SOFE 3950U / CSCI 3020U: Operating Systems

TUTORIAL #7: Signals and Data Structures

Rishab Singh – 100787473
CRN: 74027

Conceptual Questions

1. What are signals, what is their purpose?

Signals in operating systems are software interrupts that notify a process of an event. They serve as a means of communication between the operating system and processes, as well as between processes themselves. Signals can indicate various events such as errors, hardware exceptions, or user-defined events. Their purpose includes facilitating process management, handling exceptional conditions, and enabling inter-process communication.

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

SIGINT (Signal Interrupt): This signal is typically generated by pressing Ctrl+C on the keyboard. It's commonly used to request termination of a program in a graceful manner, allowing it to clean up resources before exiting.

SIGTSTP (Signal Stop): Generated by pressing Ctrl+Z, it suspends the execution of a program, effectively putting it in the background. The program is stopped but remains in memory, allowing it to be resumed later.

SIGCONT (Signal Continue): This signal is used to resume the execution of a previously stopped program, typically after receiving a SIGTSTP signal. It restores the execution of a stopped process.

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

kill(pid, sig): This function sends a signal specified by sig to the process identified by pid. It can be used to terminate a process by sending a SIGKILL signal (signal 9), which forces termination without allowing the process to clean up.

waitpid(pid, status, options): This function suspends the calling process until the process with the ID pid terminates, or until a specific process state changes. It provides a way for a parent process to wait for its child process to finish executing before proceeding. The status parameter is used to store information about the terminated process, and options allow for additional behavior control.

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

A linked-list is a data structure consisting of a sequence of elements, where each element points to the next one in the sequence, forming a chain-like structure. A queue is a type of linked-list where elements are added at one end (rear) and removed from the other end (front). FIFO stands for "First-In, First-Out," which means that the element that is added first is the one removed first.

Common operations for a linked-list (queue) include:

- Insertion (enqueue): Adding elements to the rear of the queue.
- Deletion (dequeue): Removing elements from the front of the queue.
- Traversal: Iterating through the elements of the queue.
- Search: Finding a specific element within the queue.
- Empty check: Determining if the queue is empty.
- Size determination: Counting the number of elements in the queue.

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?

In a linked-list, each node contains two parts: the data field, which holds the value of the element, and a pointer to the next node.

To implement a queue using a linked-list, you would have a pointer pointing to the front (head) of the queue and another pointer pointing to the rear (tail) of the queue.

- **Enqueue (Add):** To add an element to the rear of the queue, you would create a new node, assign its data, and adjust pointers to maintain the integrity of the queue.
- **Dequeue (Remove):** To remove an element from the front of the queue, you would adjust pointers to skip the first node and deallocate memory if needed. These operations can be performed efficiently with a linked-list structure, providing constant-time insertion and deletion at both ends of 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

```
rishab@Rishabs-Laptop tut7 % ./question1
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
rishab@Rishabs-Laptop tut7 %
```

-

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.

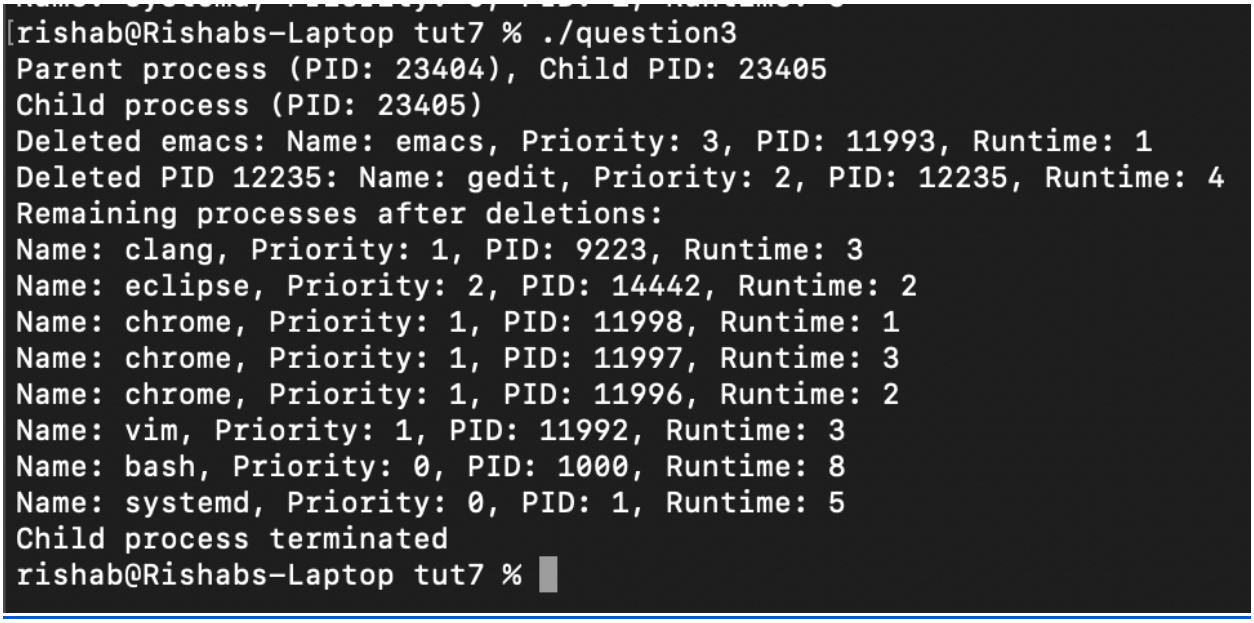
```
rishab@Rishabs-Laptop tut7 % ./question2
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
rishab@Rishabs-Laptop tut7 %
```

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.

```
rishab@Rishabs-Laptop tut7 % ./question4
Parent process (PID: 23474), Child PID: 23475
Child process (PID: 23475)
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
Sending SIGTSTP to suspend the child process.
Sending SIGCONT to resume the child process.
Child process terminated
rishab@Rishabs-Laptop tut7 %
```

-

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.

```
rishab@Rishabs-Laptop tut7 % ./question5
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
Process: bash, Priority: 0, PID: 23601, Runtime: 8
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
Process: systemd, Priority: 0, PID: 23622, Runtime: 5
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
Process: clang, Priority: 1, PID: 23635, Runtime: 3
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
Remaining processes after deletions:
Name: clang, Priority: 1, PID: 9223, Runtime: 3
Name: eclipse, Priority: 2, PID: 14442, Runtime: 2
Name: chrome, Priority: 1, PID: 11998, Runtime: 1
Name: chrome, Priority: 1, PID: 11997, Runtime: 3
Name: chrome, Priority: 1, PID: 11996, Runtime: 2
Name: vim, Priority: 1, PID: 11992, Runtime: 3
Name: bash, Priority: 0, PID: 1000, Runtime: 8
Name: systemd, Priority: 0, PID: 1, Runtime: 5
Process: eclipse, Priority: 2, PID: 23642, Runtime: 2
Deleted emacs: Name: emacs, Priority: 3, PID: 11993, Runtime: 1
Deleted PID 12235: Name: gedit, Priority: 2, PID: 12235, Runtime: 4
```

(The full output was extremely long. Can be seen by running the program yourself)