Practical no: 10

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

Name: Aditi Dinesh Mulay

Class: T.E. Computer

Subject: SPOS

Div: A

Roll no: 02

PRN No. 71918146B

SPOS

Assignment C-3

Aditi Dinesh Mulay
T.E. Comp Div:A
Roll no: 02.

Aim: Implement UNIX system calls like for process management.

Pre requisites - 1) Explain concept of system call.
2. Explain state dig. working of new process.

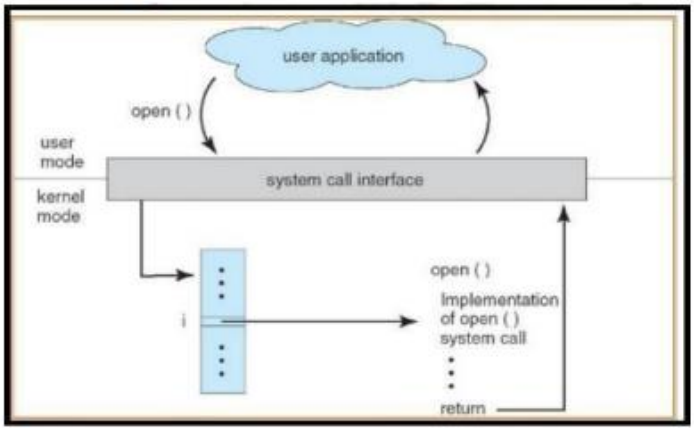
Software requirements - o/s - Ubuntu
s/w name - CTurbo or Gcc.

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

Theory :

System call:

1) When a program in user mode require access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via system call.
2) When program makes a system call, the mode is switched from user mode to kernel mode. This is context switch.
3) 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.



Kernel Mode:

1) When CPU is in kernel Mode, the code being executed can access any memory address & any hardware resource
2) In user mode, if any program crashes, only that particular program is halted.
3) That means system will be in a safe state even if a program in user mode crashes.
4) Hence, most programs in an OS run in user mode.

System call basics:

1) Since system calls are functions, we need to include the proper header files
2) Most system calls have a meaningful return value.

Syscalls for processes:

1) pid_t fork (void)
2) int execl (char *name, char * arg (), ..., (char*)0)
3) pid_t wait (int *status)
4) void exit (int status)
5) int kill (pid_t pid, int sig)

UNIX System Calls:

1) ps command : (Process status)
   - It is used to provide information about the currently running processes, including their process identification number (PIDs).
   A process also reffered to as a task, is an

executing instance of a program. Every process is assigned a unique PID by system : Syntax : ps [opt^]

2) Fork command :

-It is used to create processes. When a process makes a fork() call, an exact copy of process is created. There are two processes, 1) parent 2) child.

3) Join Command :

-It is a command line utility for joining lines of two files on a common field. Used to join files by selecting fields within line & joining files on them. Result written to std. o/p. Syntax : Join [opt]... file1 file2.

4) Exec() command

- Used to create processes. But there is one big diff. bet^n fork() & exec() calls. Fork() calls creates a new process while preserving the parent process. But an exec() call replaces the add space, text segment, data segment of current process with new process.

* Child process may terminate due to any of these.
1) It calls exit(); it returns (an int) from main.
It receives a signal (from OS) whose default act^n is to terminate.

* Conclusion:,

Thus, the process system call program is implemented and studied various system calls.