Real-Time Operating System (48450)

Week 3 Lab Exercises

Handling Signals

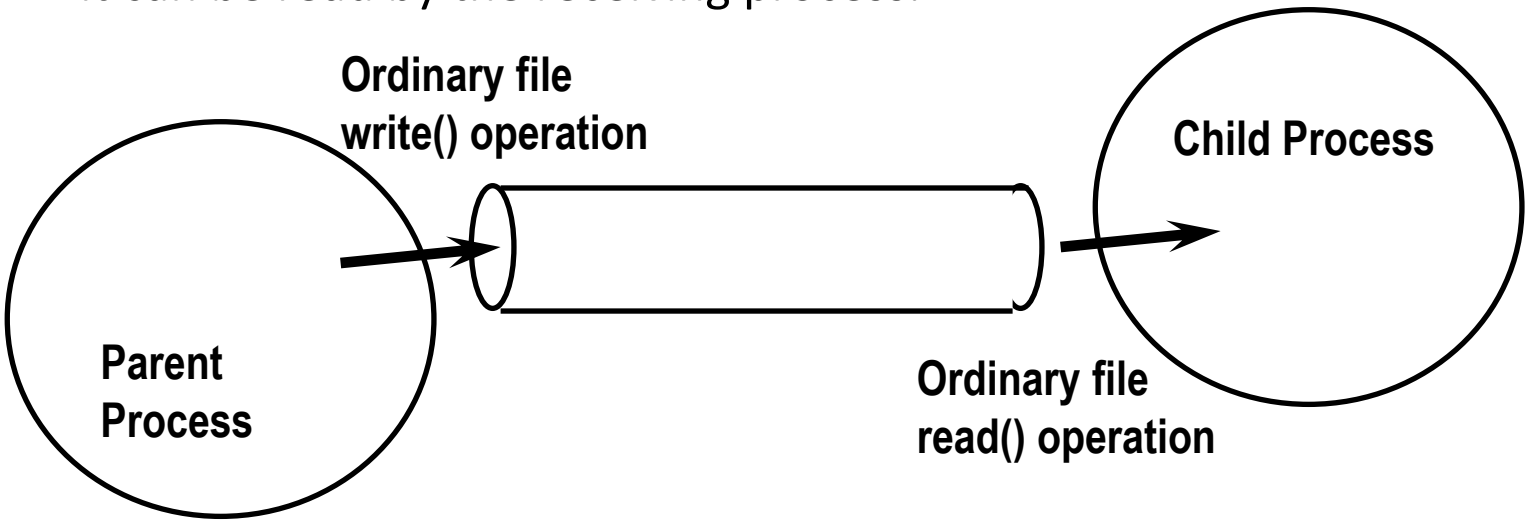
- The kernel handles signals in the context of the process that receives them, so a process must run to handle signals.
- There are three types of handling for signals:
  - The process exits. (default for some signals)
  - The process ignores. (default for some other signals)
  - The process executes a signal handler:

```
#include <signal.h>
int sigaction (int signum, const struct
  sigaction *act, struct sigaction *oact);
```

Note: *signum* specifies the signal and can be any valid signal except SIGKILL and SIGSTOP.

Pipes (POXIX.1)

- A pipe is a method used to pass information from one program process to another or **from one thread to another**
- Unlike other types of inter-process communication (IPC), a pipe only offers one-way communication by passing a parameter or output from one process to another or output from one thread to another.
- The information that is passed through the pipe is held by the system until it can be read by the receiving process.



https://www.youtube.com/watch?v=uHH7nHkgZ4w

4.11 For multi-**processes**, is it possible to have concurrency but not parallelism? Explain.

Answer:

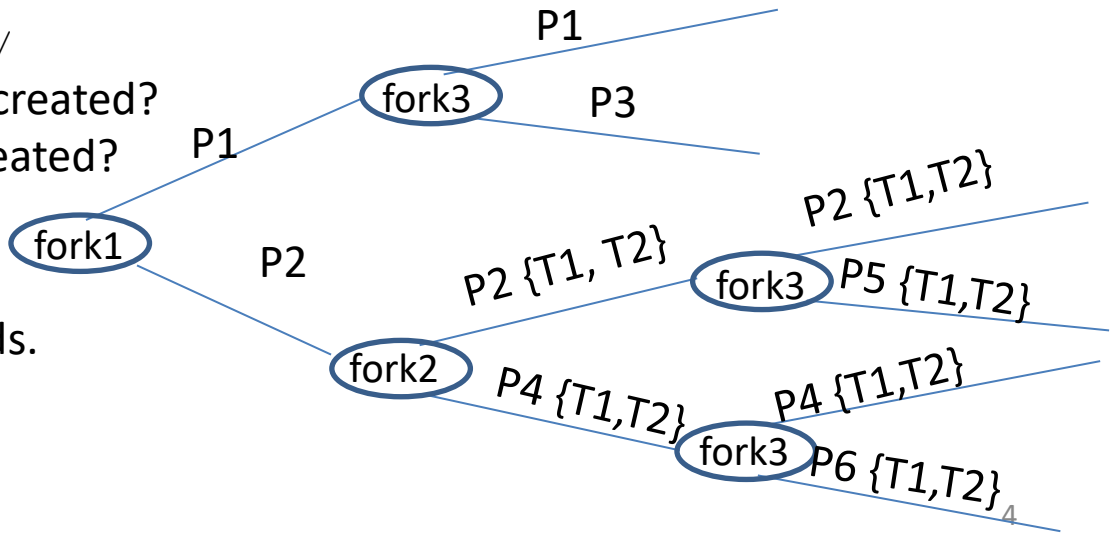
Yes. Concurrency means that more than one process (or thread) is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core (multi-core processor architecture).

4.15 Consider the following code segment:

```
pid t pid;
pid = fork(); /*fork 1 */
if (pid == 0) { /* child process */
     fork(); /*fork 2 */
     thread create( . . .);
}
fork(); /*fork 3 */
```

a. How many unique processes are created?

b. How many unique threads are created?



Answer:

There are six processes and two threads.

Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**

- **Parallelism on a multi-core system:**