DATE 3/9/24 EXP. NO. 2

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

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 getPid() 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 getPid() System

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

Step 9 :- Else if ~~Child~~ Process id value is lessthan Zero the Print as " i am in the Child Process"

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

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

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

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

Step 14:- Stop the Program.

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

EXPT TITLE:

PAGE NO. 5

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
  int id, childid;
  id = getPid();
  if((childid = fork())>0)
{
Printf("\n i am in the Parent Process %d", id);
Printf("\n i am in the Parent Process %d", getPid());
Printf("\n i am in the Parent Process %d", getPPid());
}
else
{
Printf("\n i am in the Child Process %d", id);
Printf("\n i am in the Child Process %d", getPid());
Printf("\n i am in the Child Process %d", getPPid());
}
}
```

Aneetha

O/P:

```
i. am in the Parent Process 4970
0 am in the Parent Process 4970
0 am in the Parent Process 4201
1 am in the Child Process 4970
? am in the Child Process 4971
```

1.10.24 | EXPT TITLE: Write a C Program to implement Fork() & wait() System Call | PAGE NO. 6

NO. 3a

Algorithm:

Step 1:- Start the Program.

Step 2:- Decleare the Variable Pid and i as integer

Step 3:- Get The Child Id 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 Execute & 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

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

Step 11:- After both the Parent & Child Processes get terminated it Execute the Exit() System call to Permanently get deleted from the OS

Step 12:- Stop the Program.

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

EXPT TITLE:

PAGE NO. 8

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 ("\n Child process starts :");
   for (i=0 ; i<5; i++)
  {
 printf (" \n Child Process %d is Called ", i);
  }
  printf ("\n Child Process ends");
  }
 else
  {
   wait (0);
   printf (" \n Parent Process ends");
   }
   exit (0);
}
```

Anitha.

8/10/24

3b

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

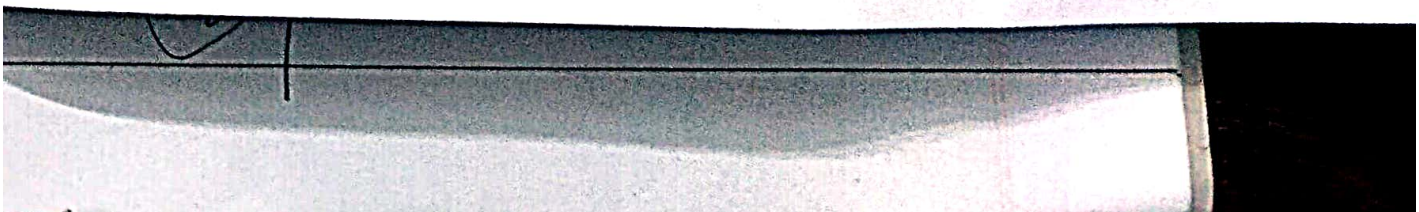
PAGE NO. 9

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("\n 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:
Parintf("\n 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", WEXITSTATUS(rv));
Printf("PARENT: I'm outta here!\n");
exit(0);
}
return 0;
}
```

Anitha



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)
  {
    perror (" 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 (closed (dirp) == -1)
{
  perror ("Failed to close directory");
return EXIT_FAILURE;
}
return EXIT_SUCCESS;
}
```

Anitha

DATE 22/10/24 | EXPT TITLE: write a program to stimulate the working of Scheduling | PAGE NO. 14

EXP. NO. 5

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/type.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 the all the
                Process");
  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\n");
float twt = 0.0;
float tat = 0.0;
for(i=0 ; i<n; i++)
{
  Printf("%d\t\t", Pid[i]);
  Printf("%d\t\t", bit[i]);
  Printf("%d\t\t", wt[i]);
// Calculating & Printing turnaround time of
      each Process
Printf("%d\t\t", bt[i]+wt[i]);
Printf("\n");
twt += wt[i];
tat += (wt[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