EXPERIMENT 3
System Calls

OBJECTIVE:

- To understand about Linux system calls and their use in processes

BACKGROUND:

What are system calls?

System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf. A system call is invoked in a variety of ways, depending on the functionality provided by the underlying processor. In all forms, it is the method used by a process to request action by the operating system.

Process Creation:

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination

I. fork()

- Has a return value
- Parent process => invokes fork() system call
- Continue execution from the next line after fork()
- Has its own copy of any data
- Return value is > 0 //it's the process id of the child process. This value is different from the Parents own process id.
- Child process => process created by fork() system call
- Duplicate/Copy of the parent process //LINUX
- Separate address space
- Same code segments as parent process
- Execute independently of parent process
- Continue execution from the next line right after fork()
- Has its own copy of any data
- Return value is 0

II. wait ()

- Used by the parent process
- Parent's execution is suspended
- Child remains its execution
- On termination of child, returns an exit status to the OS
- Exit status is then returned to the waiting parent process //retrieved by wait ()
- Parent process resumes execution
- #include <sys/wait.h>
- #include <sys/types.h>

III. exit()

- Process terminates its execution by calling the exit() system call
- It returns exit status, which is retrieved by the parent process using wait() command
- EXIT_SUCCESS // integer value = 0
- EXIT_FAILURE // integer value = 1
- OS reclaims resources allocated by the terminated process (dead process) Typically performs clean-up operations within the process space before returning control back to the OS
- Terminates the current process without any extra program clean-up
- Usually used by the child process to prevent from erroneously release of resources belonging to the parent process

IV. exec()

- Runs an executable file
- Called by an already existing process //child process
- Replaces the previous executable //overlay
- Has an exist status but cannot return anything (if exec() is successful) to the program that made

the call //parent process

- Return value is -1 if not successful

- Overlay => replacement of a block of stored instructions or data with another
- **Information Maintenance:**

i. sleep()

- Process goes into an inactive state for a time period
- Resume execution if
- Time interval has expired
- Signal/Interrupt is received
- Takes a time value as parameter (in seconds on Unix-like OS and in milliseconds on Windows OS)
- sleep(2) // sleep for 2 seconds in Unix
- Sleep(2*1000) // sleep for 2 seconds in Windows

ii. getpid() // returns the PID of the current process

- getppid() // returns the PID of the parent of the current process
- Header files to use
- #include <sys/types.h>
- #include <unistd.h>
- getppid() returns 0 if the current process has no parent

Example:

```
#include <stdio.h>
#include <sys/types.h>
int main()

{

pid_t pid;
pid=fork();
if(pid==0){

printf("I'm in a child process \n\n");

}

else if (pid>0){

printf("I'm in the parent process \n\n");

}

else

{

printf("Error \n\n");

}

return 0;

}
```

In-lab problems:

1. Create a program that creates a child process. The child process prints "I am a child process" 100 times in a loop. Whereas the parent process prints "I am a parent process" 100 times in a loop.
2. Create a program named stat that takes an integer array as command line argument (deliminated by some character such as $). The program then creates 3 child processes each of which does exactly one task from the following.

   a) Adds them and print the result on the screen. (done by child 1)
   b) Shows the average on the screen. (done by child 2)
   c) Prints the maximum number on the screen. (done by child 3)

3. Invoke at least 4 commands from your programs, such as
   Cp, mkdir, rmdir, etc (The calling program must not be destroyed)

Post-Lab Questions:

Q1. Write a program which uses fork () system-call to create a child process. The child process prints the contents of the current directory and the parent process waits for the child process to terminate.

Q2. Write a program which prints its PID and uses fork () system call to create a child process. After fork () system call, both parent and child processes print what kind of process they are and their PID. Also the parent process prints its child's PID and the child process prints its parent's PID.