**ICS 2103 - INTRODUCTION TO SYSTEMS PROGRAMMING.**

**CAT 2.**

1. A process is a program that is run on Unix.
2. The UNIX command for viewing the current processes is:

$ ps //Process status

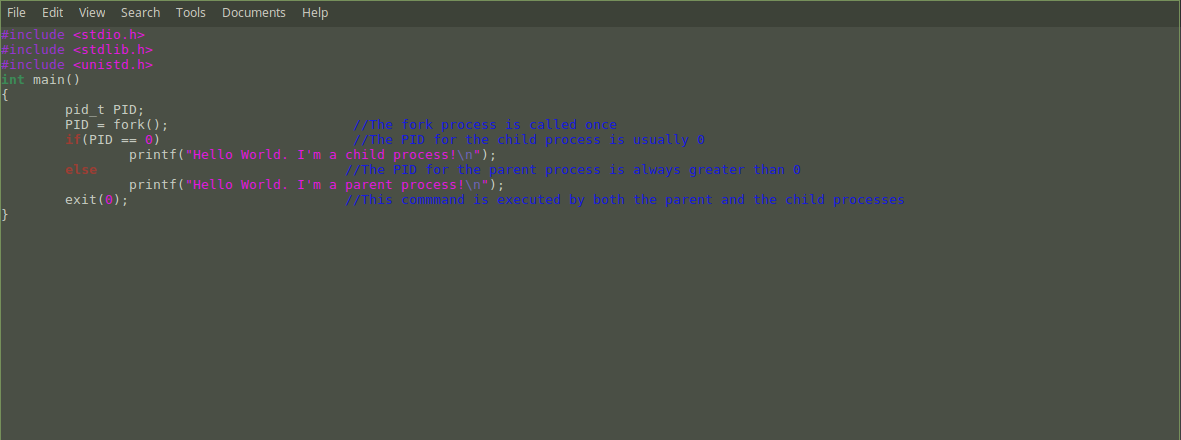
1. The following are some process properties:
2. TTY. This is an identifier for the terminal session that triggered a process, in other words the controlling terminal. Most processes, except daemons, are attached to a terminal.
3. PID, process identifier. Every process has an id attached to it. It is a unique identifier that enables one to reference a specific process.
4. PPID, parent process identifier. Almost every process has a parent process, the process that was responsible for creating it. This is a unique identifier for that process.
5. UID, user id. It is the identifier for the user that is the owner of the process. It defines the permissions that the process will have.
6. ARGS. This is the command, followed by its arguments, that is running in a process.
7. Processes are born via two steps in a UNIX system: the fork and the exec.

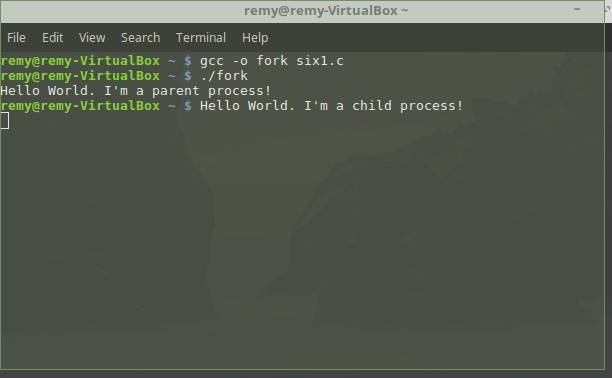
Every process is created using the fork system call. Fork creates a copy of the calling process, called the parent process and the newly created process is called the child process. The child inherits everything that the parent has in memory, except for its identifier.

The second step is the exec. It replaces the current process with a new one. The caller process is gone, and the new process takes its place within the same environment. This will replace all the text, data, stack segments and heap of the caller process with that of the new process called.

1. Some types of processes include:
2. Daemon. This is a process that runs in the background and is not attached to any controlling terminal, examples are database and web servers. This process runs from system boot to system shutdown.
3. Zombie. This is a process which has finished execution but still has entry in the process table to report to its parent process. A child process always first becomes a zombie before being removed from the process table.
4. Orphan. This is a process whose parent process no more exists either because it has finished or terminated without waiting for it to terminate. A child process is soon adopted by the init process (the process that starts all other processes) once its parent process dies.
5. To create a new process in UNIX, the fork() system call is used. It creates a new context based on the context of the existing process. It returns twice in that it returns in both the process calling fork(), the parent process and in the newly created process, the child process. The child process returns zero and the parent process returns a number greater than zero.

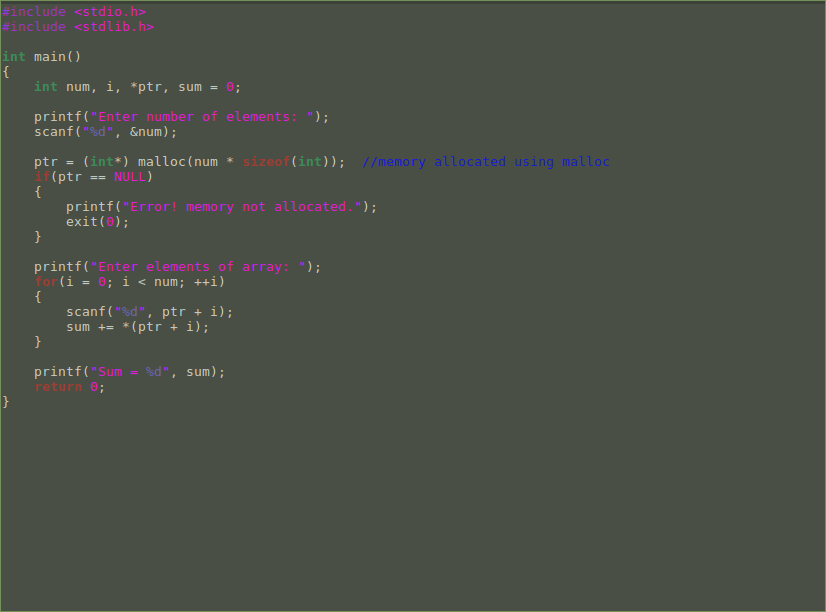
Below is a program to demonstrate that fork is called once but returns twice:

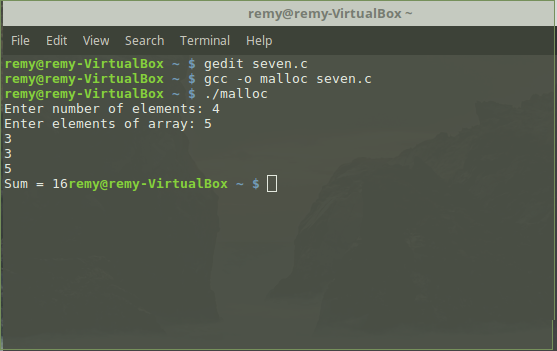




1. Dynamic memory allocation uses a pointer variable to request memory from the Operating System while the program is running. The malloc function is used to allocate a block of memory of specified size. This block of memory is obtained from an area of memory called the heap. It returns a pointer of type void, so we can assign it to any type of pointer variables. The malloc function will return NULL if it fails to allocate the required memory space.

Below is a program to demonstrate dynamic memory allocation using malloc:





In UNIX, processes are created using the fork() system call. When a program calls fork(), a new process is created which is an exact copy of the parent process, except for a few differences such as a unique process ID (PID) and a different return value from the fork() call.

The new process, known as the child process, starts executing from the point where the fork() call was made. The parent process continues executing as normal, with the added knowledge that it has successfully created a child process.

Once the child process has been created, it can execute a new program using the exec() system call. This replaces the child process's memory space with the memory space of the new program, effectively starting the new program within the child process.

Processes can also be created using other system calls, such as clone() and vfork(), which have different features and behaviors. However, fork() is the most commonly used system call for creating new processes in UNIX.