FirstLab

Use the file `FirstLabAnswers.docx` to answer questions in this lab.

Introduction

Lab 1 is a review of C programming and an introduction to Linux processes. You have to implement a doubly linked list of process structures (or process control blocks). You are provided several files to start your program. The files are located on Canvas > Files > Labs > FirstLab.

- `proc.h` and a skeleton `proc.c` - this C module maintains a double linked list of process structures. `proc.h` is fully coded. `proc.c` has several pre-coded functions, but there are others you must design and implement to get the program running.
- `split.h` and `split.c` - this C module defines a `split` function that splits a C string into an array of `char` pointers. `split` is similar to the Java `String.split` function. You should study this code also. `split` is used in `main` to process commands and arguments.
- `main.c` - is full coded, but it will not work until you have designed and implemented `proc.c`.
- `makefile` - a simple makefile that can be used to make the program `firstLab` from the `proc` module, `split` module, and `main.c`
- `testX_in.txt` and `testX_out.txt` - test cases with inputs and expected output. You can run the test cases using the following
  ```
  $ ./firstLab < test1_in.txt  # displays output in terminal
  $ ./firstLab < test1_in.txt > file.txt # places output in file.txt
  ```

You must complete the `proc.c` file such that the `main.c` file executes correctly. You have the expected output file for each input file. A Linux process has context information (like registers) and transitions through various states. For example, a Linux process can be running, waiting to run, waiting on I/O to complete, and terminated. In this lab, you see Linux process information maintained in a `struct proc`; however, we do not manipulate this process information. Instead we create mechanisms that allow us to maintain a queue of processes, where the queue is ordered in priority. The idea of a priority queue is that the highest priority process is at the head of the queue. You enqueue a process with its priority. Your algorithm determines where the enqueed process is placed in the queue. A normal queue is a first-in-first-out (FIFO) data structure. A priority queue allows a higher priority element to be placed in front of lower priority elements. Consider a process queue that has four processes: A, B, C, and D with priorities 4, 10, 6, and 32. This process queue is ordered as follows.

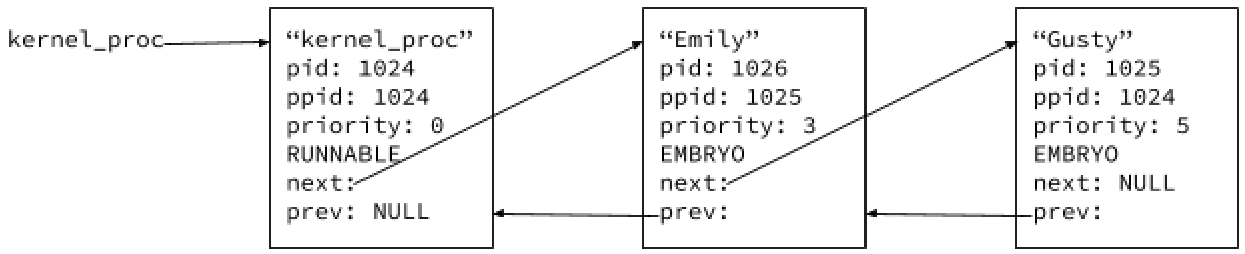
1. Process A has priority 4 and is the first process in the queue.
2. Process C has priority 6 and is the second process in the queue.
3. Process B has priority 10 and is the third process in the queue.

4. Process D has priority 32 and is the fourth process in the queue.

A dequeue operation on this queue results Process A, which is the process with priority 4. If you enqueue a process X with priority 8, it placed between Process C and Process B.

Diagram

This assignment is like software maintenance. You have code, which has a problem and your job is to fix the problem. When performing software maintenance, you have to understand the data structures and algorithms of the existing code. A diagram of the data structure can be most helpful. Your program must maintain a priority queue in a double linked list of `struct proc` data structures. The variable `struct proc *kernel_proc` is initialized by the function `bootstrap`. `kernel_proc` points to an OS kernel process that is priority 0. You can think that the process described by `*kernel_proc` runs when no other process is running. Thus your linked list always has the `struct proc` that is the kernel process (`*kernel_proc`), and it is the first `struct proc` in the list. As other procs are created they are inserted after the `struct proc` pointed to by `kernel_proc`. The following diagram shows a doubly linked list of `struct proc`s ordered in priority order. Suppose process “Emily” forks a new process named “Coletta” with a priority of 4. The “Coletta” process would be placed between “Emily” and “Gusty”. If “Coletta” process had a priority of 10, it would be placed after “Gusty.



Test Cases

Several test cases - input and expected values are provided. The input test cases use several commands. You should study the input test cases and the `main.c` algorithms to see how the commands are processed.

- `fork Gusty 5 kernel_process` - This command creates a new process named `Gusty`. `Gusty` has priority 5. `Gusty`’s parent process is the `kernel_process`.
- `fork Emily 3 Gusty` - This command creates a new process named `Emily`. `Emily` has priority 3. `Emily`’s parent process is `Gusty`.
- `printprocs` - This command prints all procs in the list. Sample output is.

```
procs in queue:
pname: kernel_process, pid: 1024, ppid: 1024, priority: 0, state: RUNNABLE
pname: Emily, pid: 1026, ppid: 1025, priority: 3, state: EMBRYO
```

```
pname: Gusty, pid: 1025, ppid: 1024, priority: 5, state: EMBRYO
```

- `kill Emily` - This command removes the process named Emily from the list.

Submissions

Be sure your submission uses the `FirstLabAnswers.docx` to answer the questions that follow so I can easily decipher your answers in your submission.

1. Submit your code on Canvas. Do not place code here.
2. Submit a run log of you executing the test cases provided. Copy/paste run log here.
3. Create at least three other test cases and expected results. Think of some concepts you want to test. Describe the concept of your test case. Simply adding another fork to an existing test case will not count as a test case. Copy/paste your test cases here.
4. Submit a run log of you executing your test cases. Copy/paste run log here.
5. Describe the algorithm used by the `split` module by answering the following questions.
    a. `split` returns a `char **`. What is a `char **`? Describe a `char **.`
    b. `split` has two loops. The first loop is a counting loop. What is the first loop counting?
    c. How does the second loop use the count computed in the fist loop?
6. What relationships between files does the `makefile` define?
7. Can you solve the lab 1 problem with a singly linked list? Justify your answer.
8. Write a reflective report on your performance on lab 1. Your reflections must include the following.
    a. Whether you think the lab is easy/difficult.
    b. Why do you think the lab is easy/difficult?
    c. Do you think you could more easily solve this lab in another programming language? If so, why and what language?
    d. What aspects of C programming do you need to improve?
    e. What is your strategy for improving your C programming?
    f. Do you think it is important to improve your C programming?
    g. What part of lab 1 was most difficult for you?
    h. What part of lab 1 was easiest for you?

Place your reflective report here.