Operating Systems
Pipes

Distributed Systems Group
University of Minho

1 Objectives

Become familiar with and use system calls relating to communication between processes via anonymous pipes.

2 System Calls

```
#include <unistd.h>     /* system calls: essential defs and decls */

int pipe(pd[2]);
```

3 Exercises

1. Write a program that creates an anonymous pipe and then creates a child process. Try using the parent to send an integer via the pipe's write descriptor, while using the child process to read that integer from the pipe's read descriptor.

   (a) Try causing a delay before the parent sends the integer (e.g., `sleep(5)`).
   Note that the child's read blocks as long as the parent does not perform the the write operation on the pipe.

2. Now try reversing the roles so that the information is transmitted from the child to the parent.

   (a) Try causing a delay before the parent reads the integer. Repeat with a sequence of integers.
   Note that the child's write blocks until the parent performs the read operation on the pipe.

   (b) Modify the previous program so that the pipe is read as long as an *end of file* is not detected in its descriptor. Note that this situation only occurs when no process – in this case, parent and child – have opened the pipe's write descriptor.

3. You want to determine all the occurrences of a given integer in the rows of a matrix of integers where the number of columns is much greater than the number of rows. Implement, by using processes and *pipes*, a function that returns all the occurrences found in a vector. The initial matrix, the value to search for and the vector in which to store the results must be supplied as parameters.

4 Additional exercises

1. Implement a program that reads a very large text file and prints the lines that contain a keyword. Use the `str()` function to search for the word on each line. Use several processes and avoid storing the entire file in memory.