**ASSIGNMENT 3**

**AIM:** TO LEARN THE PROCESS FUNCTIONS

**Theory:**

# fork() in C

Fork system call is used for creating a new process, which is called *child process*, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

*Negative Value*: 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 process ID of newly created child process.

exec()

The exec() family of functions replaces the current process image with a new process image. It loads the program into the current process space and runs it from the entry point.

The exec() family consists of following functions, I have implemented execv() in following C program, you can try rest as an exercise

# Wait System Call in C

Prerequisite : [Fork System call](https://www.geeksforgeeks.org/fork-system-call/)

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.

Child process may terminate due to any of these:

* It calls exit();
* It returns (an int) from main
* It receives a signal (from the OS or another process) whose default action is to terminate.

Zombie Process:

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

In the following code, the child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn’t call [wait()](https://en.wikipedia.org/wiki/Wait_%28system_call%29) and the child process’s entry still exists in the process table.

Orphan Process:

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now.

However, the orphan process is soon adopted by init process, once its parent process dies.

Code:

parent.c

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<string.h>

int main(){

    int st;

    char in[100];

    int nums[100];

    int i ,j,t;

    char tmp[10];

    //Accepting inputs

    for( i = 0; i <3; i++){

        scanf("%s",tmp);

        nums[i] = atoi(tmp);

        strcat(in,tmp);

        strcat(in,",");

    }

    char \*args[]={"./child.o",in,NULL};

    st = fork();

    if(st == 0){

        //Child Process

        execv(args[0],args);

    }

    else{

        //Parent Process

        printf("From Parent Descending: ");

        for(i = 0; i < 3; i++){

            for(int j = i+1; j < 3; j++){

                if(nums[i]<nums[j]){

                    t = nums[i];

                    nums[i] = nums[j];

                    nums[j] = t;                                }

            }

            printf("%d,",nums[i]);

        }

        printf("\n");

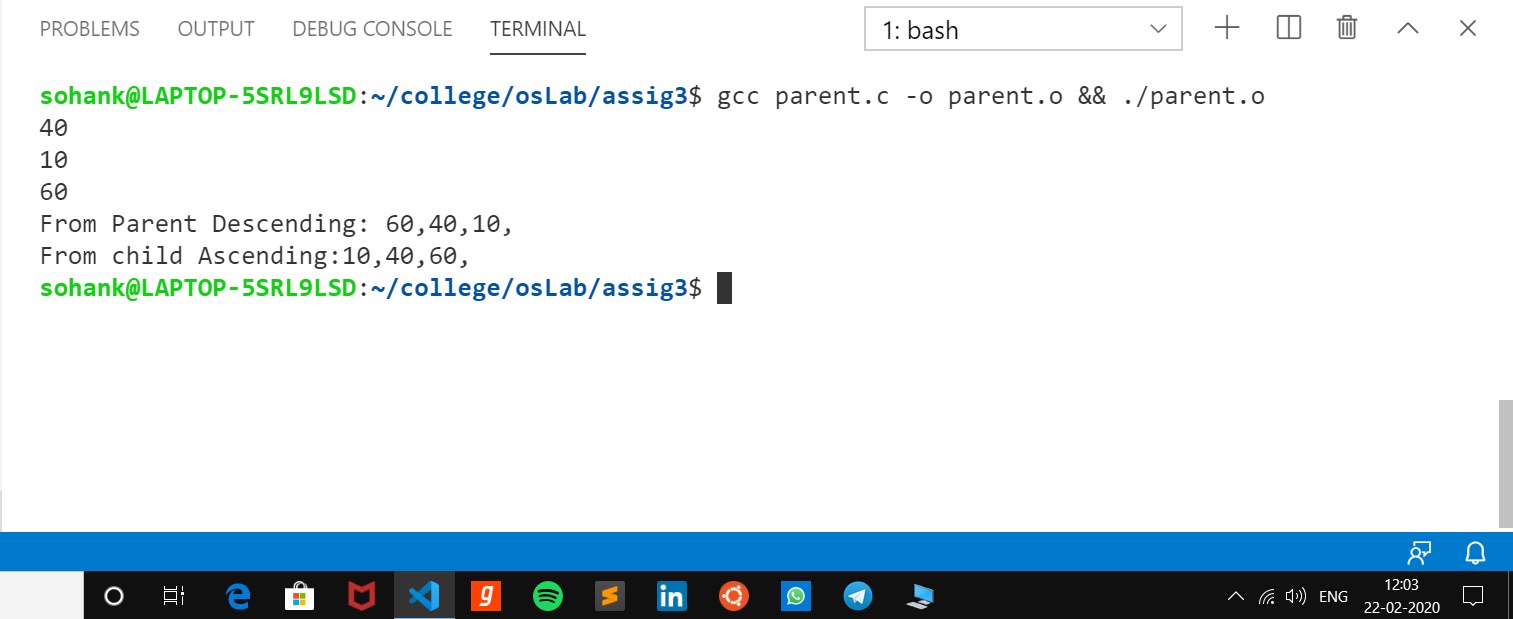
        //Waiting to finish child

        wait(NULL);

    }

    return 1;

}



child.c

#include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<stdlib.h>

void main(int args, char\* argv[]){

    char arr[10];

    strcpy(arr,argv[1]);

    int i,j=0;

    int integ[3];

    int tmp;

    char t[2];

    char s[3];

    //Converting string to the Integer Array

    for(i = 0; i<strlen(arr); i++){

        t[0] = arr[i];

        t[1] = '\0';

        if(strcmp(t,",")==0){

            integ[j] = atoi(s);

            strcpy(s,"\0");

            j++;

        }

        else{

            t[0] = arr[i];

            t[1] = '\0';

            strcat(s,t);

        }

    }

    integ[j] = atoi(s);

    //Sorting the array

    for(i = 0 ; i < 3; i++){

        for(j = i ; j < 3; j++){

            if(integ[i] > integ[j]){

                tmp = integ[i];

                integ[i] = integ[j];

                integ[j] = tmp;

            }

        }

    }

    printf("From child Ascending:");

    for(i=0;i<3;i++)

        printf("%d,",integ[i]);

    printf("\n");

}

Output:

