Rutgers ECE 434, Spring 2021 Prof. Maria Striki

Project 1: LINUX OS, Processes and Inter Process Communication

Issue Date: Wednesday 02-17-2021, Due Date: Friday March 5th 2021 10.00pm

Total Points (8 + 20 + 14 = 42 points max)

Group Number, Group member Names-Emails and Contributions:

Very useful resources for writing C code:

https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm
https://www.cprogramming.com/tutorial/c-tutorial.html
Programming Language: by Brian W. Kerninghan, Dennis M.Ritchie

Problem 1 (Introduction) (8 points)

Concatenating two files into a third one (4 pts):
You are asked to write a C program that generates an output file, the context of which is derived by concatenating the contexts of two input files. Your program will be labeled myfiles and will take two or three arguments. The first and second arguments must be the input files, while the third argument must be the output file. The default name for the output file is myfile.out

Note 1: If your program is run without the proper arguments, you must make sure that the proper message appears on screen that guides the user into how many and what sort of arguments are required. If one of the input files provided in the arguments does not exist, then the program must print on screen the corresponding error message.

Note 2 (2 pts): Execute your program using OS instruction `strace` (please research on how this works). Copy the output of `strace` that is produced from your code.

Note 3 (2 pts): Modify your initial code to support indefinite number of input files. The last argument though is always the output file.

For your implementation you are expected to use the following functions:
---- `void WriteInFile (int fd, const char *buff, int len)`: writes data to file descriptor fd.
---- `void CopyFile (int fd, const char *file_in)`: writes the contexts from file named file_in to file descriptor fd. WriteInFile is called from within CopyFile.

Below we provide one example of similar execution:

```
$ ./myfiles A1
Usage:./myfiles file_in_1 file_in_2 [file_out (default:myfile.out)]

$ ./myfiles A1 A2
A1: No such file or directory

$ echo 'Data for file_in_1 This is OS 434 Sp21,' > A1
$ echo 'data for file_in_2 but also OS 579 Sp21!' > A2
$ ./myfiles A1 A2
$ cat myfile.out
Data for file_in_1 This is OS 434 Sp21,
data for file_in_2 but also OS 579 Sp21!
```

```
$ ./myfiles A1 A2 A3
$ cat A3
Data for file_in_1 This is OS 434 Sp21,
data for file_in_2 but also OS 579 Sp21!
```

Solution:

Problem 2: Introduction to multi-process environments and Inter-process Communication (20 points)

Problem Introduction and Description: Generate a text file and populate it with L (>=5,000) positive integers. Identify the MAXIMUM, the AVERAGE, and a number of "H" GIVEN HIDDEN KEY INTEGERs **out of the 20 hidden keys randomly placed in the file**. Here H in [2,4] different locations in the file of L integers. You may use any form of inter-process communication (IPC) to partition the file/array into chunks and distribute work to more than one processes (if there are multiple ones) (e.g., pipes, signals, or shared memory).

You must "hide" your secret keys in a uniform way into your list/file. The hidden keys will be **20 negative integers** (assume all are – 10) equally spaced within your file. But then, after creation of the file, you "forget" about where the hidden keys are located. The only information that you know is that you are looking for "H" such elements of value -10.The arguments L and H are provided at run time and are handled by `main (int argc, char *argv[]).` If you want, you can also ask this information from the keyboard and have your program read/scanf the information entered from the user.
(For those that are not familiar how to use argc-argv[] please read the related section on tutorialspoint site provided above, it is very helpful).

Objective: Use multiple processes "PN" (where PN: Process Number argument), use different process layout strategies (DFS, tree: see next page), and IPC to speed up your searching on the Maximum element, the Average element, and the "H" hidden elements. You can obtain argument "PN" from the arguments at run time (or read it from the user's input later) and must then impose that your program generates no more than PN processes.

Remark: Record the time it takes for each of the programs to run and comment on your observations. Try it on lists of size L: 5k, 10k,100k, 1M.
Hint: You should first load the file into an array then start working on the data.

Input Format:
Input will be in a text file. Each integer will be separated by a newline character (\n).
Example:

100
20
35

Output Format: You should print out the results in a text file. Every process that is created should print out their own process id and their parent's process id. Then once you have computed the final max, avg, and hidden keys, you will print those out. Please follow this format:

Hi I'm process 2 and my parent is 1.
Hi I'm process 3 and my parent is 2.
Hi I'm process 4 and my parent is 2.
Max=50, Avg =36 (fictitious numbers).

Hi I am process 3 and I found the hidden key in position A[65].
Hi I am process 7 and I found the hidden key in position A[123].
Hi I am process 12 and I found the hidden key in position A[204].
Hi I am process …. (all the above are fictitious numbers).

Detailed Instructions on Processes Layout (15 points):

1) Write a program to solve the problem using only one process.
2) Write a program to solve the problem using multiple processes where each process spawns at most one process. (Like DFS). The maximum number of processes spawned should be NP.
3) Conversely, write a program to solve this problem using multiple processes where the first process spawns a fixed number of processes, say X, and they (the children) spawn their own X multiple processes each, and so on and so forth. The maximum number of processes spawned should be NP.
4) Ensure that in all cases you are using the appropriate system calls for InterProcess Communication (IPC) and for making the processes communicate and terminate gracefully, and without zombies.
5) Your ultimate goal should be to produce such a hierarchy of processes (process tree) that produces the final result faster. Can you fine-tune the size of variable NP to do that? Or are the preceding configurations (i.e., single process or DFS (chain) of processes) the clear winners? Consider lists of increasing sizes. There is a trade-off w.r.t. to the number of processes but there is also a trade-off w.r.t. the number of integers in the file (and increasing file sizes). What are the trade-offs you observe here? Can you articulate and explain?
6) Can you create an arbitrary number of processes or are there any limitations? If you find limitations in your version of Linux OS, please show (e.g. via a printscreen or a file) what are your exact limitations in increasing the number of processes you will be generating. Why is this so?
7) You should also identify and output which processes (which pids) are the ones that contain the first H hidden keys identified.
8) What is the maximum number of processes you can create before your system crashes?

Variation of the Program – only identify the 3 hidden elements (5 points):

Now your task is to identify only the H hidden elements, not to compute the maximum and or average element. Identify and implement the changes you can make to your program that **improve its speed and ensure the following operation**: as soon as the H hidden keys are identified, **your program terminates gracefully**, **without creating zombies or orphans**. You must use the proper system calls.

Clarification Notes – EXTREMELY IMPORTANT (please read): I will grade Problem 2 leniently. You will get the freedom, depending on your familiarity with C programming to handle Question 3 of Problem 2 in various ways.

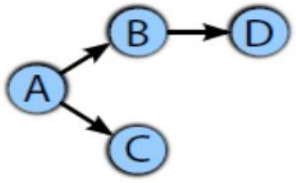
1) You may decide on a predefined number X and a predefined depth of using X so that the tree that you build is predefined and you do not have to use any complicated for loops or recursion or counters. **Penalty**: **2 points**.
2) You may use more elaborate data structures to produce your tree of processes, and you have the option to ask the user to provide X. At the end, you must produce a couple of varying tree structures and compare with each other to address the questions asked.

Regarding IPC: You may use pipes or information passed through the exit system calls, whatever each group gets most comfortable with. Do not use sigqueue at this point: I will be introducing it later because it is more complicated.

Solution:

Problem 3: (**Building a given Process Tree**) **(14 points)**

In this exercise you are asked to write a program which generates the following process tree (Scheme 1).



Scheme 1: A given process tree.

The processes that you generate must remain active for some considerable window of time in order for the user to be able to observe the tree. The leaf process executes a call to: sleep(). The internal process awaits of the termination of all its children processes. Every process prints a corresponding message every time it transitions to another phase (for example: start, loop to wait for children termination, allowing its own termination), so that the validation of the correct program operation is feasible. In order to separate the processes, please make sure that every process **terminates with a different return code**. In this example, one scenario can be:

A = 2, B = 4, C = 6, D = 10.

At this point, you may find helpful a number of auxiliary functions for process handling, such as those that:

1) have to do with identifying the circumstances under which a child process terminated (discussed in your lectures and lecture slides),
2) display the process tree starting from the root process (included in the appendix),
3) produce different ways of recursively traversing a tree once the whole process tree is generated, etc.

Additional Questions to answer:

1. What happens if root process A is terminated prematurely, by issuing: kill -KILL <pid>?
2. What happens if you display the process tree with root getpid() instead of pid() in main()? Which other processes appear in the tree and why?
3. What is the maximum random tree you can generate with your program? Why?

Clarification Notes – EXTREMELY IMPORTANT (please read): You will get the freedom, depending on your familiarity with C programming to handle Problem 3 in various ways.

1) You may build the tree illustrated in the picture in a simple way, without using the auxiliary data structures. **Penalty**: **2 points.**

2) You may first build a random tree using the auxiliary structures and then adjust it to the tree of the picture.

Solution:

What to turn in for all 3 problems:

- C files for each problem
- A makefile or a readme file in order to run your programs.
- Input text file (your test case)
- Output text file (for your test case)
- **Report**: Explain design decisions (fewer vs. more processes, process structure, etc.). Elaborate on what you have learned from each problem. Answer the question(s) below each part/subproblem. A neat and detailed report, along with your C file deliverables, corresponds to a substantial portion of your grade.

Logistics:

- For Project 1 please work in groups of 4-5-6 students.
- You are expected to work on this project using LINUX OS
- For those that do not have access to LINUX in their laptop, you may use one of the solution that I have already posted online regarding how to get access to a LINUX platform.
- Make ONE submission per group. In this submission provide a table of contribution for each member that worked on this project.
- Do not collaborate with other groups. Groups that have copied from each other will BOTH get zero points for this project (as a warning) no matter which copied from another, and will also incur more substantial consequences.

APPENDIX

Useful Links:

https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm

https://www.cprogramming.com/tutorial/c-tutorial.html

Programming Language: by Brian W. Kerninghan, Dennis M.Ritchie

https://www.gnu.org/software/libc/manual/html_node/Pipes-and-FIFOs.html#Pipes-and-FIFOs

https://www.gnu.org/software/libc/manual/html_node/Creating-a-Process.html#Creating-a-Process

https://www.gnu.org/software/libc/manual/html_node/Process-Completion.html#Process-Completion

https://en.wikipedia.org/wiki/Depth-first_search

Useful Auxiliary Functions and Definitions

explain_wait_status ()

```
void explain_wait_status(pid_t pid, int status)
{
  if (WIFEXITED(status))
    fprintf(stderr, "Child with PID = %ld terminated normally, exit status = %d\n",
          (long)pid, WEXITSTATUS(status));
  else if (WIFSIGNALED(status))
    fprintf(stderr, "Child with PID = %ld was terminated by a signal, signo = %d\n",
          (long)pid, WTERMSIG(status));
  else if (WIFSTOPPED(status))
    fprintf(stderr, "Child with PID = %ld has been stopped by a signal, signo = %d\n",
          (long)pid, WSTOPSIG(status));
  else {
    fprintf(stderr, "%s: Internal error: Unhandled case, PID = %ld, status = %d\n",
          __func__, (long)pid, status);
    exit(1);
  }
  fflush(stderr);
}
```

Example:

```
pid = wait(&status);
explain_wait_status(pid, status);
if (WIFEXITED(status) || WIFSIGNALED(status))
        --processes_alive;
```

Example of handling SIGCHLD

```
void sigchld_handler(int signum)
{
        pid_t p;
        int status;

        /*
         * Something has happened to one of the children.
         * We use waitpid() with the WUNTRACED flag, instead of wait(), because
         * SIGCHLD may have been received for a stopped, not dead child.
         *
         * A single SIGCHLD may be received if many processes die at the same time.
         * We use waitpid() with the WNOHANG flag in a loop, to make sure all
         * children are taken care of before leaving the handler.
         */

        do {
                p = waitpid(-1, &status, WUNTRACED | WNOHANG);
                if (p < 0) {
                        perror("waitpid");
                        exit(1);
                }
                explain_wait_status(p, status);

                if (WIFEXITED(status) || WIFSIGNALED(status))
                        /* A child has died */
                if (WIFSTOPPED(status))
                        /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
         } while (p > 0);
}
```

Auxiliary Functions and Operations on Trees and Tree Nodes

```
struct tree_node {
        unsigned          nr_children;
        char              name[NODE_NAME_SIZE];
        struct tree_node  *children;
};
```

```
static void
__print_tree(struct tree_node *root, int level)
{
        int i;
        for (i=0; i<level; i++)
                printf("\t");
        printf("%s\n", root->name);

        for (i=0; i < root->nr_children; i++){
                __print_tree(root->children + i, level + 1);
        }
}

void
print_tree(struct tree_node *root)
{
        __print_tree(root, 0);
}
```

How to write a MakeFile (Example):

```
$ cat Makefile
# a simple Makefile

CC = gcc
CFLAGS = -Wall -O2

all: fork-example

fork-example: fork-example.o proc-common.o
 <Tab> $(CC) -o fork-example fork-example.o proc-common.o

proc-common.o: proc-common.c proc-common.h
 <Tab> $(CC) $(CFLAGS) -o proc-common.o -c proc-common.c

fork-example.o: fork-example.c proc-common.h
 <Tab> $(CC) $(CFLAGS) -o fork-example.o -c fork-example.c

clean:
 <Tab> rm -f fork-example proc-common.o fork-example.o
```

```
$ make
gcc -Wall -O2 -o fork-example.o -c fork-example.c
gcc -Wall -O2 -o proc-common.o -c proc-common.c
gcc -o fork-example fork-example.o proc-common.o
```