CSE 5031 Operating Systems
2020/21 Fall Term

Project: 1 – Part 2
Topics: Standard I/O Files and Stream I/O Primitives
Date: 16.10.2020 – 24.10.2020

Objectives:

- To use Standard C Library stream I/O API.
- To experiment with standard I/O files redirection and command concatenation**.**

References:

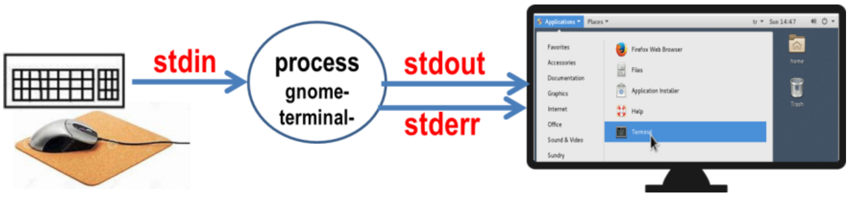
- **The GNU C Library Reference Manual** (http://www.gnu.org/software/libc/manual/pdf/libc.pdf)
- **Linux System Programming 2d ed., Robert Love, O'Reilly 2013** (http://pdf-ebooks-for-free.blogspot.com.tr/2015/01/oreilly-linux-system-programming.html)
- **Linux The Textbook. S.M. Sarwar, R.M/ Koretsky. CRC Press 2019. ISBN 978-1-138-71008-5**

Section A. Standard I/O Files and Stream I/O API

A.1 Standard I/O Files and GUI Login

CentOS 7 has been configured as a server "with **GUI**". As you logon **Linux**:

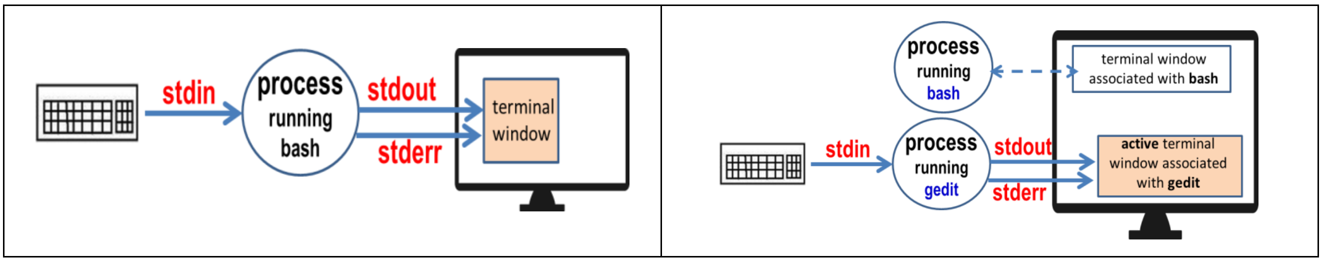
- ✓ creates a **process** to run the **GUI** program "**gnome-terminal**", defined at system generation;
- ✓ opens **3** standard files **stdin**, **stdout** and **stderr** which are pointers to **FILE type** objects in a **C** program**;** and associates them as follows:
  - ○ **stdin** (standard input file) with the keyboard;
  - ○ **stdout** (standard output file) with the GUI window;
  - ○ **stderr** (standard error file) with the GUI window.



As the user opens a **terminal window** via "Application → Favorites→ Terminal", **Linux**:

- ✓ creates a child **process** and opens a new terminal window;
- ✓ asociates **stdin**, **stdout** and **stderr** with the process that runs the "**bash**" shell defined for this account (left drawing here after).

Later on, if the user opens a second terminal and runs a program i.e. "**gedit**", or enters the command "**gedit test.c &**";. **Linux** spawns a child **process** and associates **3** standard files with the new process as shown right drawing here after.



✓ **Note that** if the user clicks on the mouse (or control keys) to select another window, or opens a new terminal, the **OS** will associate these 3 standard files accordingly.
Closing a terminal window will associate these files with the parent **process** running the GUI "**gnome-terminal**".

A.2 Stream I/O API

To run a **C** program **Linux** creates a child **process;** opens 3 standard files **stdin**, **stdout** and **stderr** associates them with the new process as described in section A.1; prepares program's parameters then invokes the **main function**.

The **stdin**, **stdout** and **stderr files** are referred as **streams;** which is an **abstraction** of a communication channel connecting the **C program** to **files, devices, processes).** For historical reasons, the type of the **C** data structure that represents a **stream** is called **FILE** rather than a "**stream**".

Standard streams are declared in the header file **stdio.h** as :**FILE * stdin**; **FILE * stdout**; **FILE * stderrn**; and they are inserted in your C program via **#include** pragma.

The **glibc I/O functions** that use **streams** are referred to as **high level I/O primitives.** They yield in portable C source codes across OSs (Linux, Windows etc.). Refer **chapter 12** of the "**GNU C Library Reference Manual**" for further details, and to learn about the syntax and semantics of the following I/O primitives:

- **fopen, fclose**
- **fgetline, fgets, fputs**
- **fread, fwrite**
- **ftell, fseek**

✓ **Chapter 3** "**Buffered I/O**" of the "**Linux System Programming"** cited under **References** contains authoritative **C** programming examples with the **streams**. You are strongly advised to refer to this programming resource, instead of wasting your valuable time in "**fishing junk**" over the Internet.

Section B. Redirecting Standard I/O Files and Command Piping

B.1 Redirecting Standard I/O Files

Most of the **shell commands** read their input from "**stdin**" and write their output to "**stdout**". These default file associations may be **redefined/overwritten**, and data flow to/from **standard I/O** files may be **directed** to other files/devices using the redirection operators: "**<**", "**>**", "**>>**".

For example the "**ls**" command, lists the directory entries on the **standard output.** Its output may be **redirected** to a file e.g. "**aggregate.txt**" using the "**>**" operator as shown hereafter. The output file is overwritten if it **exists**, or created if not.

```
ls /home/user1  >  aggregate.txt
```

The "**>>**" operator **appends** the data flow destined to the **standard output** to the end of a file if it **exists**, if not creates the file. For instance, following the execution of the previous "**ls**" command, we may append the contents of the "**abc.txt**" file to "**aggregate.txt**" as shown hereafter:

```
cat abc.txt  >>  aggregate.txt
```

Note that "**stdin**" inputs may be read from another file using the "**<**" operator; and it may be used with the input/output redirection operators. For example, "**wc -l**" command reads the input from "**stdin**" until the **end-of-file** (the "**ctrl+d**" keystroke); counts the number of lines and writes the result to "**stdout**".

Following redirection example writes the number of lines (records) stored in the "**/etc/passwd**" to the "**total.txt**" by redirecting both standard I/O units:

```
wc -l  <  /etc/passwd  > total.txt
```

B.2 Command Concatenation - Piping Standard I/O Files

The "**|**" operator **concatenates** two programs to run in sequence and redirects the "**stdout**" of the first command to the "**stdin**" of the second command. This operation is commonly referred as the "**pipelining**" or "**piping**" of **standard I/O files**. For example, to count the number of files/directories defined in the "/home/user1" directory we may use the following concatenated commands:

```
ls /home/user1  |  wc -l
```

You may use redirection operators in concatenated commands i.e. the output of the previous example will be be stored in the "**total.txt**" files by entering:

```
ls /home/user1  |  wc -l  > total.txt
```

You can also concatenate more than 2 commands. Assume that commands "**b**" and "**c**" process the records read from **stdin**, and write their output to **stdout**, and "a" writes on **stdout**. These commands may be run in **a->b->c** sequence:

```
a | b | c
```

For example, you can count the number of users that logon to a system using the "**bash**" shell with:

```
cat  /etc/passwd  |  grep "bash" |  wc -l
```

and store the result in a file:

```
cat  /etc/passwd  |  grep "bash" |  wc -l   > result.txt
```

✓ **Note that** Chapter 9 "**Redirection and Piping**" of the "**Linux The Textbook**" in the **References** explains in details these concepts. You are advised to refer to it instead of "**wandering**" over the Internet.

The reference materials you are asked to consult are integral part of this course. You are **accountable** for their use your projects as well as in your open book examinations.

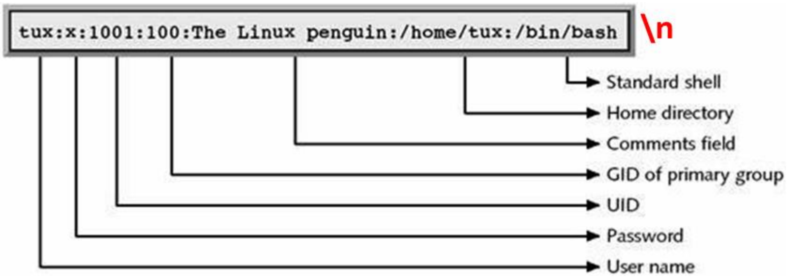
Section C. Programing with High Level Stream I/O API

C.1 Programming Example Using Stream I/O API

UNIX/Linux stores the **user accounts** in the **text** file "**/etc/passwd**". The file:

- ✓ is organized **sequentially;**
- ✓ contains **one** a **varying length record** delimited by the **new line** character "**\n**" per account.

Note that record **fields** are also of varying length and separated by a **column** character "**:**" as shown here after.



```
tux:x:1001:100:The Linux penguin:/home/tux:/bin/bash
```

i) Logon as "**sysadmin**"; open a terminal window and set your current working directory to **prj1.**

ii) Copy the "**/etc/passwd**" file in the current directory and name it "**tstpwd**".

iii) Display the "**tstpwd**"; filter out the "**sysadmin**" account, identify and analyze its fields.

The **C** program example **"prj1-sort.c**" that It **sorts** the copy of the account file in **ascending order** of the "user name" (account name) field, is stored at the course portal **CATS** in the "**Resources \ RefCprograms**" folder under the tab "**CSE5031 OS HOME**".

i) Copy the "**prj1-sort.c**" file in the current directory.

ii) Examine the program and read the comments carefully. Provided **C code** shows how to use **stream I/O API**; **pointers to arrays**, **array of pointers** etc. Review this program using the "**GNU C Library Reference Manual**" and the "**C Operators Precedence**" abstract stored at the course portal.

iii) Once you have understood its logic compile and run it to check that it performed the sort operation properly.

C.2 Modifying the Sort Program

i) Update "**prj1-sort.c**" so that it creates the sorted "**tstpwd.srt**" file in the current directory. **Use only stream I/O** primitives referred in the section A.2. Proceed with the next step only if you are successful.

ii) Modify further "**prj1-sort.c**" to include the following features:

- ✓ Replace the **static** array definitions **SortTab** [**MAXRNUM**] and **SortIdx** [**MAXRNUM+1**] by allocating them dynamically.
- ✓ Read the size of both arrays from "**stdin**"; and define in program comments how you feed the array size dynamically using **concatenated commands**, rather than entering it manually.

C.3 Developing a Query Program

Develop a new program, the "**qpwd.c**", that searches in the sorted "**tstpwd.srt**" file you have created in **section C.2** for a given account record. Use **exclusively stream I/O** primitives.

The query program should:

- ✓ display the name of the query program and the user account running it (extracted from program arguments);
- ✓ read the **account name** to query from **standard input**;
- ✓ look up the "**tstpwd.srt**" file for the account name, sequentially;
- ✓ if the search is successful display queried account`s:
  - ○ home directory and
  - ○ login shell.

Section D. Project Report

If the **programs** you have developed in **Sections C.2 and C.** are compiling without error and operate as specified herein store "**prj1-sort.c**" and "**qpwd.c**" in the "**Prj1-Part2**" folder, located at the course web site under the tab **CSE5031 – OS Section -X/Assignment**; where "**X**" stands for (1,2,3,4) your laboratory session group.

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.