PROCESS SYSTEM CALL

`fork()`

- **The fork system call is used to create a new process called *child* process.**
  - **The return value is 0 for a child process.**
  - **The return value is negative if process creation is unsuccessful.**
  - **For the parent process, return value is positive**
- **The child process is an exact copy of the parent process.**
- **Both the child and parent continue to execute the instructions following fork call.**
- **The child can start execution before the parent or vice-versa.**

`getpid()` and `getppid()`

- **The getpid system call returns process ID of the calling process**
- **The getppid system call returns parent process ID of the calling process**

`wait()`

- **The wait system call causes the parent process to be blocked until a child terminates.**
- **When a process terminates, the kernel notifies the parent by sending the SIGCHLD signal to the parent.**
- **Without wait, the parent may finish first leaving a *zombie* child, to be adopted by init process**

`execl()`

- **The exec family of function (execl, execv, execle, execve, execlp, execvp) is used by the child process to load a program and execute.**
- **execl system call requires path, program name and null pointer**

`exit()`

- **The exit system call is used to terminate a process either normally or abnormally**
- **Closes all standard I/O streams.**

`stat()`

- **The stat system call is used to return information about a file as a structure.**

`opendir()`, `readdir()` and `closedir()`

- **The opendir system call is used to open a directory**
  - **It returns a pointer to the first entry**
  - **It returns NULL on error.**
- **The readdir system call is used to read a directory as a *dirent* structure**
  - **It returns a pointer pointing to the next entry in directory stream**
  - **It returns NULL if an error or end-of-file occurs.**
- **The closedir system call is used to close the directory stream**
- **Write to a directory is done only by the kernel.**

Exp# 1a **`fork` system call**

Aim

To create a new child process using fork system call.

Algorithm

1. Declare a variable *x* to be shared by both child and parent.
2. Create a child process using fork system call.
3. If return value is -1 then
   a. Print "Process creation unsuccessfull"
   b. Terminate using exit system call.
4. If return value is 0 then
   a. Print "Child process"
   b. Print process id of the child using getpid system call
   c. Print value of *x*
   d. Print process id of the parent using getppid system call
5. Otherwise
   a. Print "Parent process"
   b. Print process id of the parent using getpid system call
   c. Print value of *x*
   d. Print process id of the shell using getppid system call.
6. Stop

Result

Thus a child process is created with copy of its parent's address space.