OPERATING SYSTEM

Fork() System Call – Definition

The **`fork()` system call** is used in UNIX/Linux systems to **create a new process**.
It creates a **child process** that is almost an exact copy of the **parent process**.

After calling `fork()`:

- The parent process gets the **child’s PID** (a positive value)
- The child process gets **0**
- If creation fails, it returns **-1**

How fork() Works

- `fork()` returns 0 → Inside child process
- `fork()` returns positive PID → Inside parent process
- `fork()` returns -1 → Process creation failed

After `fork()` both processes continue execution **from the next instruction**.

Process Diagram After fork()

```
               Parent Process
                     |
                  fork()
                     |
        +------------+-------------+
        |                          |
Parent continues             Child process starts
return > 0                       return = 0
```

Python Example Using os.fork()

Works on **Linux/Unix/macOS** (NOT on Windows)
Demonstrates parent and child processes

```
import os

pid = os.fork()  # Create a new process

if pid == 0:
   # This block runs in the child process
   print("This is the Child Process")
   print("Child PID:", os.getpid())
else:
   # This block runs in the parent process
   print("This is the Parent Process")
   print("Parent PID:", os.getpid())
   print("Child PID returned by fork():", pid)
```

Key Points About fork()

- Creates a new, separate process.
- Child has a different PID from parent.
- Both parent and child execute simultaneously.
- Child inherits files, memory (copy-on-write), and environment.
- Used for process creation in UNIX/Linux systems.

Short Exam-Ready Definition

`fork()` is a system call used to create a new process (child process) from an existing process (parent). It returns 0 inside the child and returns the child's PID inside the parent. After `fork()`, both processes execute independently from the next instruction.