

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2023-2024** |
| **Major**: | Software Engineering |
| **Class**: | 2022 |
| **Student Name**: | 吴孜远 |
| **Student ID:** | 222022321062009 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Process Control Block in Linux | | | |
| Date | | March, 2024 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements** 2. Review the writing, compiling and running of kernel modules 3. Understand process control block and get familiar with the definition of PCB in the Linux source code 4. Can use kernel modules to traverse the list of PCBs and manipulate the list of PCBs. | | | | | |
| 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. Read the source codes for the PCB, i.e. the task\_struct struct in the sched.h file, and try to understand what all the fields in the struct means. 3. Write, compile, and load a Linux kernel module to travese through the list of PCBs and manipulate the list:    * 1. Start a process and keep it running; get the PID of the running process, say P.      2. Write a kernel module K1 to traverse the list of processes and find the running process P;      3. Write another kernel module K2 to remove the PCB of process P from the doubly linked list of PCBs.      4. Use kernel module K1 to list all the processes in the system again and see if process P can still be found. 4. Please provide your procedure and source codes to perform the tasks.   1)  首先查看linux内核的版本，用于判断我们应该看哪一个sched.h    使用find / -name命令寻找sched.h文件，  并打开查看。  主要看的是task\_struct结构体，这里是从738行到1549行    2)  i. 我们编译链接并运行这个hello，获取它的进程号P，它是一个不会停止的进程。    可以看到，pid为4336  Ii. find\_hello用于在进程链中寻找进程hello，这个寻找是通过遍历PCB链，看有没有进程号为P的进程，打印说明是否找到。Delete\_hello用于修改进程链的指针，将进程hello从进程链中断开，最重要的是了解每个task\_struct是如何连接在一起的，它们是通过内置的tasks（类型为struct list\_head），list\_head中有两个指针，分别指向前后task中的tasks。这样，我们只需要调整前后指针就好了。  首先编译链接这两个内核模块：    在hello运行的时候加载find\_hello进入内核，并打印日志：    发现能够找到进程hello。  这时我们再移除模块find\_hello，并打印日志。    成功移除。  这时我们加载delete\_hello模块，并打印日志：    然后移除这个模块，    现在按理来说已经找不到hello进程了，我们再加载find\_hello模块看一看：  确实没有找到，hello在进程链表中消失了。  这次实验全程用make tool来编译链接，真的方便，makefile如下：   1. *# 模块源文件列表* 2. MODULES := find\_hello.o delete\_hello.o 3. *# 目标模块列表，模块名.ko* 4. obj-m += find\_hello.o delete\_hello.o 5. all: 6. make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules 7. clean: 8. make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean 9. hello: hello.o 10. gcc hello.o -o hello 11. hello.o: hello.c 12. gcc -c hello.c   hello.c如下：   1. #include <stdio.h> 2. #include <unistd.h> 3. int main() 4. { 5. printf("hello world pid: %d\n", getpid()); 6. while(1); 7. return 0; 8. }   内核模块find\_hello如下：   1. #include <linux/module.h> 2. #include <linux/kernel.h> 3. #include <linux/sched/signal.h> 4. */\* This function is called when the module is loaded. \*/* 5. int pl\_entry(void) 6. { 7. struct task\_struct \*curr; 8. int found = 0; 9. for\_each\_process(curr) 10. { 11. int currpid = 4336; 12. if(curr->pid == currpid) *//please change the number to the PID of your own hello process* 13. { 14. found = 1; 15. break; 16. } 17. } 18. if (found == 1) 19. printk(KERN\_INFO "I found the PCB of hello world!\n"); 20. else 21. printk(KERN\_INFO "I could not found the PCB of hello world!\n"); 22. return 0; 23. } 24. */\* This function is called when the module is removed. \*/* 25. void pl\_exit(void) { 26. printk(KERN\_INFO "Exit find hello!\n"); 27. } 28. */\* Macros for registering module entry and exit points. \*/* 29. module\_init(pl\_entry); 30. module\_exit(pl\_exit); 31. MODULE\_LICENSE("GPL"); 32. MODULE\_DESCRIPTION("find hello"); 33. MODULE\_AUTHOR("ziyuan wu");   内核模块delete\_hello如下：   1. #include <linux/module.h> 2. #include <linux/kernel.h> 3. #include <linux/sched/signal.h> 4. */\* This function is called when the module is loaded. \*/* 5. int pl\_entry(void) 6. { 7. struct task\_struct \*curr; 8. for\_each\_process(curr) 9. { 10. int currpid = 4336; 11. if(curr->pid == currpid) *//please change the number to the PID of your own hello process* 12. { 13. struct list\_head \*pre = curr->tasks.prev; 14. struct list\_head \*next = curr->tasks.next; 15. pre->next = next; 16. next->prev = pre; 18. printk(KERN\_INFO "hello has been removed from the list\n"); 19. } 20. } 21. return 0; 22. } 23. */\* This function is called when the module is removed. \*/* 24. void pl\_exit(void) { 25. printk(KERN\_INFO "Exit find hello!\n"); 26. } 27. */\* Macros for registering module entry and exit points. \*/* 28. module\_init(pl\_entry); 29. module\_exit(pl\_exit); 30. MODULE\_LICENSE("GPL"); 31. MODULE\_DESCRIPTION("delete hello"); 32. MODULE\_AUTHOR("ziyuan wu"); | | | | | |
| 1. **Result analysis and discussion**   In this part, you are required to provide your analysis of experimental results and summing up the harvest and the existing problems; besides, you are required to provide your thinkings about the questions:   * Please investigate the definition of task\_struct in sched.h and check out how to output the *state* and *name* of each process. * After you remove the PCB from the doubly linked list, can you still use the command ps -el to find the hello process? Think about why.   **一．**  我们可以用state字段和comm字段来获取进程的状态或者名字： state字段表示进程的状态，例如睡眠、运行、僵尸等。状态值通常是一些宏定义，如TASK\_RUNNING、TASK\_INTERRUPTIBLE、TASK\_UNINTERRUPTIBLE等。  comm字段是一个字符数组，用于存储进程的名称。  **二．**  执行ps命令时，它会查询系统内核的进程列表，获取每个活跃进程的PCB信息。如果一个进程的PCB被从系统进程列表中删除，这意味着该进程已经完成了它的执行或者被操作系统终止。在这种情况下，该进程不再是活跃的，因此不会在ps命令的输出中出现。不过即使一个进程的PCB被删除，它可能仍然存在于系统中，这种状态下的进程被称为僵尸进程。僵尸进程的PCB仍然存在于系统中，但是它们不会在ps命令的默认输出中显示，因为它们不是活跃的进程。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |