SOFE 3950U / CSCI 3020U: Operating Systems

TUTORIAL #7: Signals and Data Structures

Objectives

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

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 file as <tutorial_number>_<first student's id>.pdf (e.g. tutorial7_100123456.pdf)

If you cannot submit the document on Blackboard then please contact the TA (Jonathan Gillett) with your submission on slack at **http://sofe3950u.slack.com** or send him an email via **jonathan.gillett@uoit.net**.

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.

- http://en.cppreference.com/w/c
- http://www.cplusplus.com/reference/clibrary/
- http://users.ece.utexas.edu/~adnan/c-refcard.pdf
- http://gribblelab.org/CBootcamp

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

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

Conceptual Questions

1. What are signals, what is their purpose?

2. Explain the following signals: **SIGINT, SIGTSTP, SIGCONT** how can they be used to suspend, resume, and terminate a program?

3. Explain the following functions: **kill(), waitpid()** how can they be used to terminate a process and wait until it has ended before continuing?

4. Explain what a **linked-list** (queue) is, what does **FIFO** mean? What are the common operations that a linked-list must have?

5. Explain the structure of a linked-list as implemented in C, explain how would you implement the operations to add and remove values from the queue?

Application Questions

All of your programs for this activity can be completed using the template provided, where you fill in the remaining content. A makefile is not necessary, to compile your programs use the following command in the terminal. **If you do not have clang then replace clang with gcc**, **if you are still having issues please use -std=gnu99 instead of c99.**

```
clang -Wall -Wextra -std=c99 <program name>.c -o <program name>
```

Example:

```
clang -Wall -Wextra -std=c99 question1.c -o question1
```

You can then execute and test your program by running it with the following command.

```
./<program name>
```

Example:

```
./question1
```

Template

```
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(void)
{ ... }
```

1. Create a program that does the following.
    - Create a structure called **proc** that contains the following
        - **name**, character array of 256 length
        - **priority,** integer for the process priority
        - **pid**, integer for the process id
        - **runtime,** integer for the running time in seconds
    - Create a linked list structure called **queue**, which contains the following:
        - **process,** an instance of your **proc** structure
        - **next,** a pointer to the next linked list structure initialized to **NULL**
    - Create a function: **push(proc process)**, which is used to add each process to the linked list (**queue**).
    - Your program then reads the contents of a file called **processes.txt (10 LINES)**, which contains a **comma separated** list of the name, priority, pid, and runtime.
    - For each entry read from **processes.txt** create an instance of the **proc** struct and add it to the linked list using the **push()** function.
    - After all processes have been added to the linked list, **iterate** through it and print the **name, priority, pid, runtime** of each process.
    - See the following for more information on linked lists: http://www.learn-c.org/en/Linked_lists

2. Extending from the previous problem, create a program that does the following.
    - Implements the remaining operations needed for a FIFO queue: **pop, delete_name, delete_pid.**
    - The function **pop()** removes the next item from the queue and returns the process (**proc** struct instance).
    - The function **delete_name(char *name)** deletes a process from the queue given the name and returns the process, NULL returned if not found.
    - The function **delete_pid(int pid)** delete a process from the queue given it's process id (pid) and returns the process, NULL returned if not found.
    - Your program should then read in the **processes.txt** file (similar to in question 1) and then perform the following operations:
        - **Delete** the process named **emacs** (an inferior editor to vim)
        - **Delete** the process with the pid **12235**
        - Iterate through the remaining processes in the queue and use **pop()** to remove each item from the queue and print the **name, priority, pid, runtime** of each process.

Notice

For the remaining questions, you must have the program **process** in the same location as your source code, compile the included source file **sigtrap.c** as process using the following commands. You can use **clang** or **gcc**, for most of the remaining problems you may need to use **-std=gnu99** in order for the code to compile properly.

```
clang -Wall -Wextra -std=gnu99 sigtrap.c -o process
```

3. Create a program that does the following.
    - Forks and executes **process** using the **fork** and **exec**
    - Sleeps for 5 seconds
    - Sends the signal **SIGINT** to the process using **kill**
    - You should see the process running in the terminal printing to the screen it's PID and other information with colour for five seconds and then the output will stop once it's terminated.
    - See the following for an example of using fork and exec: http://www.gnu.org/software/libc/manual/html_node/Process-Creation-Example.html#Process-Creation-Example

4. Create a program that does the following.
    - Forks and executes **process** using the **fork** and **exec**
    - Sleeps for 5 seconds
    - Sends the signal **SIGTSTP** to the process to suspend it
    - Sleeps for 10 seconds
    - Sends the signal **SIGCONT** to the process resuming it
    - Uses the **waitpid** function to wait until the process has terminated (about 5 seconds) before the main process exits.
    - You should see the process running in the terminal printing to the screen it's PID and other information with colour for five seconds and then the output will stop for 10 seconds once it's suspended and resume again until the program terminates.

5. Create a program that does the following.
    - Using the linked list and queue implemented in problems 1 and 2, create an instance of your linked list called **queue.**
    - Read in the processes from the file **processes_q5.txt**, this time **the file does not** contain the **pid**, you must initialized the pid to 0, it's set when you execute the processes.
    - When reading the file **processes_q5.txt** add each process to the queue.

- Iterate through all of the processes in the queue and execute those with a **priority of zero first**, use **delete_name()** to remove the process struct and then run the **process** binary using fork and exec.
    - Set the **pid** member in the process struct to the process id returned from **exec().**
    - Run the process for the specified **runtime** and then send it the signal **SIGINT** to terminate it.
    - Ensure that you use the **waitpid** function to wait until the process has terminated.
    - After the **process** is terminated, print the **name, priority, pid,** and **runtime** of the process.
- Then iterate through all of the remaining processes in the queue, **pop()** each item from the queue and execute **process** using fork and exec.
    - Set the **pid** member in the process struct to the process id returned from **exec().**
    - Run the process for the specified **runtime** and then send it the signal **SIGINT** to terminate it.
    - Ensure that you use the **waitpid** function to wait until the process has terminated.
    - After the **process** is terminated, print the **name, priority, pid,** and **runtime** of the process.
- Once all of the processes have been executed the main program terminates.