Linux System Calls

## System Calls

System calls provide programs running on the computer an interface to talk with the operating system.

When a program needs to ask for a service (for which it does not have permission itself) from the kernel of the operating system it uses a system call. User level processes do not have the same permissions as the processes directly interacting with the operating system. For example, to communicate with and external I/O device or to interact with any other processes, a program has to use system calls.

**System Calls**

**File Related**

Open(), Read(), Write(), Close(), Create File etc

**Device Related**

Read, Write, Reposition, ioctl, fcntl etc

**Information Related**

Getpid, attributes, get system time and date etc

**Process Control**

Load, exec, abort, fork, Wait, signal, allocate etc.

**Communication**

Pipe, create/delete connections, shmget etc.

1. Fork ( )

Used to create new processes. The new process consists of a copy of the address space of the

original process. The value of process id for the child process is zero, whereas the value of

process id for the parent is an integer value greater than zero.

Syntax: fork ( )

2. execve ( )

Used after the fork () system call by one of the two processes to replace the process‟ memory

space with a new program. It loads a binary file into memory destroying the memory image of

the program containing the execlp system call and starts its execution. The child process

overlays its address space with the UNIX command /bin/ls using the execlp system call.

Syntax: execve ( )

3. wait( )

The parent waits for the child process to complete using the wait system call. The wait system

call returns the process identifier of a terminated child, so that the parent can tell which of its

possibly many children has terminated.

Syntax: wait (NULL)

4. exit( )

A process terminates when it finishes executing its final statement and asks the operating

system to delete it by using the exit system call. At that point, the process may return data

(output) to its parent process (via the wait system call).

Syntax: exit (0)

5. getpid( )

Each process is identified by its id value. This function is used to get the id value of a

particular process.

6. getppid ( )

Used to get particular process parent’s id value.

7. perror( )

Indicate the process error