CENG328
Operating Systems

Laboratory VIII
Interprocess Communications

1. Interprocess Communications

- We discuss five types of interprocess communication:
  - **Shared memory** permits processes to communicate by simply reading and writing to a specified memory location. (We already discussed.)
  - **Mapped memory** is similar to shared memory, except that it is associated with a file in the filesystem. (We will not discuss.)
  - **Pipes** permit sequential communication from one process to a related process.
  - **FIFO**s are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the filesystem.
  - **Socket**s support communication between unrelated processes even on different computers.

1. Interprocess Communications

- **Pipe**; code40.c
  - A fork spawns a child process.
  - The child inherits the pipe file descriptors.
  - The parent writes a string to the pipe, and the child reads it out.
  - The program converts these file descriptors into FILE* streams using fdopen.
  - Why fflush is used in the function writer?

- Another example for pipe; code41.c and code42.c
  - One process sends a set of letters by means of writing to pipe.
  - Other process reads this input from pipe and reports the number of lowercase and uppercase characters in this set.
  - You should supply an argument to seed the random number generator.
  - Execute several times by changing the seed each time.

1. Interprocess Communications

- A **first-in, first-out** (**FIFO**) file is a pipe that has a name in the filesystem.
  - Any process can open or close the FIFO; the processes on either end of the pipe need not be related to each other.
  - FIFOs are also called named pipes.
  - You can make a FIFO using the mkfifo command:
    mkfifo /tmp/fifo
    ls -l /tmp/fifo
  - The first character of the output from **ls** is **p**, indicating that this file is actually a FIFO.
  - In one window, read from the FIFO by invoking the following:
    cat < /tmp/fifo
  - In a second window, write to the FIFO by invoking this:
    cat > /tmp/fifo
  - Then type in some lines of text. Each time you press Enter, the line of text is sent through the FIFO and appears in the first window.
  - Close the FIFO by pressing **CTRL + D** in the second window.
  - Remove the FIFO with this line:
    rm /tmp/fifo

1. Interprocess Communications

- Creating a FIFO; create a FIFO programmatically using the **mkfifo** function. Include **sys/types.h** and **sys/stat.h** if you call mkfifo.
  - Accessing a FIFO; access a FIFO just like an ordinary file. To communicate through a FIFO, one program must open it for writing, and another program must open it for reading.
    - To write a buffer of data to a FIFO using low-level I/O routines, you could use this code:

```
int fd = open (fifo_path, O_WRONLY);
write (fd, data, data_length);
close (fd);
```

    - To read a string from the FIFO using C library I/O functions, you could use this code:

```
FILE* fifo = fopen (fifo_path, "r");
fscanf (fifo, "%s", buffer);
fclose (fifo);
```

- Write a program that creates a FIFO and access to that FIFO.

1. Interprocess Communications

- Sockets are more flexible than previously discussed communication techniques. These are the system calls involving sockets:
  - **socket** - Creates a socket
  - **close** - Destroys a socket
  - **connect** - Creates a connection between two sockets
  - **bind** - Labels a server socket with an address
  - **listen** - Configures a socket to accept conditions
  - **accept** - Accepts a connection and creates a new socket for the connection.

1. Interprocess Communications

- Sockets are represented by file descriptors. Using Local Namespace Sockets (we also have network sockets):
  - Two programs; the server program code43.c creates a local namespace socket and listens for connections on it.
    - When it receives a connection, it reads text messages from the connection and prints them until the connection closes.
    - If one of these messages is "**quit**", the server program removes the socket and ends.
    - The socket-server program takes the path to the socket as its command-line argument.
  - The client program code44.c connects to a local namespace socket and sends a message. The name path to the socket and the message are specified on the command line.
  - List the files and see the socket during communication. The first character of the output from **ls** is **s**, indicating that this file is actually a socket.