Ontario Tech University

SOFE 3950U / CSCI 3020I Operating Systems

LAB # 2

Instructor: Nahid Hasan Khan

nahid.hasankhan@uoit.ca

Objectives

- Learn the fundamentals of signals and data structures in C
- Gain experience writing multiprocessor code and data structures
- Create a process scheduler in C

Important Notes

- Work in groups of **four** students
- All reports must be submitted as a PDF on blackboard, if source code is included submit everything as an archive (e.g. zip, tar.gz)
- Save the submission as <lab_number>_<first student's id> (e.g. lab2_100123456.pdf)
- If you cannot submit the document on blackboard then please contact the TA with your submission at nahid.hasankhan@uoit.net

Lab Details

Notice

It is recommended for this lab activity and others that you save/bookmark the following resources as they are very useful for C programming.

- https://en.cppreference.com/w/c
- https://www.cplusplus.com/reference/clibrary/
- https://gribblelab.org/CBootCamp/

The following resources are helpful as you will need to use signals and data structures to complete the task.

- https://www.gnu.org/software/libc/manual/html_node/Standard-Signals.html
- https://www.gnu.org/software/libc/manual/html_node/Signaling-Another-Process.html
- http://www.gnu.org/software/libc/manual/html_node/Process-Completion.html
- http://www.gnu.org/software/libc/manual/html_node/Signaling-Another-Process.html
- http://www.learnc.org/en/Linked_lists

For the lab activity you must have the program **process** in the same location as your **hostd** executable, compile the included source file **sigtrap.c** using the provided **Makefile** , if you are having issues replace **CC = clang** with **CC = gcc** in the makefile.
make process

Lab Activity

1. For the purpose of this lab, either create a new repository on GitHub for this lab, or create a folder in your existing GitHub repository that you created in the previous lab.

2. Download the source code and makefile and use git to add the contents and push it to your GitHub repository.

3. Before writing any of the source code for your project, review the entire host dispatcher shell project description and ensure that you understand what is required.

4. Next, work in groups to begin writing the design document for the project, describing the memory allocation algorithms, data structures, functions, and overall structure of your program (project requirements **1. a - d**)

5. Finally, work on implementing the host dispatcher shell project.

Clarifications

1. The lab consists of completing and submitting the **entire** host dispatcher shell project, ensure that you meet all of the project requirements.

2. You do **NOT** need to modify the **sigtrap.c** source code, simply compile it to make the process binary, which is executed by your dispatcher shell.

3. There are **four** process queues: **real time queue,** and **priority 1 3 queues,** for each queue you will need to create a linked list, with the **push()** and **pop()** operations to add and remove items from the queue.

4. Queue clarifications:

   ○ For processes in the **real time** queue, they are executed immediately until their runtime is completed. They are processed on a first-come-first-served basis.

   ○ For **priority 1 2** queues, after a process has been run for **1 second** it is removed from the queue and added to the next lower priority queue.

   ○ For the **priority 3** queue once processes are added to it, they are run for **1 second** then added back to the priority 3 queue.

   ○ Once a process has been executed for its runtime it is not added back to the queue, since the process has completed its execution.

5. For the resource constraints, your program has the following resources: **2** printers, **1** scanner, **1** modem, **2** CD drives, and **1024** MB of memory.

- ○ The printer, scanner, modem, and CD drives can be stored in a single structure called **resources**, when you make the structure initialize as **res_avail** and initialize the members to those values.

- ○ For the 1024 MB of memory you can make it an array called **memory** (initialized to 0) that is a member of the **resources** structure, you can use a **#define MEMORY 1024**, to define the amount of memory available

- ○ Your processes can use a similar structure to define the resources they require.

6. For allocation the memory for processes, it is recommended you use **First Fit**, whereby you allocate the first contiguous section of memory that is free in your **memory** array.

- ○ A value of **0** in the array indicates the memory is free

- ○ Once you find a free section, iterate through the following **N** MB of values, if they are all **0**, then set them all as 1 to indicate the memory is being used

- ○ Your **function** should return the **starting index** that the memory was allocated at, so that you can free it after the process is terminated

- ○ **ALWAYS** leave the last 64 values in the array free for real time processes to use.

7. Ensure that after each process is terminated you **free up** the resources it used, including the memory that was allocated for that process so that these resources are available for the next process.

8. Each time your run the **process** binary, use the **fork()** and **exec()** functions, the **PID** returned by **fork()** can be used to send signals to **process** using the **kill()** function.

9. The only signals you need to use are: **SIGINT, SIGTSTP, SIGCONT**

10.After killing a process you **must** use the **waitpid()** function to ensure that another process is not started until that process has terminated.

11. The makefile provided and source code templates include **queue.h** and **queue.c** source files, use these to implement your linkedlist (queue) data structures and all associated functions.

Deliverables

Notice

Please complete the deliverables and include your design document, all source files, and the makefile.

1. All sources files, all of the **Project Requirements** described in the host dispatcher shell project document must be met. Your lab report must include a design document as described in **Project Requirements 1. a – d**

2. A makefile is included so that the source code can be compiled, if your makefile does not work then marks will be deducted.

3. Your host dispatcher shell must be able to parse a comma-separated file called **dispatchlist** containing the processes and resources required, as described in the included host dispatcher shell project document. Your submission will be evaluated by running it with a **dispatchlist** file containing a comma-separated delimited list such as the example below.

```
0, 0, 1, 64, 0, 0, 0, 0
0, 1, 3, 128, 1, 0, 0, 1
1, 3, 6, 128, 1, 0, 1, 2
```

4. In order to ensure that your submission works correctly, test and evaluate it using the **dispatchlist** file provided.