Group B

Experiment No. 1

Title :

Aim : To study **system calls**

AIM:

THEORY:

The system call provides an interface to the operating system services.

Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

System Call Parameters

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.

3. Parameters can also be pushed on or popped off the stack by the operating system.

Types of System Calls

There are 5 different categories of system calls:

process control, file manipulation, device manipulation, information maintenance and communication.

1. Process Control

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

2. File Management

Some common system calls are create, delete, read, write, reposition, or close. Also, there is a need to determine the file attributes – get and set file attribute. Many times the OS provides an API to make these system calls

3. Device Management

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs request the device, and when finished they release the device. Similar to files, we can read, write, and reposition the device.

4. Information Management

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is time, or date.

The OS also keeps information about all its processes and provides system calls to report this information.

5. Communication

There are two models of interprocess communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data

DESCRIPTION

Process

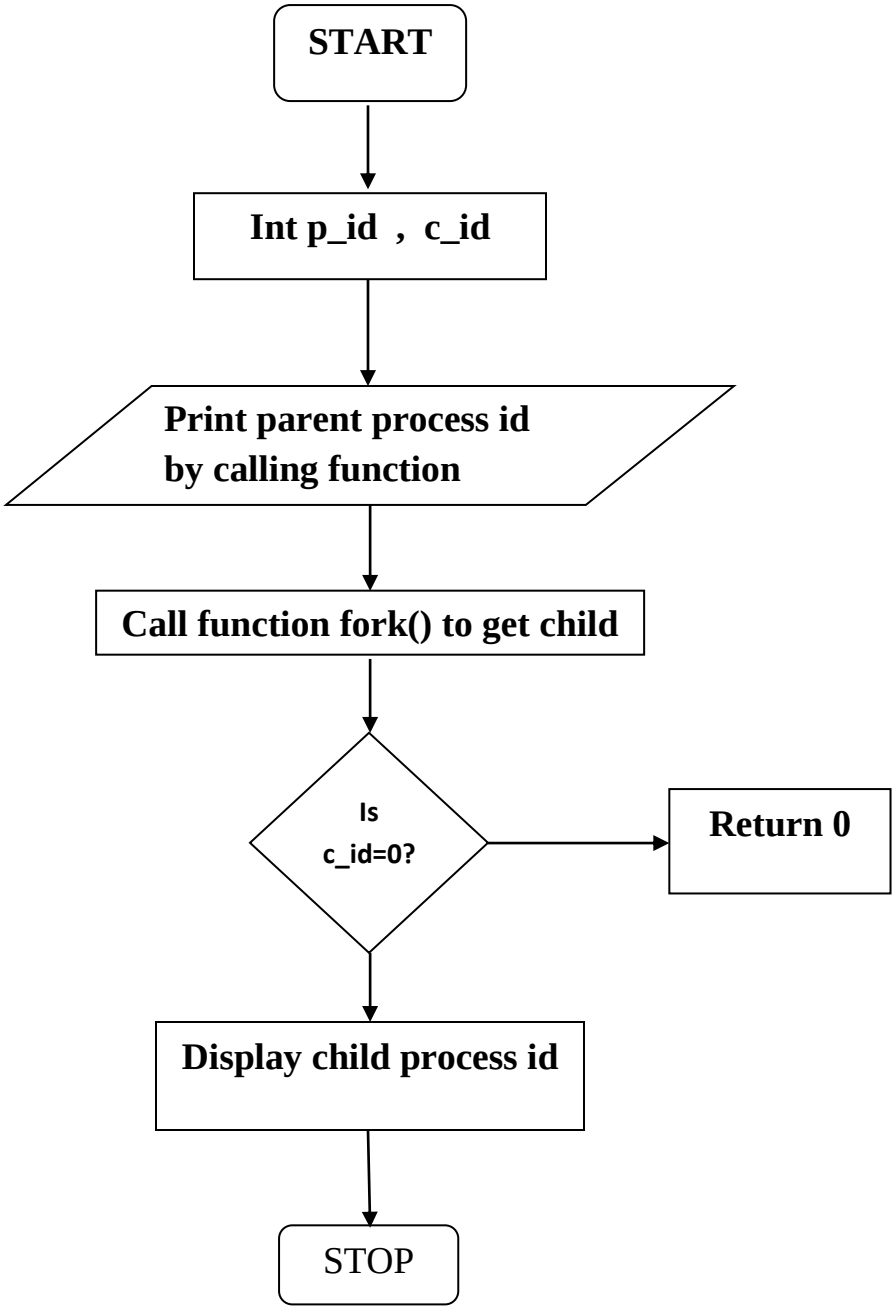
- getpid() returns the process ID of the calling process. (This is often used by routines that generate unique temporary filenames.)

- System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork().

- If fork() returns a negative value, the creation of a child process was unsuccessful.
- fork() returns a zero to the newly created child process.
- fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process.

  Therefore, after the system call to fork(), a simple test can tell which process is the child. Note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

Flow Chart for Process Handling



PROGRAAM CODE

```
/*system program for process handling*/
#include <stdio.h>
#include <sys/types.h>
int main ()
{
int  pid_t,child_pid;
printf ("the main program process ID is %d\n", (int) getpid ());
child_pid = fork ();
if (child_pid != 0)
{
printf ("this is the parent process, with id %d\n", (int) getpid ());
printf ("the child’s process ID is %d\n", (int) child_pid);
}
else
printf ("this is the child process, with id %d\n", (int) getpid ());
return 0;
}
```

----------output----------

```
the main program process ID is 1823
this is the parent process, with id 1823
the child’s process ID is 18
```