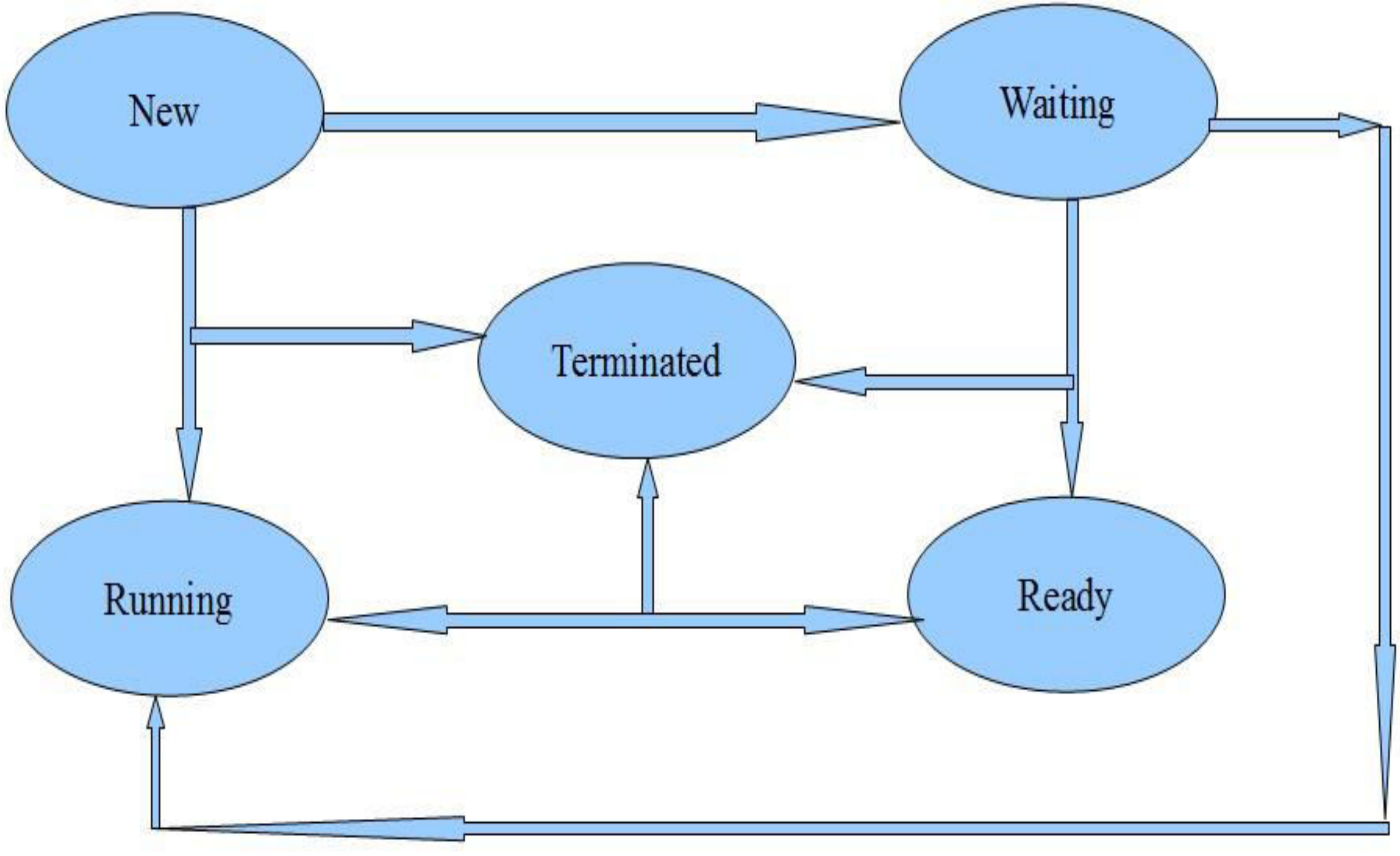


Fig 1: The Interconnections of Process States

CS-3103 : Operating Systems : Sec-A (NB) : Little bit about process ➔ fork() ….

CS3103 - Operating Systems - The Process

- A process is simply an instance of a running program.
- A process is said to be ***created*** (***born)*** when the program starts execution and remains alive as long as the program is active.
- After execution is complete, the process is said to terminate (***die)***.
- A process also has a name, usually the name of the program being executed.
- For example, when you execute the ***grep*** command, a process named ***grep*** is created.
- ***Process is the name given to a file when it is executed as a program***.

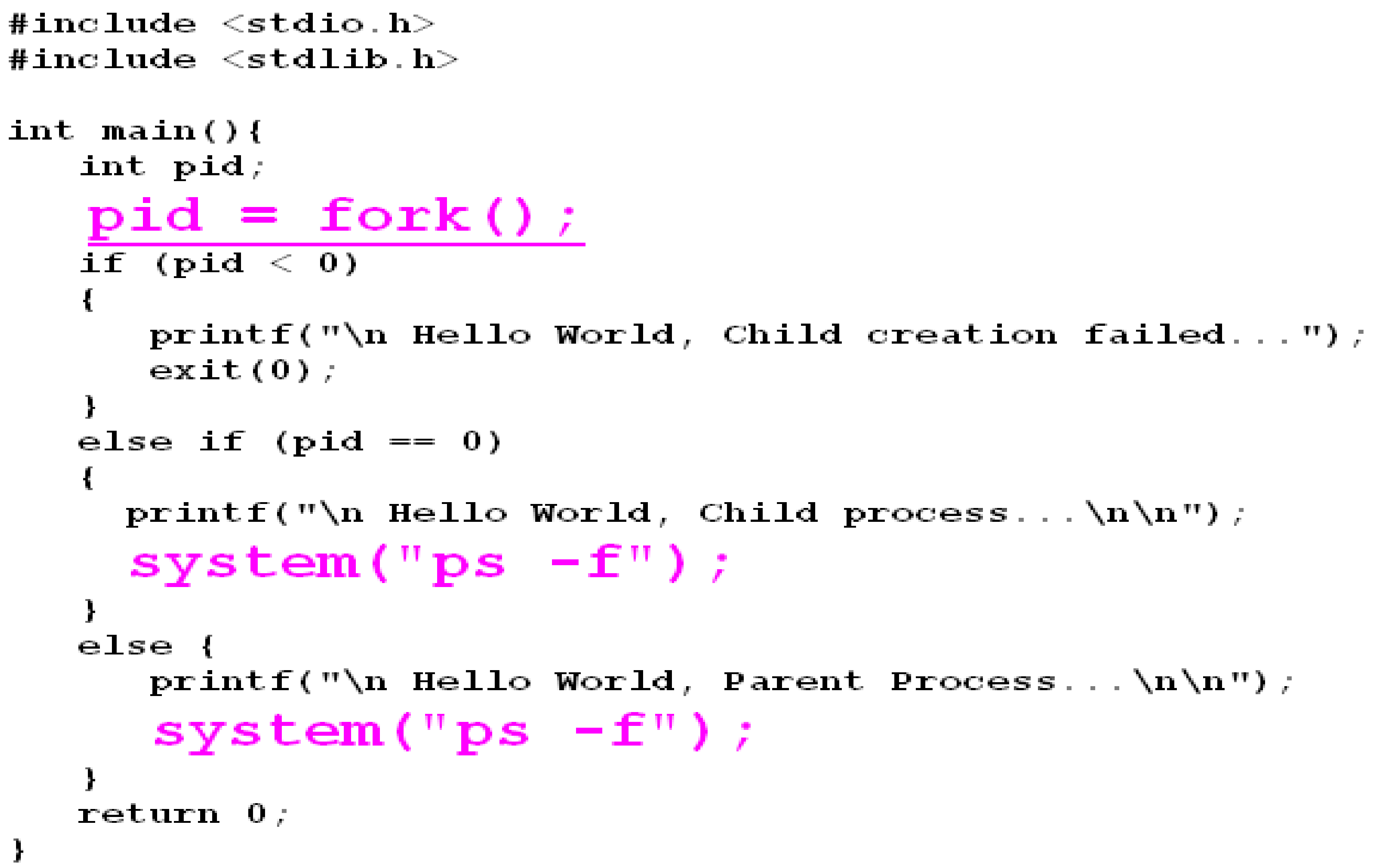
Mechanism of a Process Creation

Fork: A process in UNIX is created with the **fork** system call, which creates a copy of the process that invokes it. The process image is identical to that of the calling process, except for a few parameters like the **PID**. When a process is forked in this way, the child gets a new **PID**.

Exec: Forking creates a process but not enough to run a new program. Through ***exec***, the child process is said to *exec* a new program. No new process is created here, the **PID** and **PPID** of the exec'd process remain unchanged.

Wait: The parent executes the *wait* system call to wait for the child process to complete.

OS – Process Creation



```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int pid;
    pid = fork();
    if (pid < 0)
    {
        printf("\n Hello World, Child creation failed...");
        exit(0);
    }
    else if (pid == 0)
    {
      printf("\n Hello World, Child process...\n\n");
      system("ps -f");
    }
    else {
        printf("\n Hello World, Parent Process...\n\n");
        system("ps -f");
    }
    return 0;
}
```

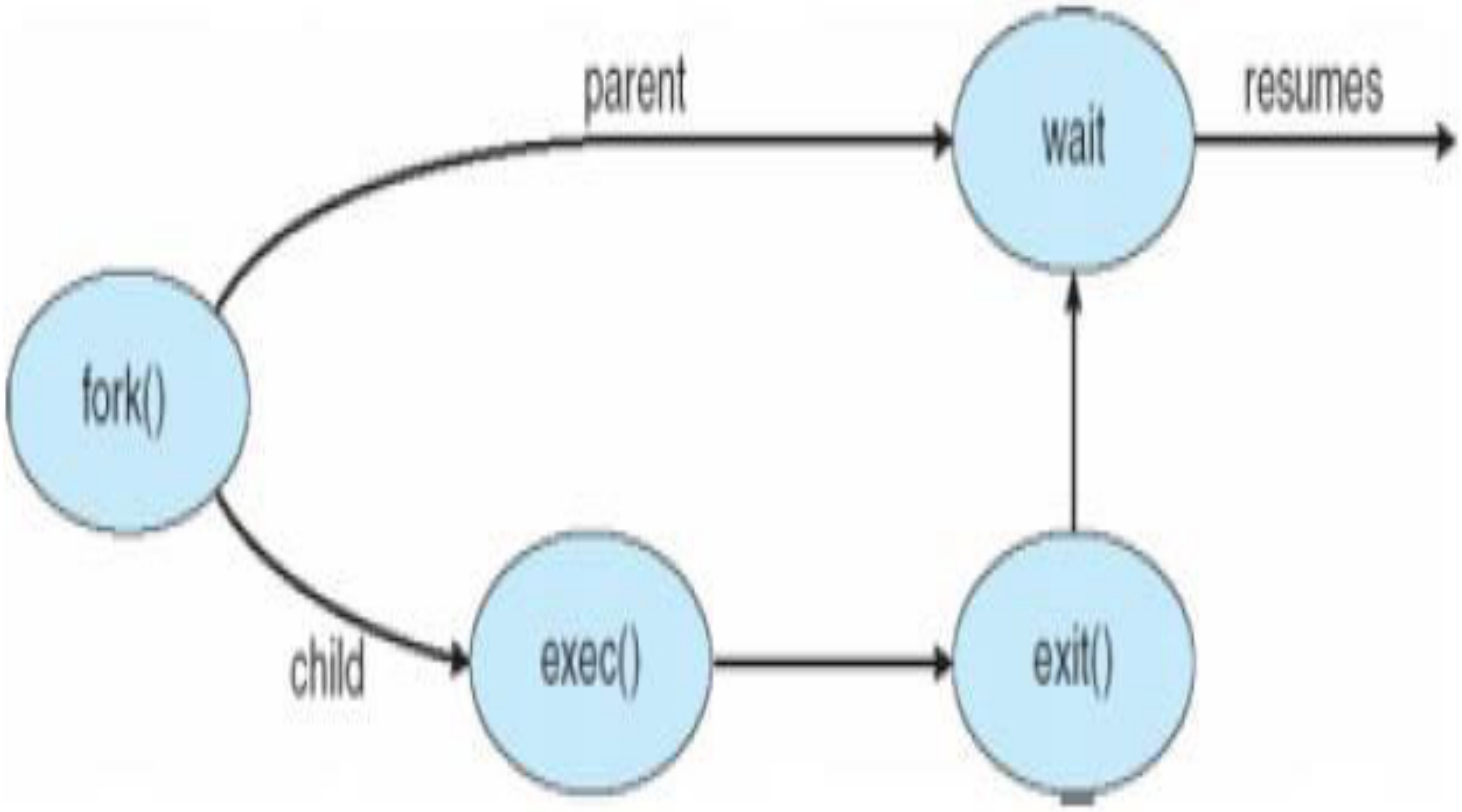
OS – Process Creation

```
Q1. nilina@nilina-HP-Pro-3330-MT:~/Desktop/cs693$ gcc processcreation.c
nilina@nilina-HP-Pro-3330-MT:~/Desktop/cs693$ ./a.out processcreation

 Hello World, Parent Process...
 Hello World, Child process...

UID      PID   PPID   C STIME   TTY        CMD
nilina   2150  2143   0 12:39   pts/0      bash
nilina   2300  2150   0 13:02   pts/0     ./a.out processcreation
nilina   2301  2300   0 13:02   pts/0     [a.out] <defunct>
nilina   2302  2300   0 13:02   pts/0     sh -c ps -f
```

Process Creation (Cont.)



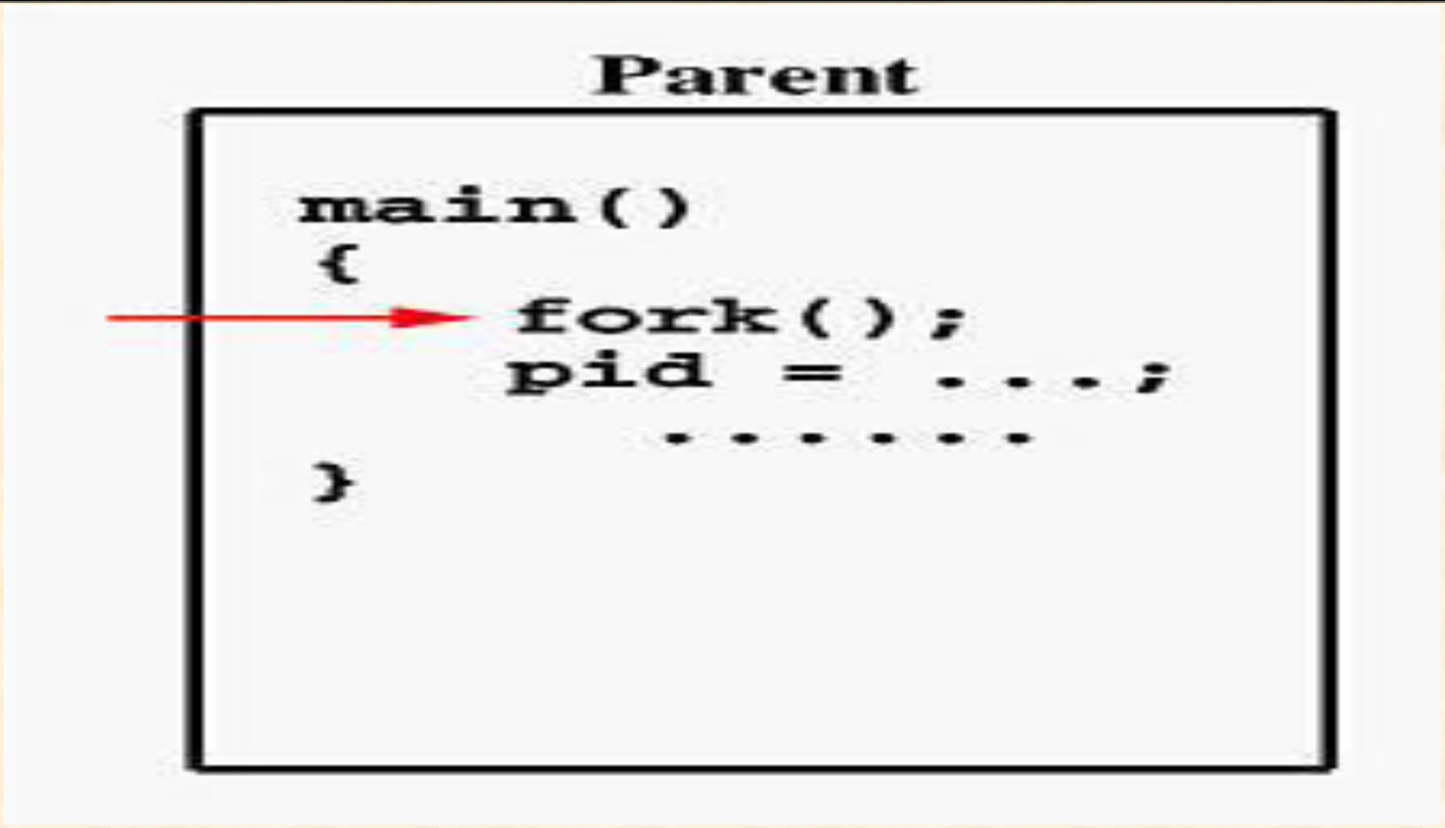
The fork() System Call

- System call **fork()** directs Unix to spawn a sub-task to service the request.
- 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.

The fork() System Call

- 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.

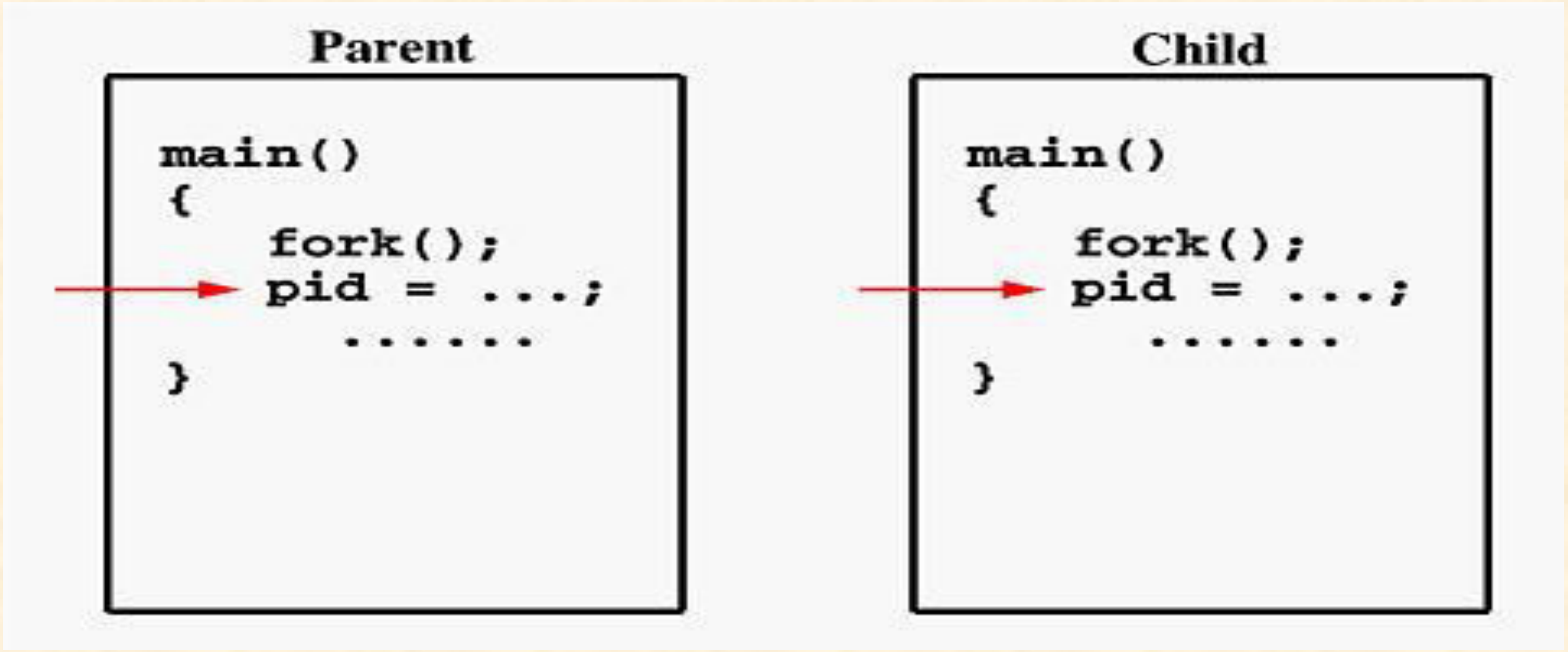
fork – create Parent and Child processes



```
main()
 {
    fork();
    pid = ...;
       ......
 }
```

If the call to **fork()** is executed successfully, Unix will make two identical copies of address spaces, one for the parent and the other for the child.

fork – create Parent and Child processes



```
main()
{
    fork();
    pid = ...;
     ......
}
```

- Since every process has its own address space, any modifications will be independent of the others. In other words, if the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space.
- Other address spaces created by **fork()** calls will not be affected even though they have identical variable names.