УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Операционные системы»

**Лабораторная работа №2**

*Вариант Procfs: page, vm\_area\_struct*

Студент

*Кошкарбаев Н.Ю.*

*P33121*

Преподаватель

*Осипов С. В.*

Санкт-Петербург, 2022 г.

Описание задания

Разработать комплекс программ на пользовательском уровне и уровне ярда, который собирает информацию на стороне ядра и передает информацию на уровень пользователя, и выводит ее в удобном для чтения человеком виде. Программа на уровне пользователя получает на вход аргумент(ы) командной строки (не адрес!), позволяющие идентифицировать из системных таблиц необходимый путь до целевой структуры, осуществляет передачу на уровень ядра, получает информацию из данной структуры и распечатывает структуру в стандартный вывод. Загружаемый модуль ядра принимает запрос через указанный в задании интерфейс, определяет путь до целевой структуры по переданному запросу и возвращает результат на уровень пользователя.

*Весь исходный код ядра, измененный мной:*

*Отследить изменения можно, посмотрев историю commit-ов.*

Выполнение

Код модуля ядра:

1. #include <linux/kernel.h>
2. #include <linux/init.h>
3. #include <linux/module.h>
4. #include <linux/kdev\_t.h>
5. #include <linux/fs.h>
6. #include <linux/cdev.h>
7. #include <linux/device.h>
8. #include <linux/slab.h>
9. #include <linux/uaccess.h>
10. #include <linux/proc\_fs.h>
11. #include <linux/pid.h>
12. #include <linux/sched.h>
13. #include <linux/sched/signal.h>
14. #include <linux/device.h>
15. #include <linux/memblock.h>
17. #define BUF\_SIZE 1024
19. MODULE\_LICENSE("Dual BSD/GPL");
20. MODULE\_DESCRIPTION("lab2 osi");
21. MODULE\_VERSION("1.0");
23. **static** **int** pid = 1;
24. **static** **int** struct\_id = 1;
25. **static** unsigned **long** adress = 0;
27. **static** **struct** proc\_dir\_entry \*parent;
29. **static** **int** \_\_init lab\_driver\_init(**void**);
30. **static** **void** \_\_exit lab\_driver\_exit(**void**);
32. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Procfs Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
33. **static** **int** open\_proc(**struct** inode \*inode, **struct** file \*file);
34. **static** **int** release\_proc(**struct** inode \*inode, **struct** file \*file);
35. **static** ssize\_t read\_proc(**struct** file \*filp, **char** \_\_user \*buffer, size\_t
36. length,loff\_t \* offset);
37. **static** ssize\_t write\_proc(**struct** file \*filp, **const** **char** \*buff, size\_t len,
38. loff\_t \* off);
40. /\*
41. \*\* procfs operation sturcture
42. \*/
43. **static** **struct** file\_operations proc\_fops = {
44. .open = open\_proc,
45. .read = read\_proc,
46. .write = write\_proc,
47. .release = release\_proc
48. };

51. // page
53. **static** **struct** page \*get\_my\_page(**struct** mm\_struct\* mm,
54. unsigned **long** address) {

57. pgd\_t \*pgd = pgd\_offset(mm, address);
58. **if** (pgd\_none(\*pgd) || pgd\_bad(\*pgd)) {
59. **return** NULL;
60. }
62. p4d\_t \*p4d = p4d\_offset(pgd, address);
63. **if** (p4d\_none(\*p4d) || p4d\_bad(\*p4d)) {
64. **return** NULL;
65. }
67. pud\_t \*pud = pud\_offset(p4d, address);
68. **if** (pud\_none(\*pud) || pud\_bad(\*pud)) {
69. **return** NULL;
70. }
72. pmd\_t \*pmd = pmd\_offset(pud, address);
73. **if** (pmd\_none(\*pmd) || pmd\_bad(\*pmd)) {
74. **return** NULL;
75. }
77. pte\_t \*pte = pte\_offset\_kernel(pmd, address);
78. **if** (!pte) {
79. **return** NULL;
80. }
82. **return** pte\_page(\*pte);

85. }

88. **static** size\_t write\_page\_struct(**char** \_\_user \*ubuf,
89. **struct** task\_struct \*task\_struct\_ref,
90. unsigned **long** address) {
91. **char** buf[BUF\_SIZE];
92. size\_t len = 0;
94. **struct** page \*page\_struct;
95. **struct** mm\_struct \*mm = task\_struct\_ref->mm;

98. **if** (mm == NULL) {
99. sprintf(buf, "Task\_struct's mm is NULL\n");
100. **return** 0;
101. }
102. **struct** vm\_area\_struct \*vm\_current = mm->mmap;
103. unsigned **long** start = vm\_current->vm\_start;
104. unsigned **long** end = vm\_current->vm\_end;
105. **while** (start <= end) {
106. page\_struct = get\_my\_page(mm, start);
107. **if** (page\_struct != NULL) {
108. len += sprintf(buf + len, "flags = %ld\n", page\_struct->flags);
109. len += sprintf(buf + len, "va = %x\n", start);
110. len += sprintf(buf + len, "pa= %x\n", page\_struct->mapping);
111. **break**;
112. }
113. start += PAGE\_SIZE;
114. }
115. **if** (copy\_to\_user(ubuf, buf, len)) {
116. **return** -EFAULT;
117. }
118. **return** len;
120. }

123. **static** size\_t write\_vm\_area\_struct(**char** \_\_user \*ubuf,
124. **struct** task\_struct \*task\_struct\_ref,
125. unsigned **long** address) {
126. **char** buf[BUF\_SIZE];
127. size\_t len = 0;
128. **struct** mm\_struct \*mm = task\_struct\_ref->mm;
129. **if** (mm == NULL) {
130. sprintf(buf, "Task\_struct's mm is NULL\n");
131. **return** 0;
132. }
133. **struct** vm\_area\_struct \*vm\_current = mm->mmap;
134. len += sprintf(buf + len, "vm\_start = %ld\n", vm\_current->vm\_start);
135. len += sprintf(buf + len, "vm\_end = %ld\n", vm\_current->vm\_end);
136. len += sprintf(buf + len, "rm\_subtree\_gap = %ld\n", vm\_current->rb\_subtree\_gap);
137. len += sprintf(buf + len, "vm\_flags = %ld\n", vm\_current->vm\_flags);
138. len += sprintf(buf + len, "vm\_pgoff = %ld\n", vm\_current->vm\_pgoff);
139. len += sprintf(buf + len, "swap\_readahead\_info = %ld\n", (vm\_current->vm\_flags));
140. **if** (copy\_to\_user(ubuf, buf, len)) {
141. **return** -EFAULT;
142. }
143. **return** len;

146. }

149. // otkritie
150. **static** **int** open\_proc(**struct** inode \*inode, **struct** file \*file)
151. {
152. printk(KERN\_INFO "proc file opend.....\t");
153. **return** 0;
154. }
156. //zakritie
157. **static** **int** release\_proc(**struct** inode \*inode, **struct** file \*file)
158. {
159. printk(KERN\_INFO "proc file released.....\n");
160. **return** 0;
161. }
163. //chetie
165. **static** ssize\_t read\_proc(**struct** file \*filp,
166. **char** \_\_user \*ubuf,
167. size\_t count,
168. loff\_t \*ppos) {
170. **char** buf[BUF\_SIZE];
171. **int** len = 0;
172. **struct** task\_struct \*task\_struct\_ref = get\_pid\_task(find\_get\_pid(pid), PIDTYPE\_PID);
174. printk(KERN\_INFO "proc file read.....\n");
175. **if** (\*ppos > 0 || count < BUF\_SIZE){
176. **return** 0;
177. }
179. **if** (task\_struct\_ref == NULL){
180. len += sprintf(buf,"task\_struct for pid %d is NULL.\n",pid);
182. **if** (copy\_to\_user(ubuf, buf, len)){
183. **return** -EFAULT;
184. }
185. \*ppos = len;
186. **return** len;
187. }
189. **switch**(struct\_id){
190. **default**:
191. **case** 0:
192. len = write\_page\_struct(ubuf, task\_struct\_ref, adress);
193. **break**;
194. **case** 1:
195. len = write\_vm\_area\_struct(ubuf, task\_struct\_ref, adress);
196. **break**;
197. }
199. \*ppos = len;
200. **return** len;
201. }

204. // zapis'
205. **static** ssize\_t write\_proc(**struct** file \*filp, **const** **char** \_\_user \*ubuf, size\_t
206. count, loff\_t \*ppos) {
208. **int** num\_of\_args, c, a, b, adr;
209. **char** buf[BUF\_SIZE];
211. printk(KERN\_INFO "proc file wrote.....\n");
213. **if** (\*ppos > 0 || count > BUF\_SIZE){
214. **return** -EFAULT;
215. }
217. **if**( copy\_from\_user(buf, ubuf, count) ) {
218. **return** -EFAULT;
219. }
221. num\_of\_args = sscanf(buf, "%d %d", &a, &b);
222. **if** (num\_of\_args != 2){
223. **return** -EFAULT;
224. }
226. struct\_id = a;
227. pid = b;
229. c = strlen(buf);
230. \*ppos = c;
231. **return** c;
232. }
234. // init
235. **static** **int** \_\_init lab\_driver\_init(**void**) {
237. parent = proc\_mkdir("lab2",NULL);
239. **if**( parent == NULL )
240. {
241. pr\_info("Error creating proc entry");
242. **return** -1;
243. }
245. proc\_create("struct\_info", 0666, parent, &proc\_fops);
247. pr\_info("Device Driver Insert...Done!!!\n");
248. **return** 0;
249. }
251. // exit from module
252. **static** **void** \_\_exit lab\_driver\_exit(**void**)
253. {
254. // dlya 1 zapisi
255. //remove\_proc\_entry("lab/struct\_info", parent);
257. // dlya full udaleniya
258. proc\_remove(parent);
260. pr\_info("Device Driver Remove\nDone.\n");
261. }
263. module\_init(lab\_driver\_init);
264. module\_exit(lab\_driver\_exit);

Напишем Makefile-ы:

*obj-m += nik\_module.o*

*all:*

*make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules clean:*

*make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean*

Код пользователя:

1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <sys/stat.h>
4. #include <fcntl.h>
5. #include <unistd.h>
6. #include <string.h>
7. #include <stdlib.h>
8. #include <errno.h>
10. **void** help(){
11. fprintf(stderr, "Usage: ./user structure\_ID PID\n "
12. "0 - page\_struct\n "
13. "1 - vm\_area\_struct\n");
14. }
16. **int** main(**int** argc, **char** \*argv[]){
17. **if** (argc != 3) help();
18. **if** (argc < 3){
19. fprintf(stderr, "Not enough arguments \n" );
20. **return** 0;
21. }
22. **if** (argc > 3){
23. fprintf(stderr, "Too many arguments \n" );
24. **return** 0;
25. }
27. **char** \*p;
28. errno = 0;
29. **long** structure\_ID = strtol(argv[1], &p, 10);
30. **if** (\*p != '\0' || errno != 0){
31. fprintf(stderr, "Provided structure\_ID must be number.\n");
32. help();
33. **return** 0;
34. }
35. errno = 0;
36. **long** PID = strtol(argv[2], &p, 10);
37. **if** (\*p != '\0' || errno != 0){
38. fprintf(stderr, "Provided PID must be number.\n");
39. help();
40. **return** 0;
41. }

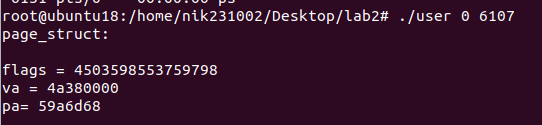
44. **if** (structure\_ID !=0 && structure\_ID !=1){
45. fprintf(stderr, "Provided structure ID is not supported.\n");
46. help();
47. **return** 0;
48. }
50. **if**(PID<0){
51. fprintf(stderr, "PID must be positive \n" );
52. help();
53. **return** 0;
54. }
56. **char** inbuf[4096];
57. **char** outbuf[4096];
58. **int** fd = open("/proc/lab2/struct\_info", O\_RDWR);
59. sprintf(inbuf, "%s %s", argv[1], argv[2]);
61. write(fd, inbuf, 17);
62. lseek(fd, 0, SEEK\_SET);
63. read(fd, outbuf, 4096);
65. **if** (structure\_ID == 0){
66. printf("page\_struct: \n\n");
67. } **else** {
68. printf("vm\_area\_struct: \n\n");
69. }
70. puts(outbuf);
71. **return** 0;
72. }

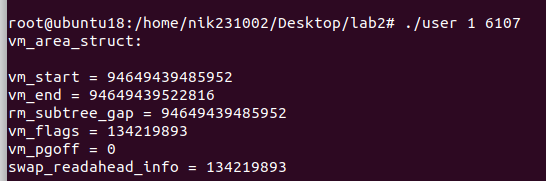
Загружаем модуль в ядро, собираем пользовательскую сторону и тестируем:

*make*

*insmod nik\_module.ko*

*gcc user.c -o user*

**

**

Выводы

Во время выполнения лабораторной работы я углубился в работу ядра linux. Написал собственные системные вызовы, добавил их в ядро и проверил работоспособность.