```
Name     :- Amit.Y.Zope
Roll No. :- 223067
Gr No.   :- 21810714
Division :- C
Batch    :- C3
Subject  :- Operating System
```

# Assignment No 3

**Aim:** 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. Show the demonstration of wait() and zombie process".

**Objective:** To Study System Calls.

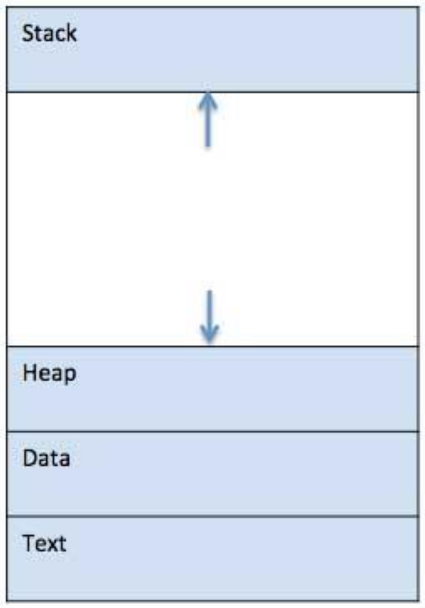
**Theory:**

## What is Process ?

A process is a program in execution. For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data.



| | |
|---|---|
| 1 | **Stack**<br>The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
| 2 | **Heap**<br>This is dynamically allocated memory to a process during its run time. |
| 3 | **Text**<br>This includes the current activity represented by the value of Program Counter and the contents of the processor's registers. |
| 4 | **Data**<br>This section contains the global and static variables. |

## Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

| S.N. | Information & Description |
|---|---|

| | |
|---|---|
| 1 | **Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever. |
| 2 | **Process privileges** This is required to allow/disallow access to system resources. |
| 3 | **Process ID** Unique identification for each of the process in the operating system. |
| 4 | **Pointer** A pointer to parent process. |
| 5 | **Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process. |
| 6 | **CPU registers** Various CPU registers where process need to be stored for execution for running state. |
| 7 | **CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process. |
| 8 | **Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system. |
| 9 | **Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc. |

|  |  |
|---|---|
| 10 | **IO status information**<br>This includes a list of I/O devices allocated to the process. |

## fork() System Call

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().

***1. Negative Value***: creation of a child process was unsuccessful.
***2 .Zero***: Returned to the newly created child process.
***3. Positive value***: Returned to parent or caller. The value contains process ID of newly created child process.

## wait() 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.

# exec() system call

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.**

**execv** : This is very similar to execvp() function in terms of syntax as well. The syntax of **execv()** is as shown below:

**Syntax:**

```
int execv(const char *path, char *const argv[]);
```

**path:** should point to the path of the file being executed.
**argv[]:** is a null terminated array of character pointers.

## 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.

## 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:

### Parent Proces Code:

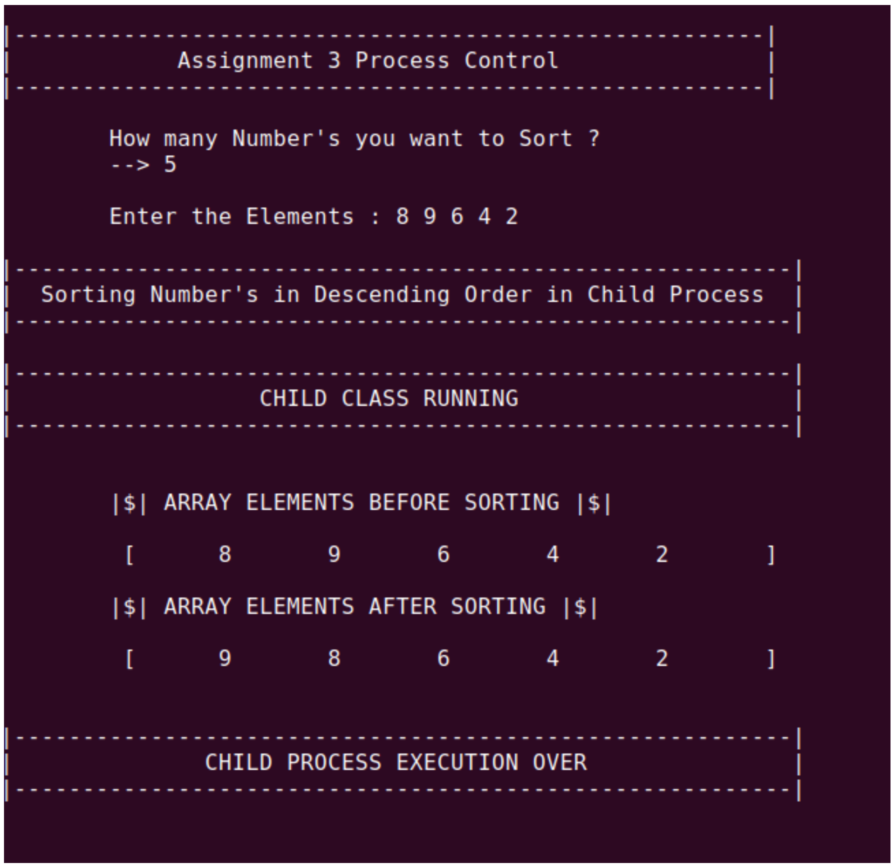
```
            /**********************************************/
/**     Assignment 3(A)
**    Sort Number in ascending in Parent Process
**    Sort Number in descending in Child Process
**
**********************************************/
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<string.h>
#include<unistd.h>

int main(){
      int array_size; //array size to store the count of number's
      char exe[100] ={"./childp "}; //exe character array to contain the
executable filename


      printf("\n|----------------------------------------------------------|\n");
      printf("|            Assignment 3 Process Control               |\n");
      printf("|----------------------------------------------------------|\n");

      printf("\n\tHow many Number's you want to Sort ?\n");//Asking user how many
number's user want's to sort
      printf("\t--> ");
      scanf("%d",&array_size); //taking array size input from user

      int array_num[array_size]; //declaring an array to store the number's

      printf("\n\tEnter the Elements : ");

      for(int i=0;i<array_size;i++){

            scanf(" %d",&array_num[i]); //taking number's input to array
            char buffer_array[15];          //buffer array which stores the number
as character type temporary array
            sprintf(buffer_array,"%d ",array_num[i]);//sprintf used to convert the
present input as string in buffer array
            strcat(exe,buffer_array);//this string concatenate is use to
concatenate all the number and executable file together
      }

      printf("\n|----------------------------------------------------------|\n");
      printf("|  Sorting Number's in Descending Order in Child Process  |\n");
      printf("|----------------------------------------------------------|\n");


      int pid = fork();//fork() system call is used to create child process

      if(pid == 0 ){

            int j = 0;
```

```
          char *token = strtok(exe," ");//tokenizing exe array
          char *args[20] = {NULL}; //final array use to pass to child proces in 
exec system call
          while(token!=NULL){
               args[j++] = token;
               token = strtok(NULL," ");
          }

          execv(args[0],args);//execv() system call use to execute another 
process than parent

     }
        else{
          wait(NULL);//untill the child process is not over this parent process 
will wait


          printf("\n|----------------------------------------------------------
|\n");
          printf("|                    PARENT CLASS RUNNING                    
|\n");
          printf("|----------------------------------------------------------
|\n\n");

          printf("\n|----------------------------------------------------------
|\n");
          printf("|  Sorting Number's in Ascending Order in Parent Process     
|\n");
          printf("|----------------------------------------------------------
|\n\n");

          printf("\t|$| ARRAY ELEMENTS BEFORE SORTING |$|\n");
          printf("\n\t [\t");
          for(int i=0;i<array_size;i++){
               printf("%d",array_num[i]);
               printf("\t");
          }
          printf("]\n\n");
          int temp2;
          //Bubble sorting in ascending order
          for(int i = 0;i<array_size;i++){
               for(int j=0;j<array_size-1-i;j++){
                    if(array_num[j]>array_num[j+1]){
                         temp2 = array_num[j];
                         array_num[j] = array_num[j+1];
                         array_num[j+1] = temp2;
                    }
               }
          }
          //displaying sorted array number's
          printf("\t|$| ARRAY ELEMENTS AFTER SORTING |$|\n");

          printf("\n\t [\t");
          for(int i=0;i<array_size;i++){
               printf("%d",array_num[i]);
               printf("\t");
          }
          printf("]\n\n");
          printf("\n\n");

          printf("\n|-------------------------------------------------------------
|\n");
          printf("|               PARENT CLASS EXECUTION OVER
|\n");
          printf("|-----------------------------------------------------------
|\n\n");

     }
          printf("\t!!!$$$ PROGRAM EXITED SUCCESSFULLY $$$!!!\n\n");

return 0;
}
```

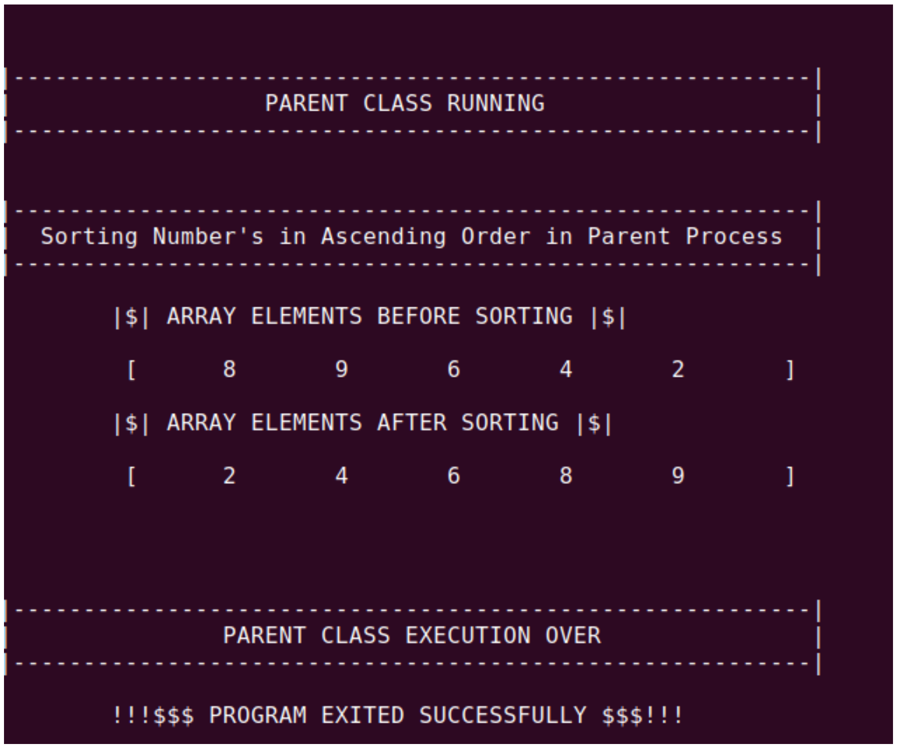
## Child Process Code :

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

int main(int argc,char *argv[]){


          printf("\n|-------------------------------------------------------------
|\n");
          printf("|                  CHILD CLASS RUNNING
|\n");
          printf("|-----------------------------------------------------------
|\n");

          int array_sort[argc];//this array is use to store the number's with
size of argument count

          for(int i = 0;i<argc-1;i++){
               array_sort[i] = atoi(argv[i+1]);//convert the argument from
character to integer

          }

          printf("\n\n");
          printf("\t|$| ARRAY ELEMENTS BEFORE SORTING |$|\n");
          printf("\n\t [\t");
          for(int i=0;i<argc-1;i++){
               printf("%d",array_sort[i]);
               printf("\t");
          }
          printf("]\n\n");
          int temp2;
          //Bubble sorting descending order
          for(int i =0;i<argc-1;i++){
               for(int j=0;j<argc-2;j++){
                    if(array_sort[j]<array_sort[j+1]){
                         temp2 = array_sort[j];
                         array_sort[j] = array_sort[j+1];
                         array_sort[j+1] = temp2;
                    }
               }
          }


          //Displaying Sorted Elements
          printf("\t|$| ARRAY ELEMENTS AFTER SORTING |$|\n");

          printf("\n\t [\t");
          for(int i=0;i<argc-1;i++){
               printf("%d",array_sort[i]);
               printf("\t");
          }
          printf("]\n\n");

          printf("\n|---------------------------------------------------------
|\n");
          printf("|              CHILD PROCESS EXECUTION OVER
|\n");
          printf("|---------------------------------------------------------
|\n");




          printf("\n\n");

return 0;
}
```

Output :



```
|-------------------------------------------------------|
|            Assignment 3 Process Control               |
|-------------------------------------------------------|

        How many Number's you want to Sort ?
        --> 5

        Enter the Elements : 8 9 6 4 2

|---------------------------------------------------------|
|  Sorting Number's in Descending Order in Child Process  |
|---------------------------------------------------------|

|---------------------------------------------------------|
|                   CHILD CLASS RUNNING                   |
|---------------------------------------------------------|


        |$| ARRAY ELEMENTS BEFORE SORTING |$|

         [      8       9       6       4       2       ]

        |$| ARRAY ELEMENTS AFTER SORTING |$|

         [      9       8       6       4       2       ]


|---------------------------------------------------------|
|               CHILD PROCESS EXECUTION OVER              |
|---------------------------------------------------------|
```



```
|----------------------------------------------------------|
|                   PARENT CLASS RUNNING                   |
|----------------------------------------------------------|


|----------------------------------------------------------|
|  Sorting Number's in Ascending Order in Parent Process   |
|----------------------------------------------------------|

        |$| ARRAY ELEMENTS BEFORE SORTING |$|

         [      8       9       6       4       2       ]

        |$| ARRAY ELEMENTS AFTER SORTING |$|

         [      2       4       6       8       9       ]



|----------------------------------------------------------|
|               PARENT CLASS EXECUTION OVER                |
|----------------------------------------------------------|

        !!!$$$ PROGRAM EXITED SUCCESSFULLY $$$!!!
```