**ASSIGNMENT 3**

**AIM: I**mplement 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. Show the demonstration of wait() and zombie process”

**THEORY:**

Process*: A process is basically a program in execution. The execution of a process must progress in a sequential fashion.*

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. 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 running process with a new process. It can be used to run a C program by using another C program. It comes under the header file **unistd.h.**

wait():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.

Zombi 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.

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.

**CODE:**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<stdlib.h>

**int** main()

{

**int** pid,i,n;

printf("Enter the size of the array \n");

scanf("%d",&pid);

**char** \*args[pid+2];

printf("Enter the numbers array elements.\n");

**for**(i=1;i<pid+1;i++)

{

args[i] = (**char**\*) malloc(4\***sizeof**(**char**)); *//assigning memory to the string pointer.*

scanf("%4s",args[i]);

}

args[i]=**NULL**; *//Terminating the args array with NULL.*

**if**(fork()==0) *//Executing the child process.*

{

args[0]="./child";

system("gcc child.c -o child");

printf("Child object created\n");

execv(args[0],args); *//Executing the child object file.*

}

**else***//Executing the parent process.*

{

pid=wait(**NULL**); *//Waiting for the child process to execute*

printf("\n\nWaiting for child process with pid : %u \n",pid);

args[0]="./parent";

system("gcc parent.c -o parent");

printf("Parent object created\n");

execv(args[0],args); *//Executing the parent object file.*

}

**return**0;

}

code(for parent process):

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

**int** main(**int** args,**char** \* argv[])

{

printf("This is the parent process\n");

**int** arr[args-1],i,j,c,min;

**for**(i=1;i<args;i++)

arr[i-1]=atoi(argv[i]); *//Converting the string variable to integer.*

printf("\n\nAscending order :\n");

args--;

**for**(i=0;i<args-1;i++) *//Ascending order.*

{

c=i;

**for**(j=i;j<args;j++)

{

**if**(arr[j]<arr[c])

c=j;

}

min=arr[i];

arr[i]=arr[c];

arr[c]=min;

}

**for**(i=0;i<args;i++)

printf("%d\n",arr[i]);

**return**0;

}

Code for child process:

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<stdlib.h>

**int** main(**int** args,**char** \* argv[])

{

printf("This is the child process\n");

**int** arr[args-1],i,j,c,max;

**for**(i=1;i<args;i++)

arr[i-1]=atoi(argv[i]); *//Converting the string variable to integer.*

printf("\n\nDescending order\n");

args--;

**for**(i=0;i<args-1;i++) *//Descending order.*

{

c=i;

**for**(j=i;j<args;j++)

{

**if**(arr[j]>arr[c])

c=j;

}

max=arr[i];

arr[i]=arr[c];

arr[c]=max;

}

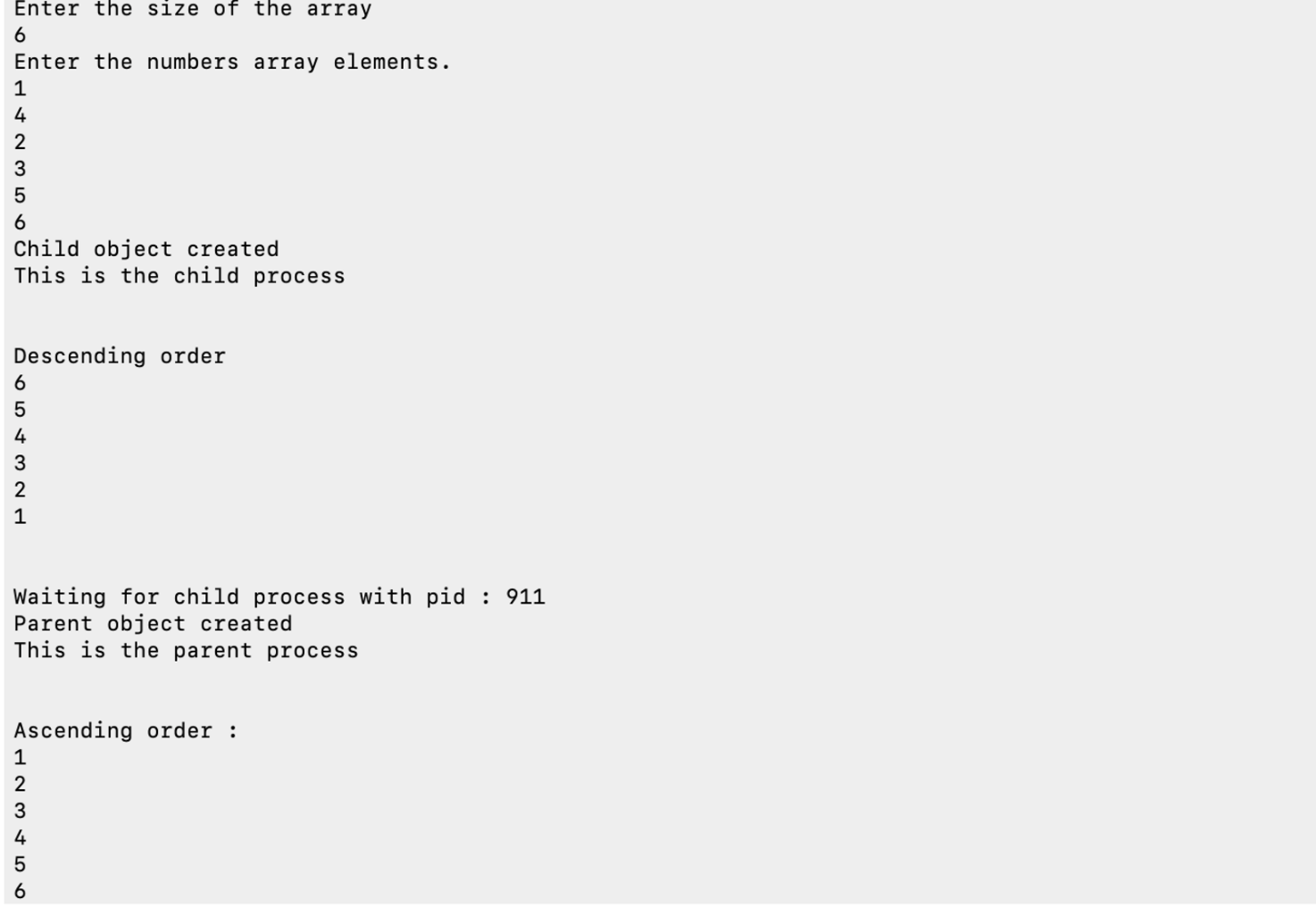
**for**(i=0;i<args;i++)

printf("%d\n",arr[i]);

**return**0;

}

**Output:**

****

**Conclusion:**

In this assignment we learn about basic of processes: how they are created, killed, their basic attributes and their types.