt=0,atat=0;
   printf("enter no of process you want:");
   scanf("%d",&n);
   printf("enter %d process:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&p[i]);
   }
   printf("enter %d arrival time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&at[i]);
   }
   printf("enter %d burst time:",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&bt[i]);
   }
```

```
    /* calculating 1st ct */
   ct[0]=at[0]+bt[0];
   /* calculating 2 to n ct */
   for(i=1;i<n;i++)
   {
    //when proess is ideal in between i and i+1
    temp=0;
   if(ct[i-1]<at[i])
   {
     temp=at[i]-ct[i-1];
   }
   ct[i]=ct[i-1]+bt[i]+temp;
   }
   /* calculating tat and wt */
   printf("\np\t A.T\t B.T\t C.T\t TAT\t WT");
   for(i=0;i<n;i++)
   {
   tat[i]=ct[i]-at[i];
   wt[i]=tat[i]-bt[i];
   atat+=tat[i];
   awt+=wt[i];
   }
   atat=atat/n;
   awt=awt/n;
   for(i=0;i<n;i++)
   {
    printf("\nP%d\t %d\t %d\t %d \t %d \t %d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
   }
   printf("\naverage turn around time is %f",atat);

   printf("\naverage wating time is %f",awt);
   return 0;
}
```

FCFS SCHEDULING ALGORITHM

OUTPUT

```
Enter the number of processes:
4
Enter process id of all the processes:
1
2
3
4
Enter burst time of all the processes:
```

10
5
20
15

Process ID	Burst Time	Waiting Time	TurnAround Time
1	10	0	10
2	5	10	15
3	20	15	35
4	15	35	50

Avg. waiting time= 15.000000
Avg. turnaround time= 27.500000

3) Develop a C program to simulate producer-consumer problem using semaphores.

PRODUCER - CONSUMER PROBLEM

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1) {
printf("\nENTER YOUR CHOICE\n");
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;
```

```
}
}
}
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("\nproducer produces the 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 item1
ENTER YOUR CHOICE
1
producer produces the item2
ENTER YOUR CHOICE
1
producer produces the item3
ENTER YOUR CHOICE
1
```

```
BUFFER IS FULL
ENTER YOUR CHOICE
2
consumer consumes item3
ENTER YOUR CHOICE
2
consumer consumes item2
ENTER YOUR CHOICE
2
consumer consumes item1
ENTER YOUR CHOICE
2
BUFFER IS EMPTY
```

5)Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

```
#include <stdio.h>
int main()
{
	// P0, P1, P2, P3, P4 are the names of Process

	int n, r, i, j, k;
	n = 5; // Indicates the Number of processes
	r = 4; //Indicates the Number of resources
	int alloc[5][4] = { { 0,0,1,2 },	// P0 // This is Allocation Matrix
			{ 1,0,0,0 },	// P1
			{ 1,3,5,4 },	// P2
			{ 0,6,3,2 },	// P3
			{ 0,0,1,4 } };	// P4

	int max[5][4] = { { 0,0,1,2 },	// P0 // MAX Matrix
			{ 1,7,5,0 },	// P1
			{ 2,3,5,6 },	// P2
			{ 0,6,5,2 },	// P3
			{ 0,6,5,6 } };	// P4

	int avail[4] = { 1,5,2,0 }; // These are Available Resources

	int f[n], ans[n], ind = 0;
	for (k = 0; k < n; k++) {
		f[k] = 0;
	}
	int need[n][r];
	for (i = 0; i < n; i++) {
		for (j = 0; j < r; 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 < r; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
                                                break;
                                        }
                                }

                                if (flag == 0) {
                                        ans[ind++] = i;
                                        for (y = 0; y < r; y++)
                                                avail[y] += alloc[i][y];
                                        f[i] = 1;
                                }
                        }
                }
        }

        printf("Th SAFE Sequence is as follows\n");
        for (i = 0; i < n - 1; i++)
                printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);

        return (0);

}
```

Th SAFE Sequence is as follows

P0 -> P2 -> P3 -> P4 -> P1

6) Develop a C program to simulate the following contiguous memory allocation Techniques:

First fit.

```
#include<stdio.h>
void main()
```

```
{
	int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
	for(i = 0; i < 10; i++)
	{
		flags[i] = 0;
		allocation[i] = -1;
	}
	printf("Enter no. of blocks: ");
	scanf("%d", &bno);
	printf("\nEnter size of each block: ");
	for(i = 0; i < bno; i++)
		scanf("%d", &bsize[i]);
	printf("\nEnter no. of processes: ");
	scanf("%d", &pno);
	printf("\nEnter size of each process: ");
	for(i = 0; i < pno; i++)
		scanf("%d", &psize[i]);
	for(i = 0; i < pno; i++)	//allocation as per first fit
		for(j = 0; j < bno; j++)
			if(flags[j] == 0 && bsize[j] >= psize[i])
			{
				allocation[j] = i;
				flags[j] = 1;
				break;
			}
	//display allocation details
	printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
	for(i = 0; i < bno; i++)
	{
		printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
		if(flags[i] == 1)
			printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
		else
			printf("Not allocated");
	}
}
```

```
Enter no. of blocks: 3
Enter size of each block:
10
20
30
Enter no. of processes:
3
Enter size of each process:
15
```

20
40

Block no.	size	process no.	size
1	10	Not allocated	
2	20	1	15
3	30	2	20

7) Develop a C program to simulate page replacement algorithms:

a) FIFO b)LRU

7a) 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
```

7b) LRU PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
```

```
c++;
k++;
for(i=1;i<n;i++)
	{
		c1=0;
		for(j=0;j<f;j++)
		{
			if(p[i]!=q[j])
			c1++;
		}
		if(c1==f)
		{
			c++;
			if(k<f)
			{
				q[k]=p[i];
				k++;
				for(j=0;j<k;j++)
				printf("\t%d",q[j]);
				printf("\n");
			}
			else
			{
				for(r=0;r<f;r++)
				{
					c2[r]=0;
					for(j=i-1;j<n;j--)
					{
					if(q[r]!=p[j])
					c2[r]++;
					else
					break;
				}
			}
			for(r=0;r<f;r++)
			 b[r]=c2[r];
			for(r=0;r<f;r++)
			{
				for(j=r;j<f;j++)
				{
					if(b[r]<b[j])
					{
						t=b[r];
						b[r]=b[j];
						b[j]=t;
					}
				}
			}
			for(r=0;r<f;r++)
			{
				if(c2[r]==b[0])
				q[r]=p[i];
				printf("\t%d",q[r]);
```

```
                        }
                        printf("\n");
                }
        }
}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT:

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3
    7
    7   5
    7   5   9
    4   5   9
    4   3   9
    4   3   7
    9   3   7
    9   6   7
    9   6   2
    1   6   2

The no of page faults is 10
```

8) Simulate following File Organization Techniques a) Single level directory b) Two level directory

8a) SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
```

```
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
```

```
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
getch();
}
```

OUTPUT:

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC

File ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2

Enter the name of the file – B
File B is deleted

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5

8b) TWO LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
```

```
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
```

```
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
getch();
}
```

OUTPUT:

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR1
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1
Enter name of the file -- A1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1
Enter name of the file -- A2
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR2
Enter name of the file -- B1
File created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 4

Enter name of the directory – DIR
Directory not found

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 3
Enter name of the directory – DIR1

Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice – 6

9) Develop a C program to simulate the Linked file allocation strategies.

/* Program to simulate linked file allocation strategy */

Program Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
```

```
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

Program Output:

Enter how many blocks already allocated: 3

Enter blocks already allocated: 1 3 5

Enter index starting block and length: 2 2

2-------->1

3 Block is already allocated

4-------->1

Do you want to enter more file(Yes - 1/No - 0)0

10) Develop a C program to simulate SCAN disk scheduling algorithm.