

DATE 3/9/94  
EXP. NO. 2

EXPT TITLE: Write a C Program to implement  
The System call using fork()

PAGE NO.

3

### Algorithm :-

Step 1:- Start the Program

Step 2:- Declare the Variable Pid and Childid

Step 3:- Get the Child id Value using System Call

Step 4:- If the Child id value is greater than  
Zero then Print as "I am in the Parent Process"

Step 5:- If Child id != 0 Then using get Pid()  
System Call get the Process id

Step 6: ~~Print I am in the Parent Process and~~  
~~Print the Process id~~

Step 7: If Child id != 0 Then using get Pid()  
System

Step 8:- Print I am in The Parent Process &  
Print The Parent Process id

Step 9:- Else if Child Process id Value is less than  
Zero the Print as "I am in the Child Process"

Step 10:- If Child id != 0 then using get Pid () System  
call get the Process id.

DATE .....	EXPT TITLE:	PAGE NO.
EXP. NO. ....		4

Step 11:- Print "I am in The Child Process" & Print the Process id

Step 12:- If Child id != 0 Then using getpid() System Call get the Parent Process Id.

Step 13:- Print "I am in the Child Process" and Print the Parent Process id

Step 14:- Stop the Program.

Program :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    int id, child_id;
    pid = getpid();
    if ((child_id = fork()) > 0)
    {
        printf("In I am in the Parent Process %d", id);
        printf("In I am in the Parent Process %d", getpid());
        printf("In I am in the Parent Process %d", getppid());
    }
    else
    {
        printf("In I am in the Child Process %d", id);
        printf("In I am in the Child Process %d", getpid());
        printf("In I am in the Child Process %d", getppid());
    }
}
```

Anetra

O/P:

o am	in the	Parent Process	4970
o am	in the	Parent Process	4970
o am	in the	Parent Process	4201
o am	in the	Child Process	4970
o am	in the	Child Process	4971

Algorithm:

Step 1:- Start the Program.

Step 2:- Decrease The Variable Pid and i of Integer

Step 3:- Get The Child Pid Value using the System Call fork()

Step 4:- If Child Id Value is less than Zero then  
Print fork Failed

Step 5:- Else if Child Id Value is Equal to Zero  
It is the Id Value of the Child and then  
Start The Child Process to Create & Perform  
Steps 7 & 8

Step 6:- Else Perform Step 9

Step 7:- Use a Forloop For almost Five Child  
Process to be Called.

Step 8:- After Execution of the Forloop then  
Print "Child Process ends"

Step 9:- Execute the System Call wait () to Make  
The Parent to wait for the Child Process  
to get over

DATE .....  
EXP. NO. .....

EXPT TITLE:

PAGE NO.

7

Step 10: Once the Child Processes are terminated  
The Parent terminates & hence prints  
"Parent Process Ends".

Step 11: After both the Parent & Child Processes  
get terminated or execute the exit(1)  
System call to permanently get deleted  
from the OS

Step 12: Stop the Program.

Program:

```
#include <stdio.h>
#include <unistd.h>
int Main()
{
    int i, Pid;
    Pid = fork();
    if (Pid == -1)
        {
            printf("Fork failed");
            exit(0);
        }
    else if (Pid == 0)
        {
            printf("In Child Process Starts : ");
            for (i=0 ; i<5 ; i++)
                {
                    printf("In Child Process %d ; is called , : ");
                }
            printf("In Child Process ends ");
        }
    else
        {
            wait(0);
            printf("In Parent Process ends ");
        }
    exit(0);
}
```

Anitha

8/10/94

3b

EXPT TITLE: write a C program to implement  
Exit() & wait() System call using switch  
case

PAGE NO.

9

## Program

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

int main()
{
    pid_t pid;
    int rv;
    switch (pid = fork())
    {
        case -1:
            perror("fork");
            exit(1);
        case 0:
            printf("In CHILD: This is the child Process\n");
            fflush(stdout);
            printf("CHILD: My PID is %d\n", getpid());
            printf("CHILD: Enter my exit Status (make it small): \n");
            scanf("%d", &rv);
            printf("CHILD: I'm outta here!\n");
            exit(rv);
        default:
            printf("In PARENT: This is the Parent Process\n");
    }
}
```

printf("PARENT: My PID is '%d\n'", getpid());  
fflush(stdout);

printf("PARENT: I'm now waiting for my  
child to exit (...) ... \n");  
wait(&rv);

printf("PARENT: My child's PID is '%d\n', pid);  
fflush(stdout);

printf("PARENT: My child's exit status is '%d\n', wait status(rv));

printf("PARENT: I'm outta here! \n");  
exit(0);

3

getchar();

Anitha

3

## Program

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>
#include <string.h>
```

```
void read_file (const char *file path)
```

{

```
FILE *file = fopen (file path, "r");
```

```
if (file == NULL)
```

{

```
errno = "Failed to open file ";
```

```
return;
```

}

```
char line [256];
```

```
while (fgets (line, sizeof (line), file) != NULL)
```

{

```
printf ("..%s", line);
```

}

~~fclose(file);~~

}

```
int main (int argc, char *args[])
```

{

```
if (argc != 2)
```

{

```
fprintf (stderr, "Usage : %s <directory-name>\n",
        argv[0]);
```

```
return EXIT_FAILURE;
```

{

```
DIR * dirP = opendir(argv[1]);
```

```
if (dirP == NULL) {
```

```
    perror("Failed to open directory");
```

```
    return EXIT_FAILURE;
```

{

```
struct dirent * direntP;
```

```
while (direntP = readdir(dirP)) != NULL {
```

```
    if (strcmp(direntP->d_name, "..") != 0 && strcmp(direntP->d_name, ".") != 0)
```

{

```
    printf("File: %s\n", direntP->d_name);
```

```
    if (direntP->d_type == DT_REG) {
```

```
        char *filepath[1024];
```

```
        snprintf(filepath, sizeof(filepath), "%s/%s", argv[1],  
                 direntP->d_name);
```

```
        printf("Contents of %s:\n", filepath);
```

```
        read_file(filepath);
```

{

```
    printf("\n");
```

{

```
    if (closedir(dirP) == -1)
```

{

```
        perror("Failed to close directory");
```

```
        return EXIT_FAILURE;
```

{

```
    return EXIT_SUCCESS;
```

{

DATE 9.2.110194  
EXP. NO. 5

EXPT TITLE: write a program to stimulate  
the working of scheduling

PAGE NO.

14

### ALGORITHM: Shortest Running Time Implementation

Purpose : To represent the working of the Shortest remaining time algorithm.

Input : No of process ; Burst time of each process  
↳ Quantum time

Output : Average waiting & turnaround time

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

```
int main()
```

{

```
int pid[15];
```

```
int bt[15];
```

```
int n;
```

```
printf("Enter the number of processes ");
```

```
scanf("%d", &n);
```

```
printf("Enter Process id of all the processes ");
```

```
for(int i=0; i<n; i++)
```

{

```
scanf("%d", &pid[i]);
```

}

```
printf("Enter burst time of all the processes ");
```

```
for(int i=0; i<n; i++)
```

{

```
scanf("%d", &bt[i]);
```

}

```
int i, wt[n];
```

```
wt[0] = 0;
```

```
for(i=1; i<n; i++)
```

```
{
```

```
wt[i] = bt[i-1] + wt[i-1];
```

}

printf("Process ID Burst Time waiting Time  
Turnaround Time(h)");

float twt = 0.0;

float tat = 0.0;

for (i=0; i<n; i++)

{

printf("%d %f %f", Pid[i], bt[i], twt[i]);

printf("%d %f %f", bt[i], twt[i]);

printf("%d %f %f", bt[i], twt[i]);

// Calculating & printing turnaround time of  
each process

printf("%d %f %f", bt[i], twt[i]);

twt += twt[i];

tat += (wtt[i] + bt[i]);

}

float att, awt;

awt = twt/n;

att = tat/n;

printf("Avg. waiting time = %f(n", awt);

printf("Avg. turnaround time = %f", att);

Anitha