OS Lab

Process

Program under execution.

fork() is used to create a child process.

If fork() value is negative, no child is created.

If fork() value is 0, child is created.

If fork() value is positive, i.e process id is for the parent.

Note: If n times fork() is called, then 2^n times statement below will be printed and (2^n -1) child will be created.

Header Files

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

Example

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

int main()
{
  fork();
  printf("Hello\n");
  return 0;
}
```

Output

```
Hello
Hello
```

Program 1 (Fork Implementation)

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

int main()
{
    fork();
    printf("LINUX\n");
    fork();
    print("UNIX\n");
    fork();
    printf("RED HAT\n");
    return 0;
}
```

Output

```
LINUX
LINUX
UNIX
UNIX
UNIX
UNIX
RED HAT
RED HAT
RED HAT
RED HAT
RED HAT
RED HAT
RED HAT
```

Program 2 (Even Odd Sum)

```
#include<stdio.h> //standard input output
#include<unistd.h> //fork system
#include<stdlib.h> //exit function
```

```
#include<sys/types.h> //call states of a particular system call
#include<sys/wait.h> //call states of a particular system call
#define max 20

int main()
{
    int pid;
    int a[max],n,sumEven=0,sumOdd=0,i,status;

    printf("\nEnter the no of terms in the array:\n");
    scanf("%d",&n);
    printf("Enter the values in the array\n");

    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }

    pid=fork();
    wait(&status);

    if(pid>0)
    {
        for(i=0;i<n;i++)
        {
            if(a[i]%2==0)
            {
                sumEven+=a[i];
            }

        }
        printf("Parent Process\n");
        printf("Sum of even nos = %d\n",sumEven);
        exit(0);
    }
    else
    {
        for(i=0;i<n;i++)
        {
            if(a[i]%2!=0)
            {
                sumOdd+=a[i];
            }

        }
        printf("Child Process\n");
        printf("Sum of odd nos = %d\n",sumOdd);
        exit(0);
    }

    return 0;
}
```

Input

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

Output

```
Child Process
Sum of odd nos = 9
Parent Process
Sum of even nos = 12
```

Program 3 (Wait System Call)

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>

int main()
{
    int pid=fork();

    if(pid==0)
    {
        sleep(5);
        printf("Child process id: %d has parent id: %d\n",getpid(),getppid());
    }
    else if(pid>0)
    {
        wait(NULL);
        printf("Parent process id: %d has grand parent id: %d\n",getpid(),getppid());
    }
    else
    {
        printf("Process not created");
    }

    return 0;
}
```

Output

```
Parent process id: 74755 has grand parent id: 71342
Child process id: 74756 has parent id: 74755
```

Note:

- Wait system call is forcing the parent to wait until one of its child got executed.
- Time process will be waiting for 5 sec.

Orphan Process

An orphan process is **a computer process whose parent process has finished or terminated**, though it remains running itself.

Zombie Process

a zombie process or defunct process is **a process that has completed execution (via the exit system call)** but still has an entry in the process table: it is a process in the "Terminated state".

Note:

- If pip_d=0, we are in a child process.
- If pid_d>0, we are in the parent process.

Program 4 (Orphan Process)

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main()
{
    pid_t p;
    p=fork();

    if(p==0) //child
    {
        sleep(50);
        printf("I am child having PID %d\n",getpid());
        printf("My parent PID is %d\n",getppid());
    }
    else //parent
    {
        printf("I am parent having PID %d\n",getpid());
        printf("My child PID is %d\n",p);
    }

    return 0;
}
```

Output

```
I am parent having PID 79653
My child PID is 79654
I am child having PID 79654
My parent PID is 1075
```

Program 5 (Zombie Process)

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>

int main()
{
    int pid=fork();

    if(pid==0)
    {
        printf("Child process id: %d has Parent id: %d\n",getpid(),getppid());
    }
    else if(pid>0)
    {
        wait(NULL);
        sleep(60);
        printf("Parent process id: %d has Grand Parent id: %d\n",getpid(),getppid());
    }
    else
    {
        printf("Process not created");
    }

    return 0;
}
```

Output

```
Child process id: 78839 has Parent id: 78838
Parent process id: 78838 has Grand Parent id: 71342
```

A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process.

Program 6 (PIPE Implementation)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>

int main()
{
    pid_t pid;
    char arr[100],str[100];
    int fd[2],nbr,nbw;

    pipe(fd); //Creating a pipe
    pid=fork(); //Calling fork to create a child process

    if(pid==0)
    {
        printf("\nEnter a string: ");
        gets(str);
        nbw=write(fd[1],str,strlen(str));
        printf("Child wrote %d bytes\n",nbw);
        exit(0);
    }
    else
    {
        nbr=read(fd[0],arr,sizeof(arr));
        arr[nbr]='\0';
        printf("Parent read %d bytes: %s\n",nbr,arr);
    }

    return 0;
}
```

Output

```
Enter a string: Graphic Era
Child wrote 11 bytes
Parent read 11 bytes: Graphic Era
```

Program 7 (FCFS)

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

int main()
{
    char p[10][5],temp[5];
    int c=0,pt[10],i,j,n;
    float bst=0.0,turn=0.0;

    printf("Enter no of process: ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Enter name of %d process: ",i+1);
        scanf("%s",&p[i]);
        printf("Enter process time: ");
        scanf("%d",&pt[i]);
    }

    printf("\n.....................................................\n");

    for(i=0;i<n;i++)
    {
        printf("\t%s",p[i]);
    }

    printf("\n.....................................................\n");

    for(i=0;i<n;i++)
    {
        printf("\t%d",pt[i]);
    }

    printf("\n.....................................................\n");

    for(i=0;i<n;i++)
    {
        bst+=c;
        turn+=c+pt[i];
        c+=pt[i]
        printf("\t%d",c);
    }
    printf("\n.....................................................\n");
    printf("\n\nAverage time: %f",bst/n);
    printf("\nTurn around time: %f\n", turn/n);

    return 0;
}
```

Input

```
Enter no of process: 3
Enter name of 1 process: P1
Enter process time: 24
Enter name of 2 process: P2
Enter process time: 3
Enter name of 3 process: P3
Enter process time: 3
```

Output

```
.....................................................
        P1      P2      P3
.....................................................
        24      3       3
.....................................................
0       24      27      30
.....................................................

Average time: 17.000000
Turn around time: 27.000000
```

Program 8 (SJFS)

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>

int main()
{
    char p[10][5],temp[5];
    int c=0,pt[10],i,j,n,temp1;
    float bst=0.0,turn=0.0;

    printf("Enter no of process: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter name of %d process: ",i+1);
        scanf("%s",&p[i]);
        printf("Enter process time: ");
        scanf("%d",&pt[i]);
    }

    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(pt[i]>pt[j])
            {
                temp1=pt[i];
                pt[i]=pt[j];
                pt[j]=temp1;

                strcpy(temp,p[i]);
                strcpy(p[i],p[j]);
                strcpy(p[j],temp);
            }
        }
    }

    printf("\n.....................................................\n");

    for(i=0;i<n;i++)
    {
        printf("\t%s",p[i]);
    }

    printf("\n.....................................................\n");

    for(i=0;i<n;i++)
    {
        printf("\t%d",pt[i]);
    }

    printf("\n.....................................................\n");
    printf("0");

    for(i=0;i<n;i++)
    {
        bst+=c;
        turn+=c+pt[i];
        c+=pt[i];
        printf("\t%d",c);
    }

    printf("\n.....................................................\n");
    printf("\nAverage time: %f",bst/n);
    printf("\nTurn around time: %f\n", turn/n);

    return 0;
}
```

Input

```
Enter no of process:3
Enter name of 1 process: P1
Enter process time: 24
Enter name of 2 process: P2
Enter process time: 2
Enter name of 3 process: P3
Enter process time: 3
```

Output

```
.....................................................
        P2      P3      P1
.....................................................
        2       3       24
.....................................................
0       2       5       29
.....................................................

Average time: 2.333333
Turn around time: 12.000000
```

File Format

- Problem Statement
- Theory or Concept
- Algorithm/Flowchart
- Code
- Output