Pritam Mogal TE(B)-17.

Assignment No. 10
UNIX system Calls
Assignment No - C3
Aim:- Implement UNIX System calls like for process management

Problem Statement:- To write a program to implement UNIX System calls like for process Management.

Pre-requisites:- 1. Explain concept of system call.
2. Explain state diagram working of new process.

Software requirements.

| S.No | Facilities required | Quantity |
|---|---|---|
| 1 | System | 1 |
| 2 | O/S | Ubuntu Kylin |
| 3 | S/W name | CTurbo or GCC |

Hardware Requirements :- No.

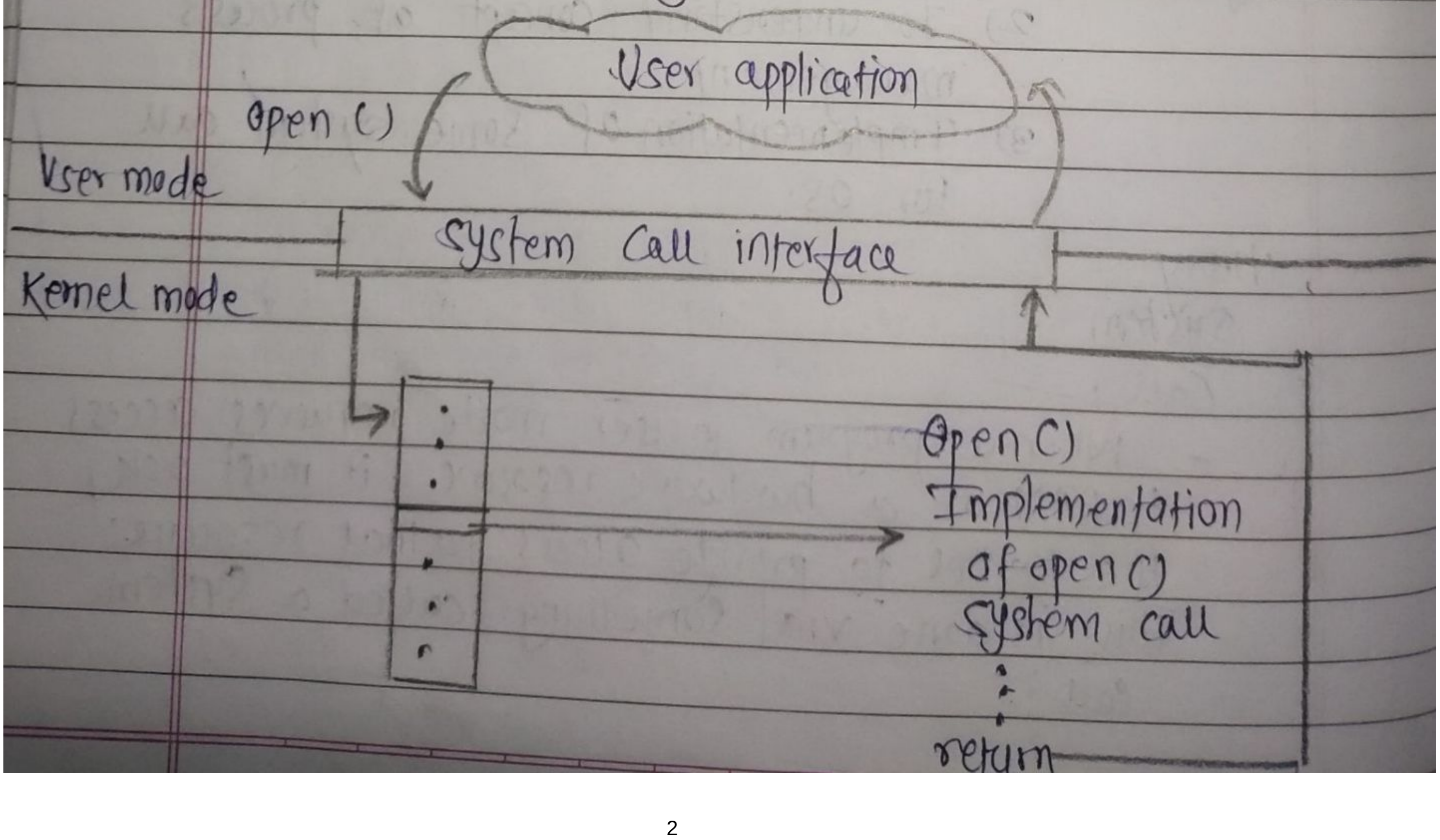
Objectives:- 1) To understand UNIX system call
2) To understand concept of process management
3) Implementation of some system call for OS.

Theory:-
System Call:
- When a program in user mode requires access to RAM or a hardware resource, it must ask the Kernel to provide access to that resource. This is done via something called a system call.

- Then the Kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from Kernel mode back to user mode.

Generally, System Call are made by the user level programs in the following situations:
- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.

Kernel Mode:-
- When CPU is a in Kernel mode, the code being executed can access any meomory address and any hardware resource.
- That means the system will be in a safe state enven if a program in user mode crashes
- Hence most programs in an OS run in user mode.



SYSTEM CALLS BASICS:-

- System called are functions, we need to include the proper header files
  - Eg for getpid () we need
    - # include <sys/ types.h>
    - # include <unistd.h>

Syscalls For processes:

- Pid_t Fork (void)
  - ✓ Create a new child process, which is a copy of the current process
  - ✓ Parent return value is the PID of the child process.

- int execl (char *name, char *arg (0), ..., (char *)0)
  - ✓ change progoam image of current process.
  - ✓ Reset stack and free meamory.

- int kill (Pid_t Pid, int sig)
  - ✓ Send a Signal to a process
  - ✓ Send SIGKILL to force termination.

Unix System Calls:-

- Ps command:

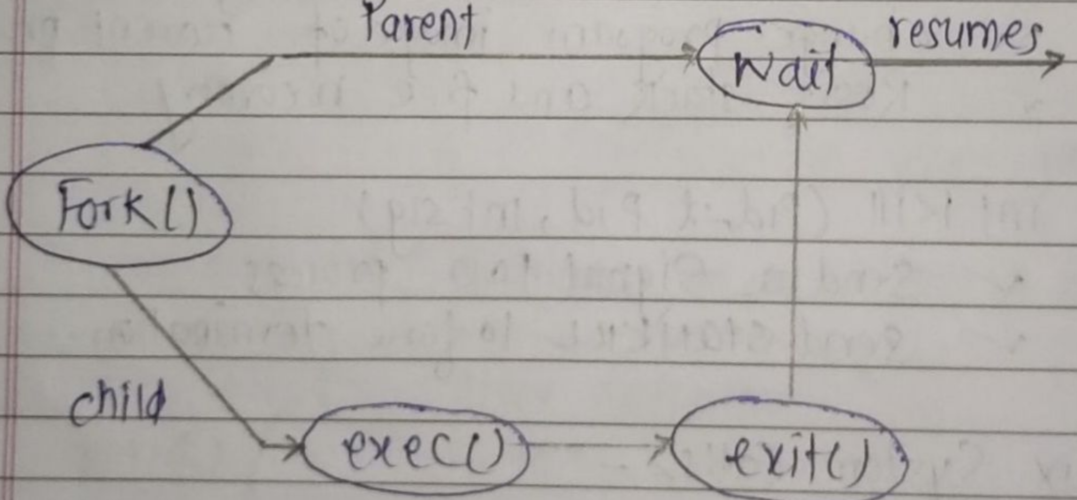
The PS (i.e process status) command is used to provide information about the currently running processes, including their process identification numbers (PID's).

- Fork command:

The fork () system call is used to create processes. when a process (a program in execution).

• Exec() Command:→
The exec() system call is also used to create processes. But there is one big difference Fork () and exec () calls. The fork() call creates a new process while preserving the parent process.

child process may terminate due to any of These
✓ if Calls exit();
✓ if returns (an int) from main:
✓ if receives a signal (from the os or another process) whose default action is to terminate.



Conclusion:
Thus, The process system call program is implemented and studied various system calls.

**Name** : Pritam Gotiram Mogal **DivRoll**: TE-B-17

# Assignment No. 10 [UNIX System Calls]

**Problem Satement**: To write a program to implement UNIX system calls like for process Management.

**1. Code**:

**Problem Statement** : Write a C program to create a child process using fork system call. Display Status of running processes used in child process(EXEC) & terminate child process before completion of parent task(wait).

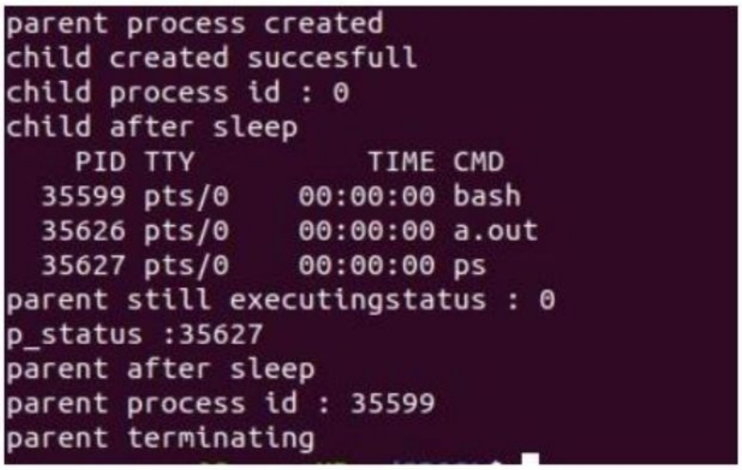
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>


int main()
{ pid_t pid , ppid , p_status ;
        int status ;
        printf("parent process created \n");
        pid = fork();
        if(pid ==0)
        {
                printf("child created succesfull\n");
                printf("child process id : %d \n",
                pid); sleep(10); printf("child after
                sleep \n");
                execlp("/bin/ps","ps",NULL);

                printf("child terminating\n");
                exit(0);
        }
        else
        { printf("parent still executing"); p_status
                = wait(&status); printf("status :
                %d \n",status); printf("p_status
                :%d \n",p_status); sleep(10);
                printf("parent after sleep\n");
                ppid = getppid();
                printf("parent process id : %d\n",ppid);
                printf("parent terminating\n");
                exit(0);
        }

        return 0;
}
```

**OUTPUT**:



```
parent process created
child created succesfull
child process id : 0
child after sleep
    PID TTY          TIME CMD
  35599 pts/0    00:00:00 bash
  35626 pts/0    00:00:00 a.out
  35627 pts/0    00:00:00 ps
parent still executingstatus : 0
p_status :35627
parent after sleep
parent process id : 35599
parent terminating
```