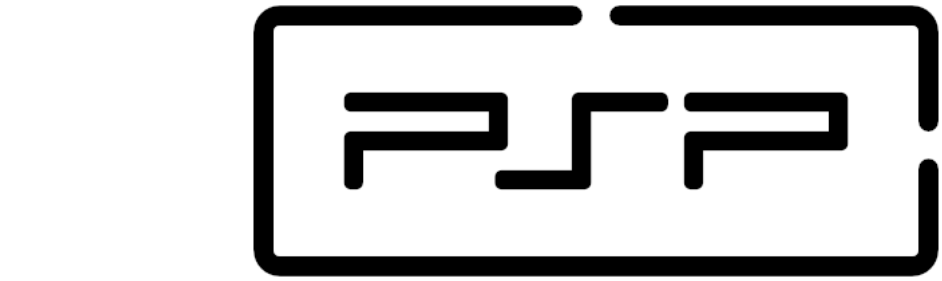

Process and Service Programming

2.1 Interprocess communication

2.1 Interprocess communication

Interprocess communication (IPC) is one of the main features of operating systems. In this section, we will focus on the communication between processes that are on the same device.

2.1.1. Communication through I/O

Communication between processes can be done in many ways, but one of the simplest and most common is communication through standard input and output.

> **I/O in Java**
>
> In Java, communication through standard input and output is done through standard input and output streams, `System.in` and `System.out` respectively.

Every process has three standard input and output streams that can be used for communication with other processes. These streams are:

- **stdin** (standard input): where the process receives data. By default, it corresponds to the keyboard and the file identifier associated with it is 0.
- **stdout** (standard output): where the process sends data. By default, it corresponds to the console and the file identifier associated with it is 1.
- **stderr** (standard error output): where the process sends error messages. By default, it corresponds to the console and the file identifier associated with it is 2.

A relatively simple IPC mechanism is the communication of processes through the redirection of standard inputs and outputs to/from other sources.

> **I/O redirection**
>
> The redirection of standard input and output can be done on the command line of UNIX and Windows systems. In Java, it can be done using the `ProcessBuilder` class that we will see in the next section of the unit.

Standard input redirection

Standard input redirection can be done using the `<` operator in UNIX and Windows systems.

```sh
$> java MyClass < input.txt
```

In the previous example, the `MyClass` program receives standard input from the `input.txt` file instead of from the keyboard.

> When standard input is redirected, the program does not have to do anything special to read from a file instead of from the keyboard. The operating system takes care of redirecting the standard input of the program to the file that is indicated.

Standard output redirection

Standard output redirection can be done using the `>` and `>>` operators in UNIX and Windows systems.

```sh
$> java MyClass > output.txt
$> java MyClass >> output2.txt
```

In the previous example, the standard output of the `MyClass` program is redirected to the `output.txt` file instead of to the console. If the `output.txt` file does not exist, it is created, and if the file already exists, its value is overwritten.

If the operator is `>>`, the output is added to the end of the file instead of overwriting it.

> When standard output is redirected, the program does not have to do anything special to write to a file instead of to the console. The operating system takes care of redirecting the standard output of the program to the file that is indicated.

Standard error output redirection

Standard error output redirection can be done using the `2>` operator in UNIX and Windows systems.

```sh
$> java MyClass 2> error.txt
$> java MyClass 2>> error2.txt
```

In the previous example, the standard error output of the `MyClass` program is redirected to the `error.txt` file instead of to the console.

If the operator is `2>>`, the error output is added to the end of the file instead of overwriting it.

> When the error output is redirected, the program does not have to do anything special to write to a file instead of to the console. The operating system takes care of redirecting the error output of the program to the file that is indicated.

Standard input redirection

Standard input redirection can be done using the `<` operator in UNIX and Windows systems.

```sh
$> java MyClass < input.txt
```

In the previous example, the `MyClass` program receives standard input from the `input.txt` file instead of from the keyboard.

> When standard input is redirected, the program does not have to do anything special to read from a file instead of from the keyboard. The operating system takes care of redirecting the standard input of the program to the file that is indicated.
>
> ### Standard output redirection
> Standard output redirection can be done using the `>` and `>>` operators in UNIX and Windows systems.

```sh
$> java MyClass > output.txt
$> java MyClass >> output2.txt
```

In the previous example, the standard output of the `MyClass` program is redirected to the `output.txt` file instead of to the console. If the `output.txt` file does not exist, it is created, and if the file already exists, its value is overwritten.

If the operator is `>>`, the output is added to the end of the file instead of overwriting it.

> When standard output is redirected, the program does not have to do anything special to write to a file instead of to the console. The operating system takes care of redirecting the standard output of the program to the file that is indicated.

Standard error output redirection

Standard error output redirection can be done using the `2>` operator in UNIX and Windows systems.

```sh
$> java MyClass 2> error.txt
$> java MyClass 2>> error2.txt
```

In the previous example, the standard error output of the `MyClass` program is redirected to the `error.txt` file instead of to the console.

If the operator is `2>>`, the error output is added to the end of the file instead of overwriting it.

> When the error output is redirected, the program does not have to do anything special to write to a file instead of to the console. The operating system takes care of redirecting the error output of the program to the file that is indicated.

2.1.2. Redirection of the output of one process to the input of another process

The redirection of standard output to the standard input of another process can be done using the `|` operator in UNIX and Windows systems.

Pipes allow you to connect the standard output of one process to the standard input of another, thus establishing a producer-consumer relationship.

The use of pipes follows the following syntax:

```sh
$> java MyClass | java MyClass2
```

In the previous example, the standard output of the `MyClass` program is redirected to the standard input of the `MyClass2` program.

> When the standard output of one process is redirected to the standard input of another, the operating system takes care of connecting the output and input streams of the processes.

2.1.3. Communication through signals

Signals are a form of communication between processes that is based on interrupting the execution of a process to perform a specific action.

Signals are asynchronous events that are sent to a process to notify it of an event. Signals can be sent by the process itself, by another process, or by the operating system.

Signals can be sent to a process using the `kill` command in UNIX systems.

```sh
$> kill -s SIGUSR1 1234
```

In the previous example, the `SIGUSR1` signal is sent to the process with PID `1234`.

Signals in the Windows shell can be sent using the `taskkill` command.

```sh
$> taskkill /pid 1234 /f
```

In the previous example, the forced termination signal is sent to the process with PID `1234`.

Signals

You can look at the UNIX signal list (https://en.wikipedia.org/wiki/Signal_(IPC)) in Wikipedia.

And you an read more on Gestión de procesos en Windows (https://openwebinars.net/blog/gestion-de-procesos-y-servicios-desde-shell-script-en-windows/) .

2.1.4. Communication through sockets

Sockets can be used for communication between processes on the same device or on different devices.

Sockets will be studied in Unit 4, where we will see how they can be used for communication between processes on different devices.