

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2020-2021** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Process Operations in Linux | | | |
| Date | | April, 2021 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements** 2. Learn the basic process operations in Linux 3. Review the writing, compiling and running of kernel modules 4. Understand process control block 5. Use kernel module to access process control block | | | | | |
| 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   Experiment with the fork(), exec(), wait(), and exit() system calls for process operations in Linux.   * + 1. In the main process, create a child process     2. Load a new task in the child process     3. The main process waits for the child to exit, and accesses the child’s exit status     4. The child exits with status 255   1. Task 2  1. Find the sched.h file containing the definition of process control block (task\_struct) on your virtual machine 2. Locate the starting line and the ending line of task\_struct 3. Locate the lines that define process identifier, process state, as well as and the process’s executable name    1. Task 3   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     3. Compare the results in i) and ii) with the list given by the‘ps -el’command (hint: useful command **wc -l**)  1. Please provide your procedure and source codes to perform the tasks.   Task1:  The code and screenshots:  The main code in mywait.c, the main process create a child process by fork().    The executed program results:    We can see that the value of ‘status’ is -1, but shown as 255, because the function WEXITSTATUS() return a 8-bit number, so -1 is same as 255.  Task2:  The results are shown in the screenshot:  Loaction of sched.h:    Location of task\_struct starting and ending: line 630 and line 1346    Location of some important variables:      Task3:  I wrote the kernel program code as follows:    Compile the kernel program and display its output messages:    Kernel program shows 313 processes:    Ps -el | wc -l command shows 314 processes:    We can see that ps -el | wc -l command counted 1 more processes than kernel program, because ‘ps’ itself is a new process. This result is as expected | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Through this experiment, I have gained a deeper understanding of Linux kernel functions, including their usage, operation mechanism and some of their properties, which will help me a lot in using the Linux system and writing Linux programs. At the same time, I learned something about the underlying structure of the file sched.h, which also helped me in some way. At the end, I tried to write a kernel program to count the number of processes instead of a command. I encountered some difficulties during the experiment, but I was able to complete the experiment after searching for information and documentation, and I got the same results as expected. In the process of completing the experiment I also learnt about the use of the Linux pipe command | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |