PL3 - Pipes

Luís Nogueira, Orlando Sousa

{lmn, oms}@isep.ipp.pt

Sistemas de Computadores
2024/2025

1. Implement a program that creates a new process and establishes a connection with it using a pipe.
   - The parent process begins by printing its PID and then sends it to the child process through the pipe.
   - The child process should read the parent´s PID from the pipe and print it.
2. Implement a program that creates a new process and establishes a connection with it using a pipe.
   - The parent process reads an integer number and a name from the keyboard, then sends this data to the child process through the pipe.
   - The child process should read both values from the pipe and display them on the screen.

   a) First, solve the exercise by sending the number and then the name in two separate write instructions.

   b) Then, solve the exercise using a struct to send both values in a single write operation.
3. Implement a program that creates a new process and establishes a connection with it using a pipe.
   - The parent process sends the contents of a text file to the child through the pipe.
   - The child process reads the data from the pipe and displays it on the screen.

   All file descriptors not used by the processes should be closed.

4. Implement a program where two processes, client (the child process) and server (the parent process), communicate via two pipes "Up" and "Down".

```
Server (parent process)
   ^ Up      |
   |         v Down
Client (child process)
```

   - The client reads a message from `stdin` and sends it to the server via the "Up" pipe, then waits for the server's answer on the "Down" pipe.
   - The server is specialized in converting characters from lower case to upper case and vice-versa. Therefore, if the client sends the message "lower case" via the "Up" pipe, the server will read the message, convert it, and send "LOWER CASE" via the "Down" pipe.
   - When the client receives the message from the server, it prints it out.

   You may assume the maximum size of any message is `PIPE_BUF` bytes.

5. Given two arrays of integers, each with 1000 elements, implement a program that creates five new processes.

   - Each child process is responsible for adding 200 positions (`vec1[i] + vec2[i]`) and then sending the result of the partial sum to its parent through a pipe.
   - The parent process should read from the pipe the five partial sums and calculate the final result.

   Use only one pipe, shared between the parent and the five children. Note that the calculations must be performed concurrently in all the children.

   Explain why the there is no need for a synchronization mechanism. Take a look at the `pipe` function complete man pages.

6. Given two arrays of integers, 1000 elements each, implement a program that creates five new processes.

   - Each child adds 200 positions (`vec1[i] + vec2[i]`), sending each sum to the parent process.
   - The parent should read the 1000 values and write them in a `result` array.

   Use 5 pipes (one for each child) and ensure that the `result` array is filled correctly. Note that the calculations must be performed concurrently in all the children.

7. Implement a program that creates 10 new processes to simulate a game called "Race to the pipe". There should be a single shared pipe between the parent and the 10 child processes.

   - The parent writes into the pipe every 2 seconds, sending a structure containing the message "Win!" and the round number (1 to 10).

- Each child process attempts to read data sent from the pipe. Upon successfully accessing the pipe, the child process prints the message and its PID, and then terminates with a value equal to the number of the round in which it managed to gain access to the pipe.
- The other child processes continue trying to read the message from the pipe.

After sending 10 messages, the parent must print the number of the round in which each of the children gained access to the pipe.

8. A retail company has sales data from the previous year recorded in an array called `sales`, and it wants to identify the records where more than 20 items were sold. Since the company has 50000 records, it wants you to optimize this task by using 10 processes, each responsible for searching through 5000 records. Consider that each record is of the following type:

```
typedef struct {
  int customer_code;
  int product_code;
  int quantity;
} record_t;
```

   - Whenever a child finds a product that meets the search criteria it should send the product's code to the parent process.
   - The parent process should record those codes in the array `larger`, printing its content whenever all child processes have been terminated.

   The values on the `sales` array should be filled with random values.

9. Implement a program that simulates a betting system. The child process begins the game with 20 euros. The game has the follwing rules:
   - The parent process generates a random number between 1 and 5;
   - Then, it writes 1 in the pipe, notifying the child that it can make another bet, or 0, if the child's credit ended;
   - The child process reads the number sent by the parent process and makes a bet or terminates, accordingly. To make a bet, the child should generate a random number between 1 and 5 and send it to the parent process;
   - The parent process waits for the child's bet or by its termination, accordingly. If the bet is correct, it increases the current balance by 10 euros. Otherwise, it deducts 5 euros.
   - The parent process sends the new credit's value to the child process. The child process prints it.

10. Implement a program that creates 5 child processes. Connect all 6 processes with a "ring" topology through pipes: the parent process will be connected to child 1, child 1 is connected to child 2, ..., and child 5 is connected to the parent process. The goal is to find the greatest random number generated by all processes:

- Each process generates a random number between 1 and 500. Then, it prints it along with its PID;
- Then, the parent process sends its generated number to child 1;
- Child 1 compares the parent's number with its own random number and sends the greater of the two to child 2;
- All other processes follow the same behavior, until child 5 sends the greater number to the parent process;
- The parent process prints the greatest random number.

11. Consider a supermarket with 5 barcode readers distributed throughout the shop. Every time a customer uses a barcode reader, the product's name and its price should be printed on the screen. Simulate this system using processes and pipes:
    - The parent process has access to the product database.
    - Each child process represents a barcode reader.
    - There is a pipe shared by all 5 child processes to request product information.
    - The parent process replies to the requesting child with the corresponding product information through a pipe that it shares only with that child.

12. Consider a factory line composed of 4 machines and a storage area:
    - Machine M1 cuts 5 pieces at a time, transfers the pieces to M2, and notifies M2;
    - Machine M2, folds 5 pieces at a time, transfers the pieces, to M3 and notifies M3;
    - Machine M3 welds 10 pieces, transfers the pieces to machine M4, and notifies M4;
    - Machine M4 packs 100 pieces at a time, transfer them to storage A1, and adds the produced parts to the inventory.

    Write a program which uses processes to represent machines and pipes to represent the flux of pieces. The program should simulate the production of 1000 pieces. All operations must be correctly printed on screen.

13. Implement a program that creates a new process and establishes a connection with it using a pipe.
    - The parent process sends the content of the file `fx.txt` to its child.
    - The child process executes the `cat` command over its standard input.

    It is expected to obtain the same result as executing `cat fx.txt` at the command line.

    Note: redirect the standard input of the child process to read data from the pipe. Use exec functions to execute the `cat` command.

14. Implement a program that creates a new process and establishes a connection with it using a pipe.

    - The child executes the `sort fx.txt` command and sends the result to its parent.
    - The parent reads the output from the child process and prints it.

    As a result, it is expected to obtain the same results as `sort fx.txt` at the command line.

    Note: redirect the standard output of the child process to write data to the pipe. Use exec functions to execute the `sort` command.

15. Implement a program that creates a new process and establishes a connection with it using a pipe.

    - The child executes the `more` command.
    - The parent sends a sequence of 100 text line to its child.

    As the `more` command needs exclusive access to the terminal, the parent process must not end before its child.

16. Implement the program `factorial`, which receives an integer from `stdin`, and prints its factorial to `stdout`.

    Implement another program that creates a new process that executes the `factorial` program. The parent should read an integer from the keyboard, send it to its child, and wait for the result. The result should then be printed on screen by the parent process.

17. Using the `dup2` system call, write a program in which the parent process prompts the user to enter a system directory to list all of its content to pipe. The child process reads from the pipe and lists only the directory names, excluding regular files.

18. Implement a program that spawns to child processes and creates the required number of pipes to solve the following problem:

    - The parent process sends the content of a text file to Child 1.
    - Child1 executes the `sort` command and sends it output to Child 2;
    - Child 2 executes the `tr` command to transform all lowercase characters into uppercase, sending its output to the parent process;
    - The parent process prints the obtained result.

19. Implement a program with the same functionality as in:

    ```
    ls -la | sort | wc -l
    ```

20. What is the goal of the following program? Justify your answer by printing the values presented by each process (assume that functions will never fail).

```
#include<string.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdio.h>
#include<stdlib.h>

int  main(){
  int i, status, error, fd[2];
  pid_t pid;
  char msg[100], read_msg[100];

  pipe(fd);
  dup2(fd[0],0);
  dup2(fd[1],1);
  close(fd[0]);
  close(fd[1]);

  for(i=1;i<5;i++){
   pipe(fd);
   pid = fork();
   if(pid > 0)
     error = dup2(fd[1],  1);
   else
     error = dup2(fd[0],  0);

   close(fd[0]);
   close(fd[1]);

   if(pid)
     break;
  }

  sprintf(msg,"This is process %d with PID %d
and its parent is ' %d\n", i, (int)getpid(), (int)getppid());

  write(1, msg, strlen(mens) + 1);
  read(0, read_msg, strlen(msg));
  wait(&status);
  fprintf(stderr, "Current process = %d, data =%s",
(int)getpid(), read_msg);
  exit(0);
}
```