fork() System Call

The `fork()` system call creates a new process in UNIX. It takes no arguments and returns a process ID (`pid_t`).

Behavior of `fork()` return values:

- **Negative value**: Fork failed.
- **Zero**: Returned to the newly created child process.
- **Positive value**: Returned to the parent process with the child's process ID.

The parent and child processes have separate address spaces but initially contain identical copies of the program.

---

ps Command

The `ps` command displays currently running processes.

Example:

```
$ ps -ef
```

- `-e` : Show all processes.
- `-f` : Full format display.

---

exec() System Call

The `exec()` system call replaces the memory space of the calling process with a new program.

Purpose:

- Used after `fork()` to load a new executable.
- Various `exec` functions exist (`execvp()` , `execl()` , etc.) with slight variations.

---

wait() System Call

The `wait()` system call makes the parent process wait for its child process to complete execution.

Behavior:

1. If a child process is running, the parent is blocked until the child exits.
2. If no child exists, `wait()` has no effect.

Zombie Process

A **zombie process** is a child process that has terminated but still occupies an entry in the process table until the parent calls `wait()`.

Orphan Process

An **orphan process** is a child process whose parent has terminated, leaving it running. These are often adopted by the `init` system.

Daemon Process

A **daemon process** is a background process that runs independently of user sessions, often used for system services.

Example: Simple `fork()` System Call

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    fork();
    printf("Hello world!\n");
    return 0;
}
```

Output:

```
Hello world!
Hello world!
```

Example: `fork()` with Process IDs

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    pid_t pid, mypid, myppid;

    pid = getpid();
    printf("Before fork: Process ID is %d\n", pid);

    pid = fork();
    if (pid < 0) {
        perror("fork() failure\n");
        return 1;
    }

    if (pid == 0) { // Child process
        printf("This is child process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process ID: %d, Parent ID: %d\n", mypid, myppid);
    } else { // Parent process
        sleep(2);
        printf("This is parent process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process ID: %d, Parent ID: %d\n", mypid, myppid);
        printf("Newly created child process ID: %d\n", pid);
    }
    return 0;
}
```