

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | Ruoxuan Fu |
| **Student ID:** | 222019321062060 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Process Control Block in Linux | | | |
| Date | | March, 2021 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements** 2. Understand process control block in Linux 3. Review the writing, compiling and running of kernel modules 4. Use kernel module to read and manipulate PCBs in Linux kernel | | | | | |
| 1. **Experimental environment (**platform and software**)**   Virtualbox + Ubuntu (or other platform+linux system combinations) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Tasks for this lab    1. Task 1 3. Find the sched.h file containing the definition of process control block (task\_struct) on your virtual machine 4. Locate the starting line and the ending line of task\_struct 5. Locate the lines that define process identifier, process state, as well as and the process’s executable name    1. Task 2   Write, compile, and load a Linux kernel module to travese through the list of PCBs in the kernel, and   * + 1. Output to the kernel buffer the PID, state, and the executable name of each process     2. Count the number of processes in your system (hint: wc -l)     3. Compare the results in i) and ii) with the list given by the ‘ps -el’ command   1. Task 3   Write, compile, and load a Linux kernel module to travese through the list of PCBs in the kernel, and at the same time remove the PCB of a certain process **P** from the list. Use the kernel module in Task 2 to traverse the list of processes in your system again and check that the process **P** cannot be found. This gives you some sense of how a running process can be hidden.   1. Please provide your procedure and source codes to perform the tasks.   Task1：   * + - 1. Use files and search tool to locate “sched.h”      * + - 1. Find the staring lines and ending lines        * + - 1. Use search tool to the lines that define process identifier, process state, as well as and the process’s executable name         Task2:   1. Write a kernel module as a c source file 2. #include <linux/module.h> 3. #include <linux/kernel.h> 4. #include <linux/sched/signal.h> 6. /\* This function is called when the module is loaded. \*/ 7. **int** pl\_entry(**void**) 8. { 9. **struct** task\_struct \*curr; 10. **int** count = 0; 11. for\_each\_process(curr) 12. { 13. printk(KERN\_INFO "PID:%d;State:%ld;Name:%s\n",curr->pid,curr->state,curr->comm); 14. count++; 15. } 16. printk(KERN\_INFO "The total number of process is :%d\n",count); 17. **return** 0; 18. } 20. /\* This function is called when the module is removed. \*/ 21. **void** pl\_exit(**void**) { 22. printk(KERN\_INFO "Exit!\n"); 23. } 25. /\* Macros for registering module entry and exit points. \*/ 26. module\_init(pl\_entry); 27. module\_exit(pl\_exit); 29. MODULE\_LICENSE("GPL"); 30. MODULE\_DESCRIPTION("list processes"); 31. MODULE\_AUTHOR("zhaohj"); 33. /\* ps -el | wc -l \*/   2. Write the makefile   1. obj-m += test.o 2. ll: 3. make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules 4. lean: 5. make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean   3. Run in the terminal      Task3:  1. Write a infinity loop in the test c source file   1. #include <stdio.h> 2. #include <unistd.h> 4. **int** main() 5. { 6. printf("hello world pid: %d\n", getpid()); 8. **while**(1); 10. **return** 0; 11. }   2. Write a kernel program to remove the test process from  the link list   1. #include <linux/module.h> 2. #include <linux/kernel.h> 3. #include <linux/sched/signal.h> 4. #include <linux/sched.h> 5. #include <linux/list.h> 6. #include <linux/types.h> 8. /\* This function is called when the module is loaded. \*/ 9. **int** pl\_entry(**void**) 10. { 11. **struct** task\_struct \*curr; 13. for\_each\_process(curr) 14. { 15. **if**(curr->pid == 2698) 16. { 17. **struct** list\_head \*temp1 = &(curr->tasks),\*temp2= temp1->prev; 18. temp2->next = temp1->next; 19. } 21. } 23. **return** 0; 24. } 26. /\* This function is called when the module is removed. \*/ 27. **void** pl\_exit(**void**) { 28. printk(KERN\_INFO "Exit!\n"); 29. } 31. /\* Macros for registering module entry and exit points. \*/ 32. module\_init(pl\_entry); 33. module\_exit(pl\_exit); 35. MODULE\_LICENSE("GPL"); 36. MODULE\_DESCRIPTION("list processes"); 37. MODULE\_AUTHOR("zhaohj"); 39. /\* ps -el | wc -l \*/   3. Run the test program and install this kernel module      4. I use the kernel module in task2 to verify the result, it can be. Seen that the test process is removed from the linklist | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Results:  Totally, the result of this experiment is satisfying. However, the kernel module in task3 cannot remove the test process from the result of command “ps -el”.  Harvest:  In order to describe the operation of the control process, the data structure of the management and control information stored in the system is called the process control block. The pcb's statement is stored in the header file <sched.h>. Pcb is implemented in linux as a structure called task\_struct, wherein an integer is used to represent pid, a long pattern is used to represent state, and the name of the process is saved with an array of characters.  Linux defines a macro in "signal.h" that can be used quickly to access all processes. This macro is as follows: init\_task system processes from the beginning of the init\_task is the process structure linkhead. #define for\_each\_process(p)  for (p = &init\_task ; (p = next\_task(p)) != &init\_task ; )  The scheduling queue for a process is implemented in linux as a bidirectional list. At the same time, we don't need to write our own list of access to each process. In linux, each task\_struct structure is nested with a type list\_head called task. His members include a leading pointer to the tasks in the previous process and a back pointer to the tasks in the last process.  The result of 'ps -el' also sees that the reason for the process that has moved out of the dispatch queue is that ps, top, and so on, view the process information by calling the readdir method to traverse the /proc directory to get process information. The fd, mem, io, cpuset, and other process information about the process is recorded in detail below each dynamically created process ID number. proc is a virtual file system that we use to communicate with user space in the Linux kernel space. In the proc file system, we can read and write virtual files as a means of communicating with entities in the kernel, but unlike ordinary files, the contents of these virtual files are created dynamically. So as long as the process is still recorded in /proc, the command to view the process will always see that the process exists. So the way to hide a process more thoroughly is to not let the process information be generated under proc.  Problems:  None. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |