

Submitted by:

Name: Aditi Sudhir Igade

Batch: A3

Roll no.:221081

PRN No.:22220311

Assignment No: - 3

AIM: To implement following programs to exhibit UNIX Process Control "Program where parent process sorts array elements in ascending order and child process sorts array elements in descending order.To show the demonstration of wait() and zombie process"

THEORY:

1. What is Process

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

- Whenever a command is issued in Unix/Linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.
- Through a 5 digit ID number Unix/Linux keeps an account of the processes, this number is called process ID or PID. Each process in the system has a unique PID.
- Used up pid's can be used in again for a newer process since all the possible combinations are used.
- At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

2. How to create a new process

A new process can be created by the fork() system call. The new process consists of a copy of the address space of the original process. fork() creates new process from existing process. Existing process is called the parent process and the process is created newly is called child process. The function is called from parent process. Both the parent and the child processes continue execution at the instruction after the fork(), the return code for the fork() is zero for the new process, whereas the process identifier of the child is returned to the parent.

Fork() system call is situated in <sys/types.h> library.

3. Fork()

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.

4. Exec()

The exec system call is used to execute a file which is residing in an active process. When exec is called the previous executable file is replaced and new file is executed.

More precisely, the exec system call will replace the old file or program from the process with a new file or program. The entire content of the process is replaced with a new program.

The user data segment which executes the exec() system call is replaced with the data file whose name is provided in the argument while calling exec().

The new program is loaded into the same process space. The current process is just turned into a new process and hence the process id PID is not changed, this is because we are not creating a new process we are just replacing a process with another process in exec.

If the currently running process contains more than one thread then all the threads will be terminated and the new process image will be loaded and then executed. There are no destructor functions that terminate threads of current process.

5. Orphan processes

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.

6. Zombi processes

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() and the child process's entry still exists in the process table.

PROGRAM:

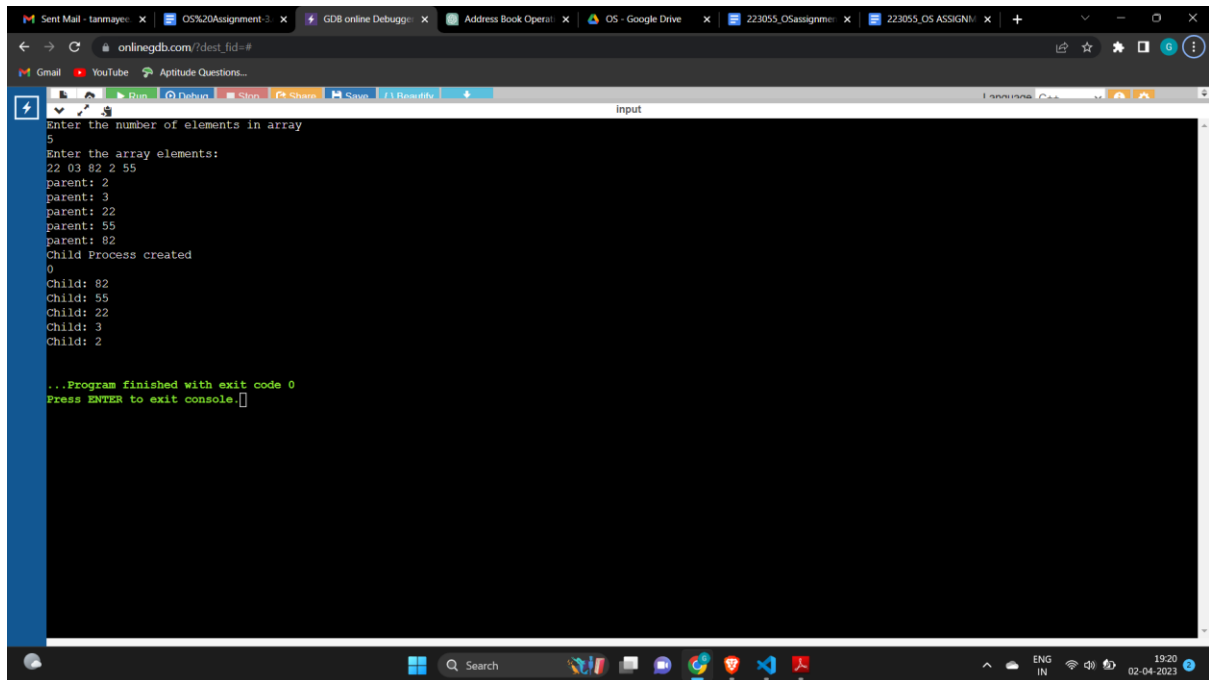
```
#include<iostream>
#include<unistd.h>
using namespace std;
int main(){
//int arr[7] = {22,45,12,33};
//int n = 4;
int n;
cout<<"Enter the number of elements in array"<<endl;
cin>>n;
int arr[n];
cout<<"Enter the array elements: "<<endl;
for(int i =0;i<n;i++){
    cin>>arr[i];
}
int j,k,l;
int pid;
pid= fork();
if (pid<0){
cout<<"Error occured"<<endl;
}
else if(pid==0){
cout<<"Child Process created"<<endl;
cout<<pid<<endl;
for(int i = 0;i<n;i++){
    for(j = n-1;j>i;j--){
        if (arr[i]<arr[j]){
            k=arr[i];
            arr[i]=arr[j];
            arr[j]=k;
        }
    }
}
```

```

        arr[j]=k;
    }
}
}
for(int i=0;i<n;i++){
    cout<<"Child: ";
    cout<<arr[i]<<endl;
}
}
else{
for(int i = 0;i<n;i++){
    for(j = n-1;j>i;j--){
        if (arr[i]>arr[j]){
            k = arr[i];
            arr[i] = arr[j];
            arr[j] = k;
        }
    }
}
for(int i=0;i<n;i++){
    cout<<"parent: ";
    cout<<arr[i]<<endl;;
}
}
}

```

OUTPUT:



The screenshot shows a web browser window with the URL `onlinegdb.com/?dest_fid=#`. The browser has several tabs open, including 'Sent Mail - tanmayee', 'OS%20Assignment-3', 'GDB online Debugge', 'Address Book Operat...', 'OS - Google Drive', '223055_OSassignment', and '223055_OS ASSIGN'. The browser's address bar shows the URL. Below the browser window, there is a terminal window titled 'input'. The terminal displays the following output:

```
Enter the number of elements in array
5
Enter the array elements:
22 03 82 2 55
parent: 2
parent: 3
parent: 22
parent: 55
parent: 82
Child Process created
0
Child: 82
Child: 55
Child: 22
Child: 3
Child: 2
...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:

Hence we studied processes and the `fork()` system call.