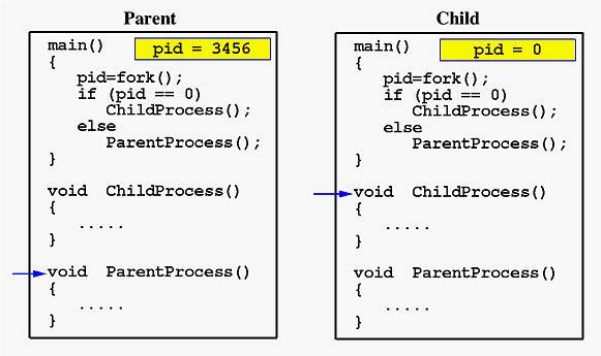
KU LEUVEN

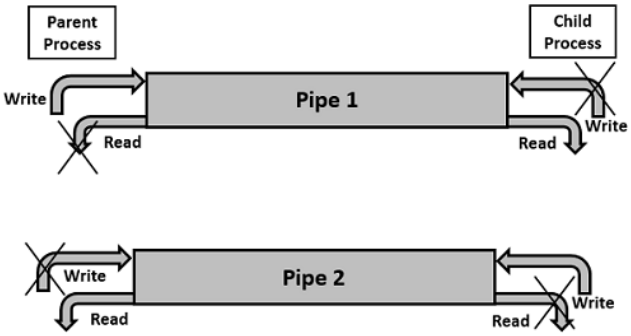
GROUP T LEUVEN CAMPUS

Operating Systems

Project labs: step 2

Milestone 2: A logging process with IPC





Bachelor Electronics/ICT

Course coordinator: Bert Lagaisse

Lab coaches:

- Ludo Bruynseels
- Toon Dehaene

Last update: juli 12, 2024

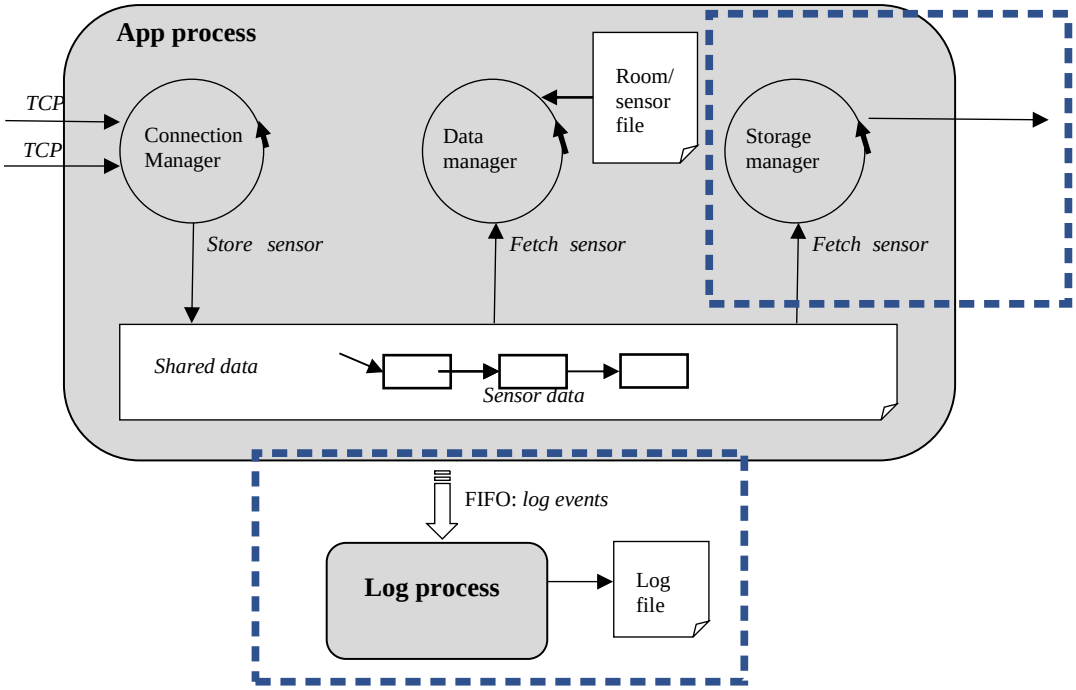
Project labs: step 2 – Inter process communication

Lab target 1: Learn how to spawn processes using fork and implement inter-process communication using pipes.

Lab target 2: Applying the concepts of forking and pipes to the logging process in the project. You will need to **submit this as milestone 2 on Toledo**.

1. Project overview

The relationship of this lab to the final assignment is indicated by the dashed blue line.



2. Warm-up Exercise 1: appending sensor data to a text file.

In the previous lab we learned how to perform file I/O in a C program. As a quick rehearsal, and as a quick warm-up for this session, you will first create the storage manager of the project. We provided the start code, including a *config.h* with the data types and structures. Using our test program (*main.c*) and our interface for the storage manager (*sensor_db.h)*, develop the *sensor_db.c* implementation to store temperature data into a text file, structured as a comma separated value file (csv). Based on the header (*sensor_db.h*) create the implementation (*sensor_db.c*) with the following operations:

- an operation to open a text file with a given name, and providing an indication if the file should be overwritten if the file already exists or if the data should be appended to the existing file,
- an operation to append a single sensor reading to the csv file,
- an operation to close the csv file.

3. Creating a child process and communicate with it

Develop a C program using ordinary pipes in which the parent process sends a string message to a child process, and the child process reverses the case of each character in the message and then prints it on the command-line. For example, if the parent process sends the message "Hi There", the child process will print "hI tHERE". You can use the pipe() command for this.

Hint 1: Use the fork() and pipe() functions as illustrated in the lectures and book.
Hint 2: You will need some interesting functions like strlen(), islower() and toupper().
Hint 3: You can assume a maximum length on the buffer that contains the text message.

Debugging hint: By default, the debugger will follow the parent process. To debug the child process:

- Set a break point at the beginning of your program (ie. the parent program, not the child program).
- Start the program in the debugger.
- Go to the debugger console (tab with the label gdb in CLion) and enter
  - **set follow-fork-mode child** and **set detach-on-fork off**.
- Continue debugging.

4. Milestone 2: creating the log process of the project

Reminder: Upload your solution for this milestone on Toledo as 1 zip!

Use the zip build target in the make file to create **milestone2.zip without subfolders**:
`make zip`

Based on the provided starting code, and with the knowledge of the 2 exercises above, we now create the logger of the project. Using our *main.c*, *logger.h* and *sensor_db.h* as provided, create a sensor_db.c and logger.c such that a new log process is created in the storage manager (sensor_db.c) as a child process, and a communication pipe between the storage manager and the newly created log process is installed. The log process receives log-events from the storage manager and writes them to a text file (the log file). A log-event contains an ASCII info message describing the type of event. A few examples of log-events are:

- A new csv file is created or an existing file has been opened.
- Data insertion succeeded.
- An error occurred when writing to the csv file.
- The csv file has been closed.

The logger should immediately write the log message to the log file, and not store it in memory, such that all information is logged when the program would crash.

- The logger process keeps running as long as your program runs, and waits for messages to log into the log file.
- For each log-event received, the log process writes an ASCII message of the format <sequence number> - <timestamp> - <log-event info message> to a new line in a log file called "*gateway.log*". Do not include the "<" and ">".

- Try to provide a clean shutdown for both processes (logger and main program) when the main program ends. Make sure you don't end up with orphan or zombie processes.
- Think about the protocol to let the processes communicate. How does the logger know when a log line ends and hence it should write it to the log file ? How does it know when the main program ends ?

We will compile and test your solution against the header *sensor_db.h* specified above, check if the logger is running as a separate process, and check if your sensor_db.csv and gateway.log contains our test data and logs.