## Operating Systems 2019 - 2020 Fall Project 1

**Date**: 03/11/2019

**Due**: 19/11/2019 23:59

Gains: Learning process creation and process communication, thread creation and communication, semaphore creation and usage as well as multi-process / multi-threaded application programming.

You must watch this short tutorial: Linux IPC <https://www.youtube.com/watch?v=vU2HDf5ZhO4>

For any questions, contact TA, Osman Gökhan Uyan <gokhan.uyan at agu.edu.tr>.

In this project, you are supposed to implement several programs which will search words provided by the user in a number of input files and will output the matching lines. The programs will be named psearch. They need to take the following command line parameters: an alphanumeric keyword, number of input files (n), input filenames, and an output filename. Number of input files will be greater than or equal to 1. Each input file is a text file; containing a sequence of lines. All of these lines may contain one or more (alphanumeric) words. A line of an input file matched if the given keyword is found in the line as a separate word. For instance, the keyword “name” appears in “His name is Ali.”, but does not appear in “He is named Ali”. The output file will contain the matched lines along with the name of the input file that the match is found in, and the line number of the match in the format below:

<input filename>, <matched line number>: <matched line>. The output must be sorted according to first <input filename> and then <line number>. An example execution of your program should be:

psearch name 3 input1.txt input2.txt input3.txt output.txt

**Task 1**:

**a)** Implement the program using multiple processes created by the parent main program. Call the program as psearch1a. Create children via the fork() system call. For n input files, n children must be created. Each child will process one input file and will write the result to a temporary output file. When all children end, the parent process will read the temporary output files and will write one final output file in the above format.

In psearch1a, **files will be used to pass information** from children to the parent process.

**b)** Repeat a, but this time name your program as psearch1b and use **Shared memory with shared anonymous mapping mmap()**. Also you need to use unnamed semaphores for synchronization.

**c)** Repeat a, but this time name your program as psearch1c and use an **anonymous (unnamed) pipeline pipe()**.

**Task 2**:

**a)** In this task, you will have separate processes for different jobs. There will be **a master and slave processes**.

Slave: Implement a program psearch2aslave which will take two arguments, first one is the search word, second one is the input file.

Master: Implement a program psearch2a which will take search word, input files and output file names from the user. For each input file, it should create a process psearch2slave and call it with proper arguments . (hint: exec() family of functions). So, the master will not search anything. It will make slaves do this. The master will only receive results from the slaves and saves the result into the output file.

To communicate with master, slaves should use **named shared memory (shared file mapping mmap() with a file descriptor)**

**b)** Repeat a), but the communication should be implemented using **Posix shared memory shm\_open() (without file mapping) + named semaphores (sem\_open() )**. Program names: psearch2bslave and psearch2b.

**c)** Repeat a), but the communication should be implemented using **Message passing (Message queues in Linux, mq\_open() )**. Program names: psearch2cslave and psearch2c.

**d)** Repeat a), but the communication should be implemented using **Named Pipes mkfifo()**. Program names: psearch2dslave and psearch2d.

**Task 3**: Implement the program, called psearch3, **using threads**. Instead of creating children processes, create one thread per input file. Each thread will add the matching line information to a **shared (global) linked list**. When all threads end, the main thread will process the linked lists and will write the output file in the above format. Take care of any possible synchronization problems (use semaphores). You need to use POSIX Pthreads library.

**Report**: Run the following tests and experiments with each programs. Use various input files of different sizes (e.g. >100MB). For each input and program, measure the run-time of the program. Then generate plots from your results answering following questions: With a fix number of input files, how does the run-time of your program change according to the changing input size? How does the run-time change according to the number of matches? How does the run-time change according to the number of input files? Run these experiments for each 3 program. Explain your results and come up with conclusions. Put all of your findings in to a report and submit your report together with your programs.

**Submission**:

Put all your files into a directory named your surname\_name. Then compress the directory (.zip) and upload it to Canvas. The uploaded file will look like yilmaz\_ali.zip and it will contain:

* psearch1(a to c).c: 3 C files.
* psearch2(a to d)slave.c and psearch2(a to e).c: Program 8 C file.
* psearch3.c: Program 1 C file.
* Makefile: A makefile to compile your programs. We will just type “make” and your programs will be compiled and executables psearch1a, psearch1b, …, psearch3 will be obtained.
* Report.pdf: Your report. Final report should be in PDF form.

**Style:**

Do not write a code that merely works. Also make it readable, clean, consistent and beautiful.

* Use K&R indent style. To do so:  
  In Eclipse, click Window (Or Eclipse) > Preferences > C/C++ > Code Style and choose K&R style.
* Use *snake\_case* style for naming convention
* Use meaningful identifiers

a = b \* c;  
is syntactically correct, but its purpose is not evident. Contrast this with:  
weekly\_pay = hours\_worked \* pay\_rate;  
  
Or instead of  
test123(aaa, bbb);  
Prefer:  
draw\_line(point\_from, point\_to);

**Important Note:**

1- Taking the solutions from the Internet or having the project done by somebody else or sharing is **NOT ALLOWED** and any such action is to be penalized by a disciplinary action. You have to produce your own solution and implementation. This is necessary for you to improve your knowledge and skills, which is the aim of your education.

2- Do **NOT** write your programs on Windows OS. For compatibility, use a Linux. If you only have Windows on your pc, install VirtualBox along with Ubuntu and use that environment. We will **NOT** try to make your program run, and treat it as it is. So correcting any issue that prevents your program from running is your responsibility.

### Appendix) Coverage of the project

Process ID

fork(), exec(), exit(), wait(), abort(), pipe(), mkfifo(), shm\_open()

* IPC
  + Message passing (**Message queues in Linux, mq\_open()** )
    - Direct - Indirect communication
  + **Shared memory In Linux**
    - **Shared anonymous mapping (only between Parent - child) Also need unnamed semaphores**
    - **Shared file mapping + named semaphores sem\_open()**
    - **Posix shared memory shm\_open() (without file mapping) + named semaphores sem\_open()**
* Client server systems
  + Socket (Sockets on Unix domain, AF\_UNIX)
  + Remote Procedures
  + **Pipes**
    - **Named mkfifo() and Anonymous pipe()**

Non-blocking and **blocking calls**

Buffers zero - bounded - unbounded