Lab 6: I/O Redirection and Pipes

Standard Input/Output Redirection

Standard input and output are the "natural" feed (input) and sink (output) streams of a program: User input is read from the keyboard, and the program's output is displayed on monitor. However, in many cases one would like to read input from a file, rather than from keyboard. Similarly, one may wish to store the output in a file, rather than display it on monitor. Both requirements can be met by standard input/output redirection, without changing the code of the original program.

As a convention, input redirection is triggered by adding "<" after the invoked command. For instance, **"cat < my.txt"** tells the shell program to redirect the input of **cat** to file **my.txt**. As a result, cat displays the content of **my.txt** instead of displaying user feed from the keyboard. Output is triggered by adding ">", for instance: **"ls > out.txt"** which stores the list of files in the current directory in **out.txt**. Combining input and output redirection is also possible, e.g. **"cat < my.txt > yours.txt"**, which stores the content of **my.txt** in **yours.txt**.

How does the shell program achieve such an effect? Simply by closing the standard input stream (file decriptor 0) or the standard output stream (file decriptor 1), and opening a new file, which in turn is automatically allocated with the lowest available file-descriptor index. This effectively overrides stdin or stdout.

Introduction to Pipelines

Pipelining is an extremely important concept in system programming. The name "pipeline" itself suggests that pipes are, somehow, involved. But what exactly is a pipeline? This is most easily explained by example, and in our case, by typing **"echo spider-pig | head -c 6"** in the Linux shell.

The output of this command, as you will notice, is the string **spider**. However, if you execute **"echo spider-pig"** and **"head -c 6"** separately, you see something quite different. The first command simply echos **spider-pig**, and the second command asks for user input, and prints the first 6 characters. One can easily deduce that when the two commands are chained using the "|" symbol, the output of **echo** is directly fed

into **head**.

And this is where pipes come into play. The two processes communicate using a pipe which they both share: The standard output of **echo** is redirected to the pipe's "write-end", and the standard input of **head** is redirected to the "read-end". A pipeline, then, is simply a chain of processes where the standard output of one process feeds the standard input of the following process. The principle of pipelines easily generalizes to an unlimited number of processes. For instance, the command **"echo spider-pig | cat | head -c 6"** entails a pipeline consisting of three processes (**echo**, **cat** & **head**) and two pipes.