10th Assignment

Ankita Bonde
Roll No 19

Aim: Implement UNIX system call like for process management

Problem Statement WAP to implement UNIX sys call like for process management

S/w Requirement : Os ubuntu
S/w CTurbo or GCC

Theory

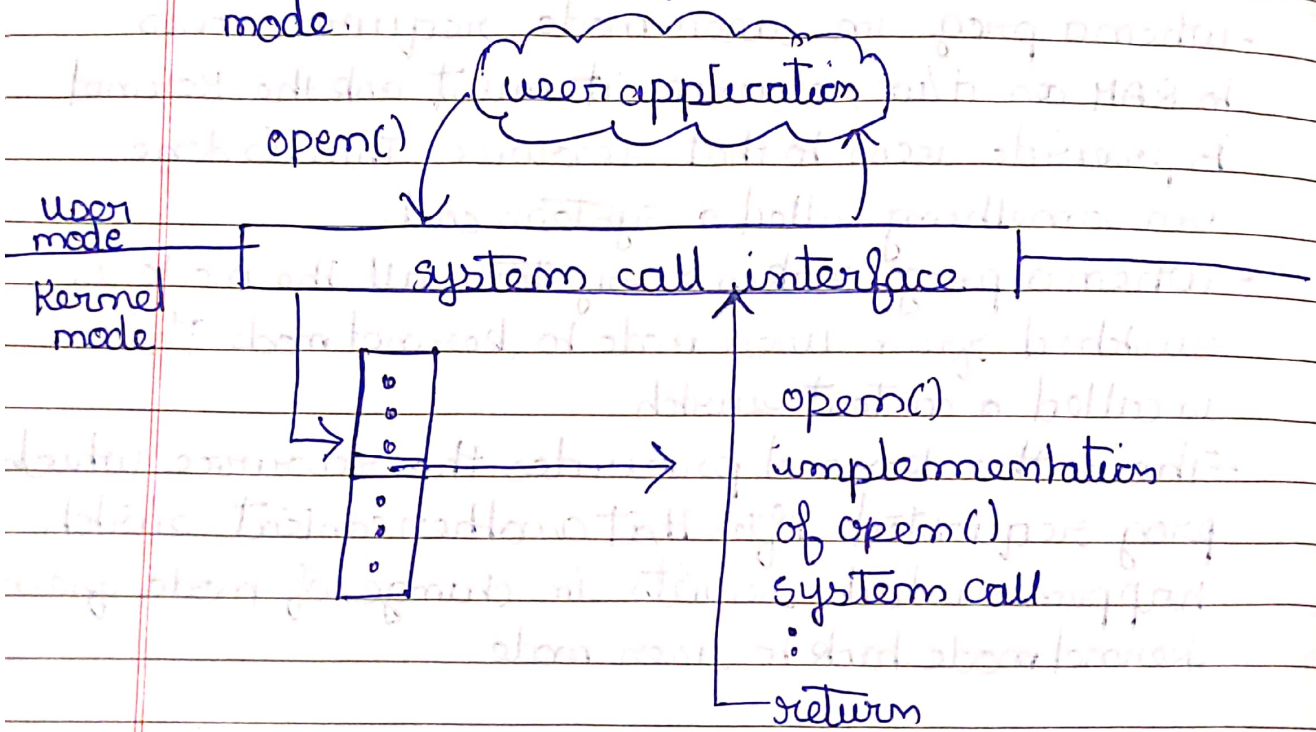
① CALL
- when a prog in user mode requires acess to RAM or H/w resource it must ask the Kernel to provide acess to that resource. This is done via something called a system call.
- when a prog makes a system call the mode is switched from user mode to kernel mode this is called a context switch
- Then the kernel provides the resource which prog requested After that another context switch happens which results in change of mode from kernel mode back to user mode

- In following situations system call are generated
① Creating, opening, closing, deleting files
② Creating & managing process
③ Creating a connection in n/w, sending & recieving packets

④ Request access to h/w device like mouse & printer.

② KERNEL Mode

- When CPU mode is in Kernel mode the code being executed can access any memory address and any H/w resource.
- In user mode if any prog crashes only that particular prog haulted.
- That is system will be in safe state even if a prog in user mode crashes.
- Hence most prog in an OS run in user mode.



③ Basics of System call

- Since system call are functions we need to include proper header files
eg for getpid() we need

#include <sys/type.h>
#include <unistd.h>

- Most system call have a meaningful return value
① usually -1 or a negative value indicates an error
② A specific error code is place in a global var called: errno
③ To access errno you must declare it: extern int errno;

④ Sys Calls for Processes:

4.1 pid_t fork (void)
- create a new child process, which is a copy of current process
- Parent return value is the PID of child process
- child return value is 0.

4.2 int execl (char *name, char *arg(), ... (char *) 0)
- change prog image of current process
- Reset stack and free memory
- start at main()
- Also see other version: execlp(), execv(), etc.

4.3. pid_t wait (int * status)
- wait for a child process to complete
- Also see waitpid() to wait for a specific process.

44 void exit (int status)
- Terminate the calling process
- send SIGKILL to force termination

(5) Unix System Call

5.1 PS command

- Used to provide info about currently running process, including their process identification nos.
- Every process assigned a unique PID by the system

Syntax: ps [options].

5.2. Fork command

- The fork () sys call is used to create process.
- When a process makes a fork() call an exact copy of process is created.
- Now there are two process one being the parent process and the other being the child process.

5.3 Join Command

- The join command in UNIX is a command utility for joining lines of two files a comm field
- It can be used to join 2 files by selecting fields within the line and joining the files on them. The result is written to std o/p.

Syntax: Join [option] ... file1 file2

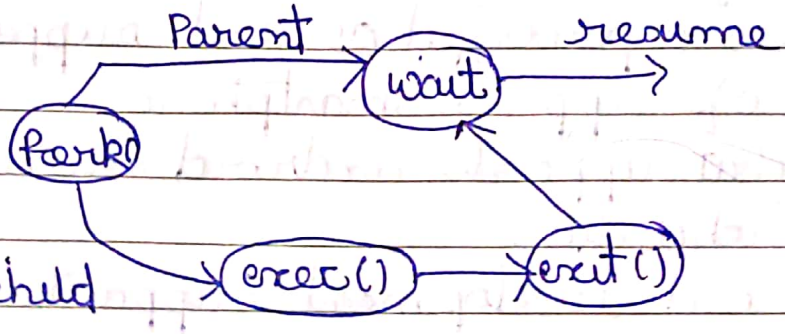
5.4 Exec Commands:

- used to create processes.
- But there is one big difference between

fork() and exec() calls.
- The fork() call creates a new process while preserving the parent process
- But an exec() replaces the address space text seg + Data segment etc. of current process with new process.

5.5 wait command
- 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 sys call instruction

child process may terminate due to any of the following reasons
- It calls exit()
- It returns (an int) from main
- It receives a signal whose default action is to terminate.



Conclusion
Thus the process sys call prog is implemented and studied various system call.

Assignment No :5

Problem Satement: To write a program to implement UNIX system calls like forprocess Management.

Name Ankita Bonde Class: TEA Roll No-19

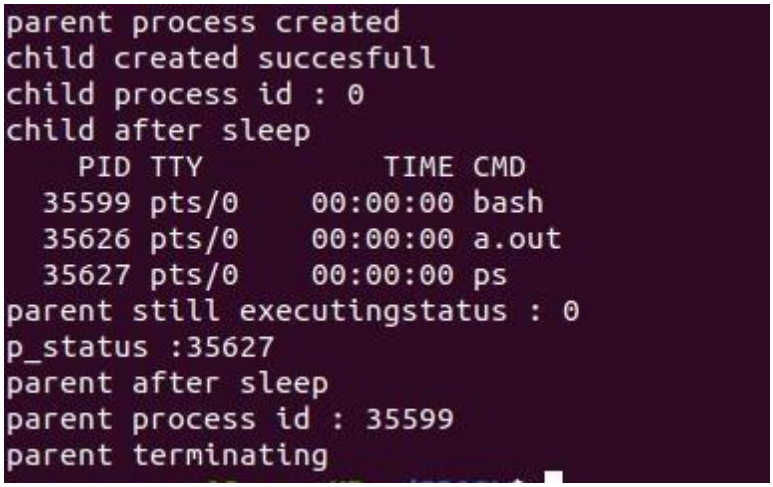
```
#include<stdio.h>
#include<stlib.h>
#include<sys/types.h>

int main()
{
        pid _t pid  , ppid ,  p_status;
        int staus;
        printf("Parent process created \n");
        pid=fork();
        if (pid==0)
        {
                printf("child process created  succesfull\n");
                printf("Parent 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\n");
                p_status=wait(&status);
                printf("status: %d\n"status);
                printf("p_status: %d\n"p_status);
                sleep(10);
                printf("parent after sleep \n");
                ppid=getpid();
                printf("Parent process id:%d \n"ppid);
                printf("parent terminating\n");
                exit(0);
        }
        return 0;

}
```


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