CMP SCI 4760 **Operating Systems**
Project # 1 Due Date: Feb 10th, 2026

Important: Please do all projects on `opsys`

Skeleton multiple processes

Purpose

This is your warm up project building on your knowledge from linux courses like our CMPSCI 2750 or other related courses. The goal of this project is to become familiar with the environment in `opsys` while creating a setup for future projects. Additionally, you should understand the different steps of compilation and linking, and creating your executable.

You may use generative AI on this project, but you must follow the guidelines as stated at the end of the document where I lay out the README.

Task

In this project you will be writing a makefile to compile two source files into two separate executables. One of the executables will be called oss, generated at least partially from the source file oss.c. The other executable will be called user, generated at least partially from the source file called user.c.

The user executable is never executed directly. Instead, oss will be launching that executable at various times.

Note you MUST have a makefile that compiles and creates these executables at the same time. This will require the use of the ”all” command in your makefile to compile multiple executables from the same makefile.

We will first discuss what the user executable does, but keep in mind that it will never be called by itself directly (though you can do so to test).

User process (the children)

The user takes in one command line argument. For example, if you were running it directly you would call it like:

```
./user 5
```

As it is being called with the number 5, it would do 5 iterations over a loop.

So what does it do in that loop? Each iteration it will output its PID, its parents PID, and what iteration of the loop it is in. For example, suppose its PID is 6577, its parents PID is 6576 and it is the 3rd iteration of the loop, it would output:

```
USER PID:6577 PPID:6576 Iteration:3 before sleeping
```

After doing this output, it should do sleep(1), to sleep for one second, and then output:

```
USER PID:6577 PPID:6576 Iteration:3 after sleeping
```

You should test this by itself, but it will not be called by itself normally.

oss (the parent)

The task of oss is to launch a certain number of user processes with particular parameters. These numbers are determined by its own command line arguments.

Your solution will be invoked using the following command:

```
oss [-h] [-n proc] [-s simul] [-t iter]
```

The proc parameter stands for number of total children to launch, iter is the number to pass to the user process and the simul parameter indicates how many children to allow to run simultaneously.

For example, if I wanted to launch oss such that it would launch 5 user processes, never allow more than 3 to be running at the same time, and then have each of the users do 7 iterations, it would be called with:

```
oss -n 5 -s 3 -t 7
```

If called with the -h parameter, it should simply output a help message (indicating how it is supposed to be run) and then terminating.

So now that I know what parameters it should run with, what should it do? oss when launched should go into a loop and start doing a fork() and then an exec() call to launch user processes. However, it should only do this up to simul number of times. In our above example, we would launch no more than 3 initially. The oss would then wait() until one of the children had finished before launching another.

After oss has finished launching up to n processes, it should continue running until all children that it launched have terminated. It can do this by wait()ing a number of times based on how many children are still in the system.

Oss should output a message whenever it launches a new process. At the end, it should also output a summary of how many children it launched.

Rough pseudocode for what oss should be doing is below, though other logic can work also:

```
// Given n is max process to launch, s is simul limit
// c is current processes running, starting at 0
// total is total processes that have launched, starting at 0
while ( c < s ) {
    Launch new child process
    oss outputs message of new child launched
    c++;
    total++;
}

while ( total < n) {
    wait();
    Launch new child process
    total++;
}

wait();

output summary report of how many total processes finished
```

Implementation details

It is required for this project that you use version control (git), a `Makefile`, and a `README`.

Your makefile should also compile BOTH executables every time. This requires the use of the all prefix.

Suggested implementation steps

- Set up your git repository, if you have not already done so. You should periodically check your code into the git repository (once a day, or whenever you make and test substantial changes). [Day 1]
- Create your Makefile and barebones skeleton of oss and user. Make sure that if you change either one, the makefile will compile both. [Day 2]
- Write code to parse options and receive the command parameters. Study `getopt(3)`, if you do not know how to do it. The man page also has an example to guide you. [Day 3]
- Implement the user taking in an option and doing its loop and output. [Day 4]
- Implement oss to fork() and then exec() off one user to do its task and wait() until it is done. [Day 5]
- Get oss to fork off users up until the -n parameter [Day 6-7]
- Implement the simultaneous restriction [Day 8-9]
- Testing and make sure your `README` file indicates how to run your project. Give a one-line example that would let me know how to run it. DO NOT SIMPLY COPY/PASTE THIS DOCUMENT INTO THE README [Day 10+]

Criteria for success

Please follow the guidelines. Start small, implement one functionality, test. Do not wait until the last minute and contact me if you are having issues.

Grading

- *Overall submission: 10 pts*. Program compiles and upon reading, seems to be able to solve the assigned problem.
- *Code readability: 10 pts*. The code must be readable, with appropriate comments. Author and date should be identified.
- *Command line parsing: 10 pts*. Program is able to parse the command line appropriately, assigning defaults as needed; issues help if needed.
- *Makefile: 10 pts*. Makefile works, compiles both files even if both are changed or one is changed. Also ensure your Makefile can do a clean.
- *README: 10 pts*. Must address any special things you did, or if you missed anything.
- *Conformance to specifications: 50 pts*. Does your application do the task.

README

It is required that you submit a README file. This file should start with the following:

```
Name: Yourname
Date: When project started
Environment: What environment you used (linux, vi)
How to compile the project:
   Type 'make'
Example of how to run the project:
   ./oss -n 3 -s 2
```

In addition, if you use generative AI, you must include a section describing your use of generative AI in detail. This should include a bare minimum of what generative AI was used and some list of prompts that you used. I also want at least a few sentence description of in general how useful you found it.

If you are not using generative AI, this is fine, but you must include a short explanation why you are not. This does not have to be in depth, but I want to see your rationale for my analysis.

For example, at a bare minimum something like:

```
Generative AI used : chatgpt
Prompts:
   Please give me a C++ command line program to play tic-tac-toe between two people
   Change the previous code to add the capability to play multiple games
   Add the capability to keep track of the previous record of players.
   Given the previous code, please add a command line option to allow one player to cheat
   etc

Summary: The initial game generated worked well, but it had one bug that I had to fix. It dic
```

Submission

Handin an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.1 where *username* is your login name on opsys. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
chmod 700 username.1

cp -p -r username.1 /home/hauschildm/cs4760/assignment1
```

If you have to resubmit, add a .2 to the end of your directory name and copy that over.

Do not forget `Makefile` your versioning files (`.git` subdircetory), and `README` for the assignment. If you do not use version control, you will lose 10 points. I want to see the log of how the program files are modified. Therefore, you should use some logging mechanism, such as git, and let me know about it in your `README`. You must check in the files at least once a day while you are working on them. I do not like to see any extensions on `Makefile` and `README` files.

Before the final submission, perform a `make clean` and keep the latest source checked out in your directory.

You do not have to hand in a hard copy of the project. Assignment is due by 11:59pm on the due date.