Homework Assignment #1

J. H. Wang

Mar. 13, 2024

Reference

- Regular Exercises, available at: https://os-book.com/OS10/regular-exercises/index-exer.html

Homework #1

- Written exercises:
  - Chap.1: 1.16
  - Chap.2: 2.15, 2.19
  - Chap.3: 3.12, 3.18
- *Programming exercises:
  - Programming Problems: 2.24*, 3.19* (, 3.21**, 3.27**)
    - Note: Each student must complete all programming problems on your own
  - Programming Projects for Chap. 2* & Chap. 3*
    - [Team-based] To select at least one programming project from each chapter
- Due: two weeks (Mar. 28, 2024)

Written Exercises

- Chap.1
  - 1.16: *Direct memory access* is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.
    - (a) How does the CPU interface with the device to coordinate the transfer?
    - (b) How does the CPU know when the memory operations are complete?
    - (c) The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of the user programs? If so, describe what forms of interference are caused.

- Chap. 2
  - 2.15: What are the two models of *interprocess communication*? What are the strengths and weakness of the two approaches?
  - 2.19: What is the main advantage of the *microkernel* approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

- Chap. 3
  - 3.12: Describe the actions taken by a kernel to context-switch between processes.
  - 3.18: Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

Programming Problems

- Chap. 2
  - 2.24*: In Sec.2.3, we described a program that copies the contents of one file to a destination file.
    - This program works by first prompting the user for the name of the source and destination files.
  - Write this program using either POSIX or Windows API.
    - Be sure to include all necessary error checking, including ensuring that the source file exists.
      (… to be continued)

(… continued from the previous slide)

–Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. (Assume the name of the executable file is FileCopy)

- Linux systems provide the *strace* utility
  - (strace ./FileCopy)
- Solaris and Mac OS X systems use the *dtrace* command
  - (sudo dtrace ./FileCopy)
- As Windows systems do not provide such features, you will have to trace through the Windows version of this program using a debugger.

- Chap. 3:
  - 3.19*: Write a C program called time.c that determines the amount of time necessary to run a command from the command line.
  - This program will be run as "./time <command>" and will report the amount of elapsed time to run the specified command.
  - This will involve using fork() and exec() functions, as well as the gettimeofday() function to determine the elapsed time.
  - It will also require the use of two different IPC mechanisms.

    (…to be continued…)

- The first version will have the child process write the starting time to a region of shared memory before it calls exec().
  - After the child process terminates, the parent will read the starting time from shared memory.
  - Refer to Section 3.7.1 for details using POSIX shared memory.
- The second version will use a pipe.
  - The child will write the starting time to the pipe, and the parent will read from it following the termination of the child process.

- [optional] (3.21**): The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

  $n = n/2$, if n is even
  $n = 3*n+1$, if n is odd

  The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1.
  For example, if n=35, the sequence is: 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.

  (...to be continued...)

- Write a C program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the command line.
  - For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1.
- Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence.
- Have the parent invoke the wait() call to wait for the child process to complete before exiting the program.
- Perform necessary error checking to ensure that a positive integer is passed on the command line.

- [optional] (3.27**): Design a file-copying program named filecopy using ordinary pipes.
  - This program will be passed two parameters: the name of the file to be copied, and the name of the copied file
  - The program will then create an ordinary pipe and write the contents of the file to be copied to the pipe
  - The child process will read this file from the pipe and write it to the destination file

  (…to be continued)

- For example, if we invoke the program as follows:
  - filecopy input.txt copy.txt
  - The file input.txt will be written to the pipe. The child process will read the contents of this file and write it to the destination file copy.txt.
- You may write this program using either UNIX or Windows pipes.

Programming Projects

- Programming Project for Chap. 2:
  Project 1: Linux Kernel Modules
  - I. Kernel modules overview
  - II. Loading and removing kernel modules
  - III. The /proc file system

- IV. Assignment:
- 1. Design a kernel module that creates a /proc file named /proc/jiffies that reports the current value of jiffies when the /proc/jiffies file is read, such as with the command:

  cat /proc/jiffies

  Be sure to remove /proc/jiffies when the module is removed.

- 2. Design a kernel module that creates a proc file named /proc/seconds that reports the number of elapsed seconds since the kernel module was loaded.
- This will involve using the value of jiffies as well as the HZ rate.
- When a user enters the command

  cat /proc/seconds

  your kernel module will report the number of seconds that have elapsed since the kernel module was first loaded.
- Be sure to remove /proc/seconds when the module is removed.

Programming Projects

- Programming Projects for Chap. 3 (Choose one)
- Project 1: UNIX Shell and History Feature
  - This project consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process
  - Your implementation will support input and output redirection, as well as pipes as a form of IPC between a pair of commands.
  - Completing this project will involve using the UNIX fork(), exec(), wait(), dup2(), and pipe() system calls and can be completed on any Linux, UNIX, or macOS system.
    - I. Overview
    - II. Executing command in a child process
    - III. Creating a history feature
    - IV. Redirecting input and output
    - V. Communication via a pipe

- Project 2: Linux Kernel Module for Task Information
  - In this project, you will write a kernel module that uses /proc file system for displaying a task's information in a Linux system.
    - Be sure to review the programming project in Chap.2 before you begin this project
    - It can be completed using the Linux virtual machine provided with the textbook
  - Part I – Writing to the /proc file system
  - Part II – Reading from the /proc file system

- Project 3: Linux Kernel Module for Listing Tasks
  - In this project, you will write a kernel module that lists all current tasks in a Linux system.
    - Be sure to review the programming project in Chap.2 before you begin this project
    - It can be completed using the Linux virtual machine provided with the textbook
  - Part I - Iterating over tasks linearly
    - Assignment: Design a kernel module that iterates through all tasks in the system using the *for_each_process()* macro. In particular, output the task name (known as executable name), state, and process id of each task. Write this code in the module entry point so that its contents will appear in the kernel log buffer, which can be viewed using the *dmesg* command.

- Part II - Iterating over tasks with a depth-first search (DFS) tree
  - Assignment: Beginning from the *init* task, design a kernel module that iterates over all tasks in the system using a DFS tree. Just as in the first part of this project, output the name, state, and *pid* of each task. Perform this iteration in the kernel entry module so that its output appears in the kernel log buffer.
  - To verify that you have indeed performed an appropriate DFS iteration, you will have to examine the relationships among the various tasks output by the ps command.

- Project 4: Kernel Data Structures
- Part I – Inserting, Deleting Elements to/from the linked lists
- Assignment:
  - In the module entry point, create a linked list containing four struct color elements.
  - Traverse the linked list and output its contents to the kernel log buffer.
  - Invoke the dmesg command to ensure that the list is properly constructed once the kernel module has been loaded.
  - In the module exit point, delete the elements from the linked list and return the free memory back to the kernel.
  - Again, invoke the dmesg command to check that the list has been removed once the kernel module has been unloaded.

- Part II – Parameter Passing
- Assignment:
  - Design a kernel module named collatz that is passed an initial value as a module parameter.
  - Your module will then generate and store the sequence in a kernel linked list when the module is loaded.
  - Once the sequence has been stored, your module will traverse the list and output its contents to the kernel log buffer.
  - Use the dmesg command to ensure that the sequence is properly generated once the module has been loaded.
  - In the module exit point, delete the contents of the list and return the free memory back to the kernel.
  - Again, use dmesg to check that the list has been removed once the kernel module has been unloaded.

Notes for Programming Exercises

- Some programming exercises (programming problems 3.19, 3.21, and all programming projects in Chap. 2 and 3) require C programming in Linux/UNIX
- You are suggested to use the Linux virtual machine provided with the textbook

Homework Submission

- For hand-written exercises, please hand in your homework on paper in class
- For programming exercises, please upload your program to the iSchool+ as follows:
  - Program uploading: a compressed file (in .zip format) including source codes, execution snapshot, and documentation
  - The documentation should clearly identify:
    - Team members and responsibility
    - Compilation or configuration instructions if it needs special environment to compile or run
  - Please contact with the TA if you are unable to upload your homework

Any Questions?