

Assignment 3

221071

1) 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"

2) THEORY: (must contain what is process, PCB, fork(), exec(), wait(), zombi, orphan processes)

-)What is a **Process**?

An instance of a program is called a 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 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.

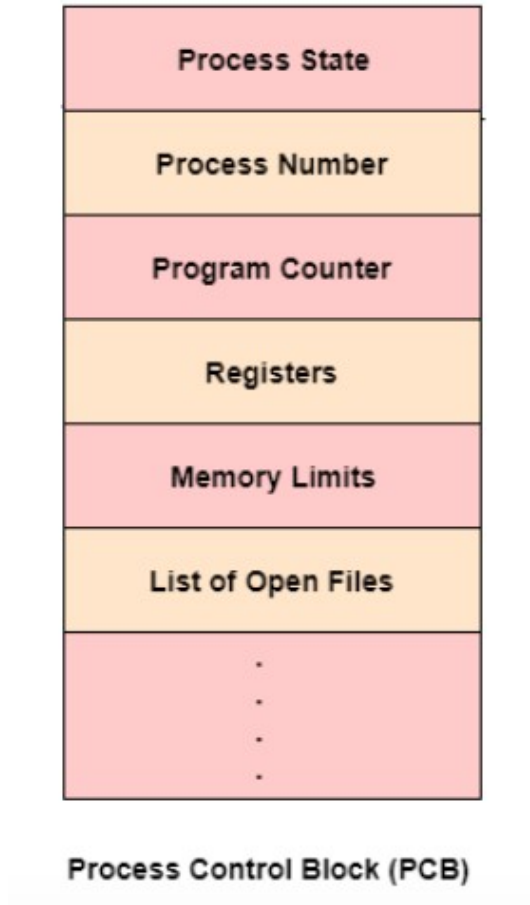
-) **Process Control Block**

Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.

Structure of the Process Control Block

The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram:

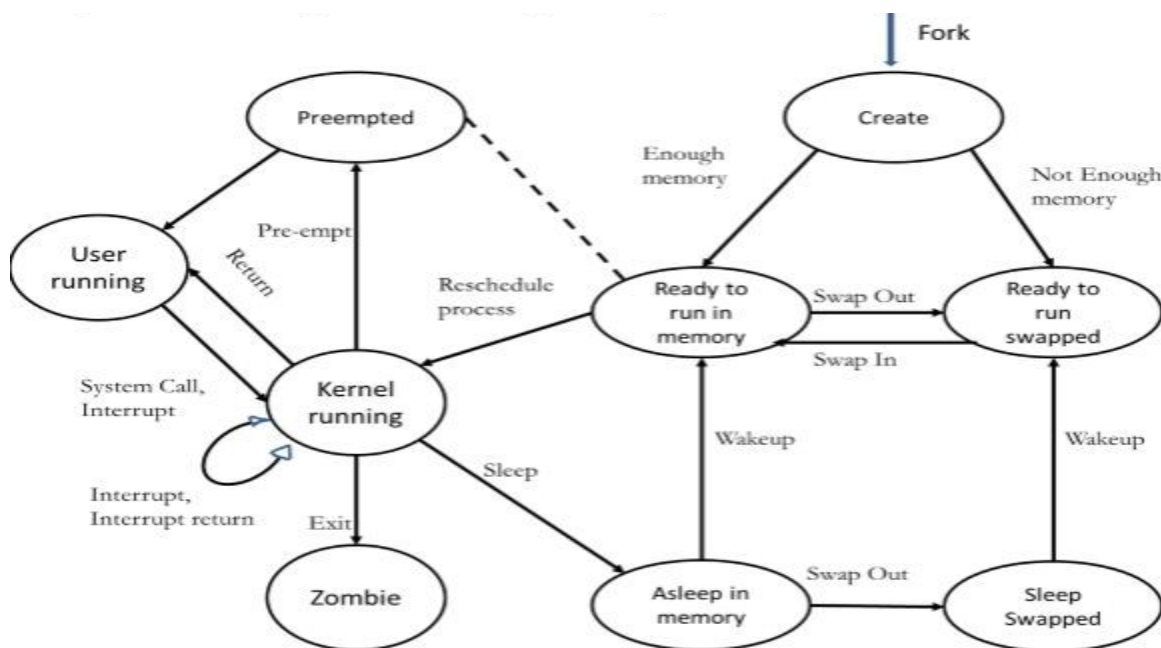


The following are the data items:

- Process State
- This specifies the process state i.e. new, ready, running, waiting or terminated.
- Process Number
- This shows the number of the particular process.
- Program Counter
- This contains the address of the next instruction that needs to be executed in the process.
- Registers
- This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.
- List of Open Files
- These are the different files that are associated with the process

- CPU Scheduling Information
 - The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.
- Memory Management Information
 - The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.
- I/O Status Information
 - This information includes the list of I/O devices used by the process, the list of files etc.
- Accounting information
 - The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.
- Location of the Process Control Block
 - The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

The process **state** diagram for UNIX system is given below:

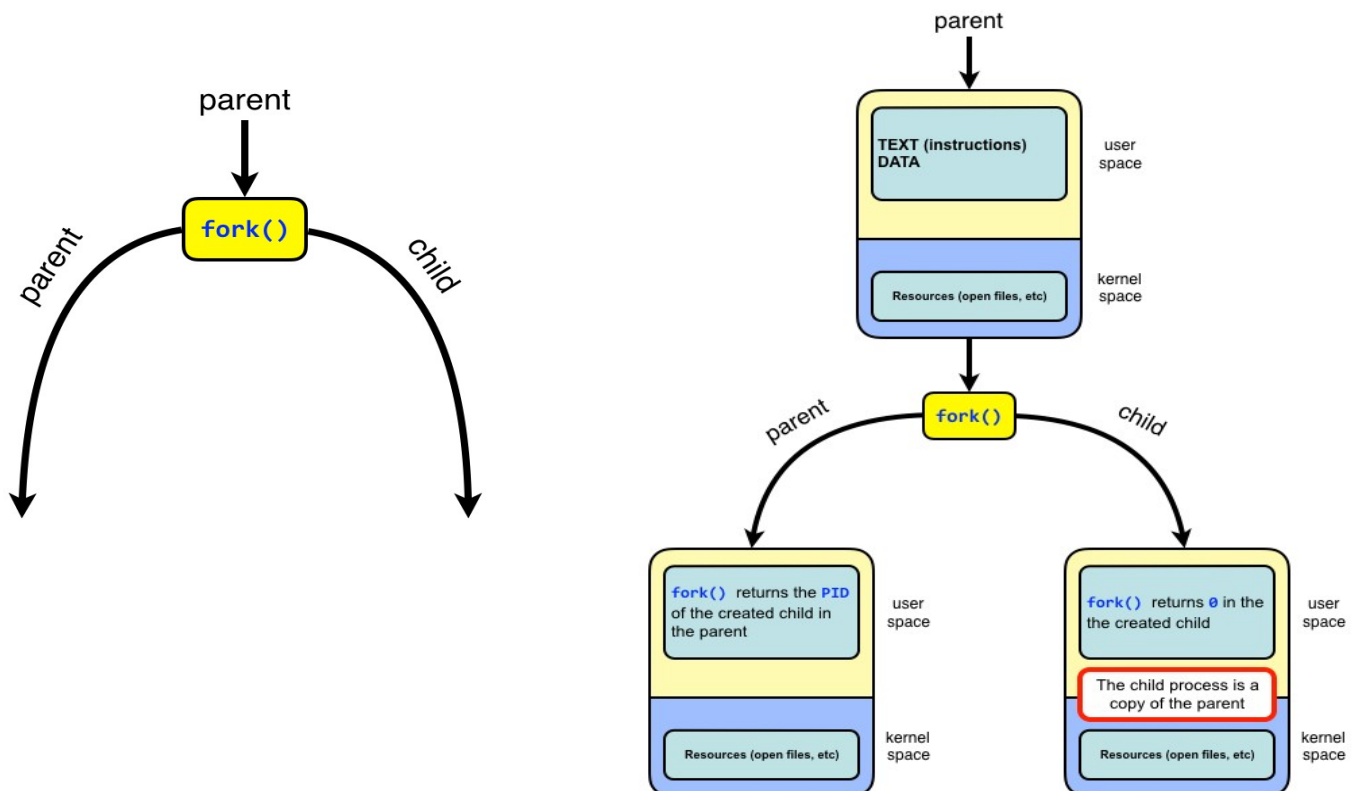


- **Created:** Process is newly created and is not ready to run.

- **User Running:** The process is currently executing in user mode.
- **Kernel Running:** The process is currently running in kernel mode.
- **Ready to run, in memory:** The process is ready to run and is waiting for the Kernel to schedule it.
- **Ready to run, swapped:** The process is ready to run, but is currently not in the main memory and is waiting for the swapper to swap it to the main memory.
- **Sleep, Swapped:** A blocked state wherein the process is awaiting an event and has been swapped to secondary storage.
- **Asleep in memory:** A blocked state wherein the process is waiting for an event to occur and is currently in the main memory.
- **Pre-empted:** Process is running from kernel to user mode. But the kernel pre-empts it and does a process switch and schedules another process.
- **Zombie:** Process no longer exists. It still leaves behind a record of it for its parent process to access.

-) Fork()

The `fork` system call is the primary (and historically, only) method of process creation in Unix-like operating systems.

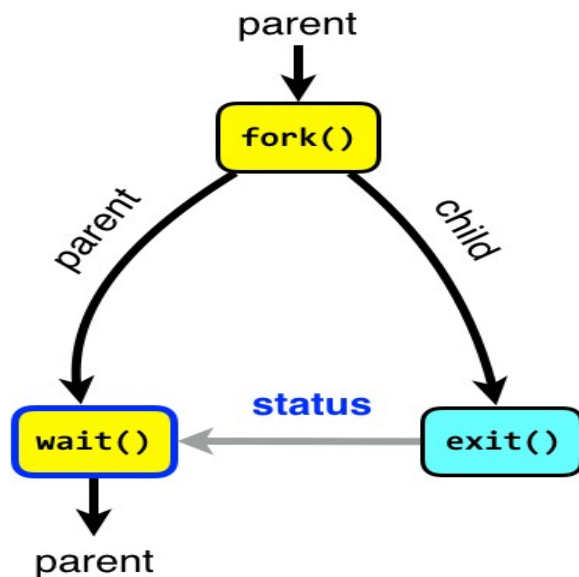


-)Orphans

An orphan process is a process whose parent process has terminated, though it remains running itself. Any orphaned process will be immediately adopted by the special init system process with PID 1.

-)Wait

The wait system call blocks the caller until one of its child process terminates. If the caller doesn't have any child processes, `wait` returns immediately without blocking the caller. Using `wait` the parent can obtain the exit status of the terminated child.



-) Zombies

A terminated process is said to be a zombie or defunct until the parent does `wait` on the child.

- When a process terminates all of the memory and resources associated with it are deallocated so they can be used by other processes.
- However, the exit status is maintained in the PCB until the parent picks up the exit status using `wait` and deletes the PCB.
- A child process always first becomes a zombie.
- In most cases, under normal system operation zombies are immediately waited on by their parent.
- Processes that stay zombies for a long time are generally an error and cause a resource leak.

-) wait()

`wait()` system call suspends execution of current process until a child has exited or until a signal has delivered whose action is to terminate the current process or call signal handler.

-) **exec()**

exec() family of functions or sys calls replaces current process image with new process image.

There are functions like **execl**, **execlp**, **execle**, **execv**, **execvp** and **execvpe** are used to execute a file.

-)**exit()**

This function is used for normal process termination. The status of the process is captured for future reference. There are other similar functions **exit(3)** and **_exit()**., which are used based on the exiting process that one is interested to use or capture.

3) code:

parent.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
    int st,arr[10],i,j,cnt=0,n,tmp,key;
    char buffer [50];

    printf("\nEnter how many numbers you want ? ");
    scanf("%d",&n);

    printf("Enter array element= ");
    for(i=1;i<=n;i++)
        scanf("%d",&arr[i]);

    printf("\nArray is\n");
    for(i=1;i<=n;i++)
    {
        sprintf(buffer, "%d", arr[i]);
        printf("\t %s",buffer);
    }
    printf ("\nprinting string...");
    printf("\n%s",buffer);

    // fork() for creating child process
    st=fork();
    char *args[] = {"/.p",buffer,NULL};
    if(st == 0)
    {
```

```

        wait(NULL);          // wait state
        execv(args[0],args); // execv() for running diffrent thing child process
        while(1);
    }
    else
    {
        printf("\nIn Parent Process..... \n");
        printf("\n\nSorting elements in ascending order ");
        // soring logic
        for (i=1;i<=n;i++)
        {
            key=arr[i];
            j=i-1;
            while (j>=0 && arr[j]>key)
            {
                arr[j+1]=arr[j];
                j=j-1;
            }
            arr[j+1]=key;
        }
        printf("\n");
        for(i=1;i<=n;i++)
            printf("\t%d",arr[i]);
        printf("\n");
    }
    return 0;
}

```

child.c

```

#include <stdio.h>
#include <string.h>
int main(int args, char * argv[])
{
    printf("\nIn child Process.... \n");
    printf("%s",argv[1]);

    int n=10,arr[10],tmp,i,key,j;
    arr[10]=atoi(argv[1]);

    // sort logic strrev(argv[1]);
    for (i=1;i<n;i++)
    {
        key=arr[i];
        j=i-1;
        while(j>=0&&arr[j]<key)
        {
            arr[j+1]=arr[j];
            j=j-1;
        }
    }
}

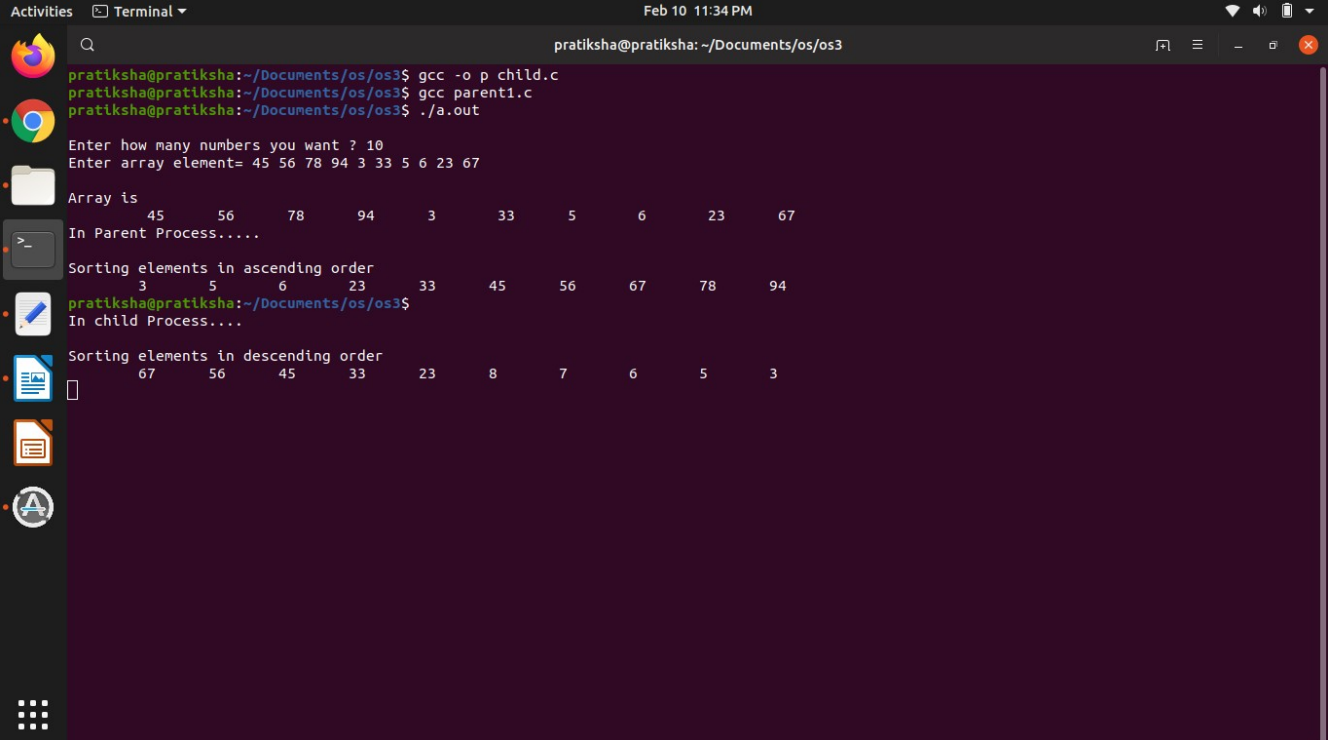
```

```

    }
    arr[j+1]=key;
    }
    printf("\n");
    for(i=1;i<=n;i++)
        printf("\t%d",arr[i]);
    printf("\n");
}

```

4) output.



```

pratiksha@pratiksha:~/Documents/os/os3$ gcc -o p child.c
pratiksha@pratiksha:~/Documents/os/os3$ gcc parent1.c
pratiksha@pratiksha:~/Documents/os/os3$ ./a.out
Enter how many numbers you want ? 10
Enter array element= 45 56 78 94 3 33 5 6 23 67
Array is
    45    56    78    94    3    33    5    6    23    67
In Parent Process.....
Sorting elements in ascending order
    3    5    6    23    33    45    56    67    78    94
pratiksha@pratiksha:~/Documents/os/os3$
In child Process....
Sorting elements in descending order
    67    56    45    33    23    8    7    6    5    3

```