

**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 scheduling in Linux: The CFS Scheduling | | | |
| Date | | April，2024 | Type | | □Confirmatory  □Design  √ Comprehensive |
| 1. **Objective & Requirements**    1. Understanding the concept of processor affinity    2. Learn how to set processor affinity in linux    3. Learn how to set process priority in linux    4. Understanding the CFS scheduling policy of linux    5. Review how to compile and load kernel module in linux. Learn how to pass parameters to kernel modules.    6. Learn how to access kernel scheduling information by loading kernel modules, and certify the CFS scheduling strategy. | | | | | |
| 1. **Experimental environment (**platform and software**)**   Virtualbox+Ubuntu linux | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Tasks for this lab   Understand how the scheduling information is represented in PCB. Write, compile and load a kernel module to access the real runtime and virtual runtime of two running processes with different priorities. Note that the two processes need to be bound to the same CPU core. It is suggested that you set the cpu affinity in the main function, and then create two processes using the fork() system call to get a parent and child process, and the child process will inherit the cpu affinity of its parent. To facilitate the task, you may need to learn how to pass parameters to kernel modules. Compute the ratio of real runtime and virtual runtime of the two processes, and compare the ratios with the ratio of the two processes’priorities. This indicates that you’d better change the default priorities of the parent/child process to make them different. In this way you can certify the CFS scheduling policy.    Figure: The weights in CFS for different NICE values   1. Please provide your procedure to perform the tasks and source codes.   首先创建两个进程，使用 sched\_setaffinity 设置当前进程（0代表当前进程）的CPU亲和性，使用 fork() 创建一个子进程，然后分别设置子进程和父进程的priority，并打印输出便于查看和确认。最后给这两个进程都设置了while(1)无限循环，让这两个进程在实验过程中一直执行。  代码如下：   1. #include <stdio.h> 2. #ifndef \_\_USE\_GNU *//double underscore* 3. #define \_\_USE\_GNU *//double underscore* 4. #endif 5. #include <unistd.h> 6. #include <sched.h> 7. #include <sys/resource.h> 8. */\** 9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*sudo to execute this program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 10. \*/ 11. int main(int argc, char \*argv[]) 12. { 13. int i = 0; 15. cpu\_set\_t mask; 17. CPU\_ZERO(&mask);    *//clear cpu settings* 18. CPU\_SET(0, &mask);   *//set core 0 (exclusively)* 19. sched\_setaffinity(0, sizeof(mask), &mask); *//0 represent current process* 20. *//affinity will be inherted by child process* 22. pid\_t pid = fork(); 24. if(pid == 0)*//child* 25. { 27. *// [-20, 19]: from high to low priorities* 28. setpriority(PRIO\_PROCESS, 0, 5); *//set priority* 29. *//PRIO\_PROCESS means to set the priority of a process,* 30. *//rather than a group of processes* 31. *//0 means current process* 32. *//[-20, 19]* 34. printf("child's new pri: %d\n", getpriority(PRIO\_PROCESS, 0)); *//0 means current process* 35. printf("child's pid: %d\n", getpid()); 37. while(1); 38. } 39. else *//parent* 40. { 41. setpriority(PRIO\_PROCESS, 0, -5); 42. printf("parent's pri: %d\n", getpriority(PRIO\_PROCESS, 0)); *//0 means current process* 43. printf("parent's pid: %d\n", getpid()); 44. while(1); 45. } 47. return 0; 48. }   然后编写内核模块，在这之间别忘了在include内核模块相关库的同时include linux/sched/signal.h，它提供调度器相关函数和结构的访问。  因为每次要查看的进程的进程号都不一样，所以我们为了不修改源代码，采用命令行参数的形式，将进程号传入函数。使用 module\_param 宏将 pid1 和 pid2 作为模块参数进行声明。  使用 for\_each\_process(curr) 宏遍历系统中的所有进程，分别用if语句找到两个进程。进程的真实执行时间和虚拟执行时间都保存在sched\_entity类型的se中。sched\_entity 结构体是用于实现完全公平调度器（CFS）的，它包含了与进程调度相关的信息。sched\_entity 结构体是调度器用来维护进程的虚拟运行时和其他调度信息的基础。  获取两个时间的语句为curr->se.vruntime 和curr->se.sum\_exec\_runtime  代码如下：   1. #include <linux/module.h> 2. #include <linux/kernel.h> 3. #include <linux/sched/signal.h> 4. int pid1; 5. int pid2; 6. module\_param(pid1, int, S\_IRUGO); *//parameter name; parameter type; parameter access permission* 7. module\_param(pid2, int, S\_IRUGO); *//S\_IRUGO: can be read by UGO but cannot be modified* 8. int module\_load(void) 9. { 10. struct task\_struct \*curr; 11. for\_each\_process(curr) 12. { 13. if(curr->pid == pid1) *//change the number to the PID of your own hello process* 14. { 15. printk(KERN\_INFO "pid1 vruntime: %lld", curr->se.vruntime); 16. printk(KERN\_INFO "pid1 runtime: %lld", curr->se.sum\_exec\_runtime); 17. } 19. if(curr->pid == pid2) *//change the number to the PID of your own hello process* 20. { 21. printk(KERN\_INFO "pid2 vruntime: %lld", curr->se.vruntime); 22. printk(KERN\_INFO "pid2 runtime: %lld", curr->se.sum\_exec\_runtime); 23. } 24. } 25. return 0; 26. } 27. void module\_rm(void) { 28. printk(KERN\_INFO "Module removed!\n"); 29. } 30. module\_init(module\_load); 31. module\_exit(module\_rm); 32. MODULE\_LICENSE("GPL"); 33. MODULE\_DESCRIPTION("kernel parameters"); 34. MODULE\_AUTHOR("wuziyuan");   我是用maketool来编译链接这两个文件：   1. obj-m += cfs.o 2. all: 3. make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules 4. clean: 5. make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean 6. pri: priority.o 7. gcc priority.o -o pri 8. priority.o: priority.c 9. gcc -c priority.c   实验过程如下:  首先编译链接pri，使得两个进程开始运行    我们可以用ps -el查看所有进程的信息，确认我们的两个进程优先级设置成功    然后我们编译链接内核模块：    编译链接完成后，我们就可以加载进内核，然后查看日志    pid1 vruntime: 40570852815  pid1 runtime: 40572931381  pid2 vruntime: 40570746200  pid2 runtime: 123618522431    P1和p2真实执行时间的比值约为3.047  根据这个nice与weight的对应图，和公式  进程获得的小时间片 t = 调度周期 T\* 进程权重 w / 所有进程权重之和（W）  可以得出，本测试用例中理论真实时间之比为3121/1024 = 3.048，  实际值和理论值几乎无差。  接下来设置重复组nice值分别为(0, 5),(-5, 5)  结果：  Pid1=0 and pid2=5    实验Pid1:pid2 = 0.3268  理论Pid1:pid2 = 0.3271  Pid1=-5 and pid2=5    实验Pid1:pid2 = 0.10729  理论Pid1:pid2 = 0.10733 | | | | | |
| 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:   * Why do you need to bound the two processes to the same core for this lab? * Suppose that you do not use the fork() system call to create the two processes. Instead, you create two individual processes that have no parent-child relation, and bound them to the same cpu core. Then try to get the virtual and real runtime of the two processes. Report and analyze your result.   **思考题**   1. **将两个进程绑定到同一个核心，是因为我们本来就是要研究单个cpu核心对进程的调度策略，如果两个进程绑定到了不同的核心，那就会涉及到不同核心之间的进程分配，我们不想研究这个或者说受到这种调度的干扰。** 2. **不用fork，那就用clone创建独立进程。子进程的PID通常小于父进程的PID，我实验中都是相邻的。而独立进程可以具有任意的PID，它们之间没有直接的数字关系。除此之外我没有发现其它的区别。** | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |