Lab 5 - Interprocess Communication

Interprocess Communication

- It's useful for a parent process to communicate with its child (and vice versa)
- There are two key ways we will learn to do this: pipes and signals
  - Pipes let two processes send and receive arbitrary data
  - Signals let two processes send and receive certain "signals" that indicate something special has happened.

Pipes

- A pipe is a set of two file descriptors representing a "virtual file" that can be written to and read from
- It's not actually a physical file on disk - we are just using files as an abstraction
- Any data you write to the write FD can be read from the read FD
- Because file descriptors are duplicated on fork(), we can create pipes that are shared across processes!

This method of communication between processes relies on the fact that file descriptors are duplicated when forking.

- Each process has its own copy of both file descriptors for the pipe
- both processes could read or write to the pipe if they wanted
- each process must therefore close both file descriptors for the pipe when finished

This is the core idea behind how a shell can support piping between processes (e.g. cat file.txt | uniq | sort).

dup2

The dup() system call allocates a new file descriptor that refers to the same open file description as the descriptor oldfd. The new file descriptor number is guaranteed to be the lowest-numbered file descriptor that was unused in the calling process.

```
int dup2(int oldfd, int newfd);
```

Virtual Memory

- Core Idea: have the processes work with virtual ("fake") addresses. The OS will decide what physical addresses they actually are.

- Challenges with multiple processes running:

  - how do we partition memory? -> The OS decides as it goes
  - what if one process accesses the memory of another? -> Virtual address spaces are separate and monitored by OS
  - what if we run out of physical memory? -> OS can play tricks to swap memory to disk when needed, and map addresses only on demand.

Programming Exercises

1. pipe.cpp : This program demonstrate a basic use of pipe.
2. pipe_fork.cpp : This program shows how pipe works across child and parent processes.
3. subprocess.cpp : Implements the subprocess routine, which is similar to popen and allows the parent process to spawn a child process, print to its standard output, and then suspend until the child process has finished.
4. pipeline.cpp : This example presents the implementation of the pipeline routine, which is similar to subprocess and allows the parent process to spawn two child processes, the first of which has its stdandard output forwarded to the second child's standard input. The parent then waits for both children to finish.