CSE 5031 Operating Systems
2019/20 Fall Term

Project: 3 – Part 2
Topic: Process Management
Date: 11.11 - 15.11.2019

Objectives:

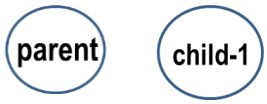
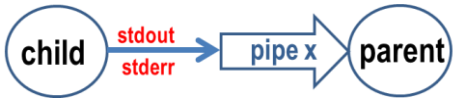
- to implement IPC with ordinary pipes
- to develop multi-process applications

References:

- **Linux System Programming 2d ed., Robert Love, O'Reilly 2013** (course web site, or http://pdf-ebooks-for-free.blogspot.com.tr/2015/01/oreilly-linux-system-programming.html
- **The GNU C Library Reference Manual** (course web site, or http://www.gnu.org/software/libc/manual/pdf/libc.pdf)

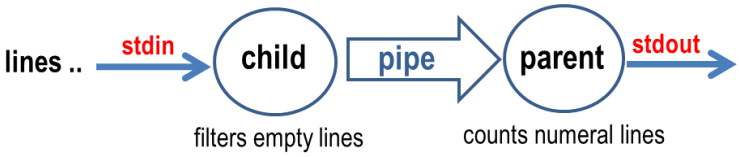
Section I. IPC Notation & Schematics used with Ordinary Pipes

Notations & schematics used to portray the **IPC framework** implemented with **ordinary pipes** & **standard I/O streams**.

✓ **Processes** are shown with circles labelled by the parenthood roles. Sibling processes are given a sequence number representing their creation sequence, i.e. Child-1, .. Child-n	
✓ An ordinary **pipe**, unidirectional channel, is depicted with a **large arrow**, the head pointing to the **read** end; the pipe may be labeled with the file descriptor array name (pfd), and its file descriptors (pfd[0] and pfd[1])	
✓ **Standard input**, **output**, **error streams** are represented with a labelled **narrow arrow**; arrow head pointing to the data flow direction	
✓ Redirection of **standard input-output-error** files to a **pipe** end is drawn with 2 connected arrows, a narrow arrow and a large arrow. The schema on the right depicts the redirection of stdout/stderr of a child to the write-end of the pipe, the parent reads from pipe's read-end	

Section II. Two Process IPC over an Ordinary Pipe

As the first step you will develop an application consisting of **2** processes: the **parent** and its **child** that share the same program and communicate using the **IPC** framework depicted here after.



Child process:

- ✓ **reads** a varying length string from **standard input unit** until the **end of file** character is entered (ctrl+d);
- ✓ **discards** empty records that contain only the new line '**n**' character;
- ✓ **writes** the string to the "**pipe**", including the new line '**n**' character that marks the end of the record;
- ✓ on **stdin end of file**, closes the "**pipe**" to signal its **parent** that the processing has ended.

Parent process:

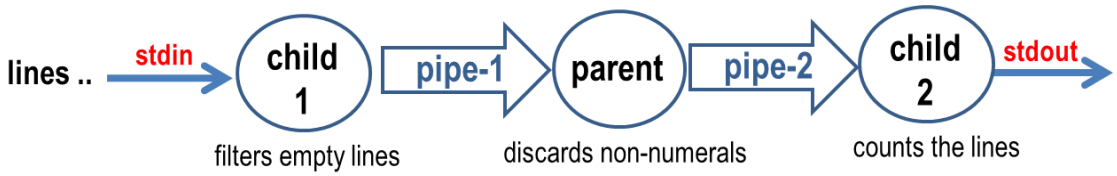
- ✓ **reads** a varying length string from "**pipe**" in a buffer of 40 bytes, until the **end of file;**
- ✓ **counts** the lines that contain only a numeral;
- ✓ on "**pipe**" **end of file**, displays the line count on **stdout**;
- ✓ **waits** for the **child** to end, then terminates.

Note that:

- **Parent** process should create the communication channel (pipe); then the child process. It is also required to wait for the termination of its **child**.
- ✓ Both processes are expected to **close** unused end of the **pipe** before starting processing; and **close used ends** before exiting.
- Use the "**fgets**" **I/O stream** function to read varying length records from **stdin** in a buffer of maximum 40 bytes (refer to the GNU C Library Reference Manual for the parameters and returned results).
- Use the constant definitions: "**#define RD 0**" and "**#define WR 1**", for expressing the input or output file descriptor for a given **pipe**, to avoid trivial indexing mistakes. You may use for instance the following notations:
  - ✓ **read** (**pfd [RD]** , buf, len); …or **write** (**pfd [WR]** buf, len);
  - ✓ **close** (**pfd [RD]** ;; …or **close** (**pfd [WR]**);

Section III. Three Process IPC over Ordinary Pipes

Once the first step is operational, extend it to develop a new application consisting of **3** processes: the **parent** and its 2 **children** that share the same program and communicate using the **IPC** framework depicted here after.



Child-1 process:

performs the functions outlined for the **Child** process in **Section I**.

Parent process:

- ✓ **reads** a varying length string from "**pipe-1**" in a buffer of maximum 40 bytes, until the **end of file;**
- ✓ **writes** the line to "**pipe-2**" using a fixed size record (e.g. 40 bytes), only if it contains a numeral;
- ✓ on "**pipe-1**" **end of file:**
  - **waits** for the **child-1** to end
  - closes the "**pipe-2**" to signal **Child-2** that the processing has ended;
  - **waits** for the **child-2** to end; then terminates.

Child-2 process:

- ✓ **reads** a fixed length record from "**pipe-2"** in a buffer of maximum 40 bytes, until the **end of file;**
- ✓ **counts** the lines;
- ✓ on "**pipe-2**" **end of file**, displays the line count on **stdout**; then terminates.

Note that, all the **processes** are expected to **close unused end** of the pipes before starting processing; and **close used ends** before exiting.

Section IV. Project-3 Part 2 Report

Do not submit a result if your program does not work as specified. Perform the following to prepare your submission

- ✓ if **phase 2** is operational name its **source code** as **phase2.c** and add a comment line consisting of your name and student-id**; if not** and if **phase 1** is operational name it as **phase1.c** add the required comments.
- ✓ Store either the **phase2.c** or the **phase1.c** filesin the "**Prj3-Part2**" folder, located at the course web site under the tab **CSE5031-X/Assignment**; where "**X**" stands for (A,B,C,D) your laboratory session group you are registered.

Warning

You are encouraged to discuss the implementation procedures and general concepts behind the projects with your fellow students. However, **plagiarism is strictly forbidden**! Submitted report should be the result of **your personal work**!

Be advised that you are **accountable** of your submission not only for this project, but also for the mid-term, and final examinations. Your project grade may be reevaluated retrospectively, had you fail to answer correctly the same or a similar examination questions that you have solved with success in your submissions.