

4.3 实验 4-3：后台守护进程

1. 实验目的

通过本实验了解和熟悉 Linux 是如何创建和使用后台守护进程的。

2. 实验要求

- 1) 写一个用户程序，创建一个守护进程。
- 2) 该守护进程每隔 5 秒去查看当前内核的日志中是否有 oops 错误。

3. 实验步骤

下面是本实验的实验步骤。

进入本实验的参考代码目录进行交叉编译。

```
cd
/home/lab466/runninglinuxkernel_4.0/rlk_lab/rlk_basic/chapter_8/lab3_daemon
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
```

编译 test 测试 app。然后把它拷贝到 runninglinuxkernel_4.0/kmodules 目录下面。

```
arm-linux-gnueabi-gcc daemon_test1.c -o daemon_test1 --static
cp daemon_test1 /home/lab466/runninglinuxkernel_4.0/kmodules
```

启动 QEMU+runninglinuxkernel。最好另外开一个窗口，运行：

```
sudo su
cd /home/lab466/runninglinuxkernel_4.0
sh run.sh arm32
```

运行 daemon_test1 程序。

```
cd /mnt
./daemon_test1
```

ls

```
benshushu:lab3_daemon# cd /mnt/
benshushu:mnt# ls
2021.2.16.12.40.56.log  2021.2.16.12.41.6.log  false_sharing  rlk_lab
2021.2.16.12.41.1.log  README                 hello.stp      test
benshushu:mnt#
```

在 `mnt` 目录可以看到“2021.2.16*.log”文件。打开这些 log 文件，可以看到的内容是内核的 `dmesg` 的日志。

另外我们通过 `top` 命令可以看到 `daemon_test1` 进程，进程 PID 为 775。

`top`

```
Mem: 34868K used, 52496K free, 6384K shrd, 0K buff, 6588K cached
CPU:  1.6% usr 13.0% sys  0.0% nic 78.8% idle  0.0% io  0.0% irq  6.5% sirq
Load average: 0.06 0.20 0.11 2/46 781
```

PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
781	772	0	R	2308	2.6	1	13.0	top
11	2	0	SW	0	0.0	1	5.6	[ksoftirqd/1]
15	2	0	SW	0	0.0	2	0.8	[ksoftirqd/2]
7	2	0	SW	0	0.0	2	0.8	[rcu_sched]
410	2	0	SW	0	0.0	0	0.8	[kworker/0:1]
1	0	0	S	2308	2.6	1	0.0	{linuxrc} init
772	1	0	S	2308	2.6	0	0.0	~/bin/sh
775	1	0	S	768	0.8	0	0.0	./daemon_test1
753	2	0	SWN	0	0.0	1	0.0	[kmemleak]
6	2	0	SW	0	0.0	1	0.0	[kworker/u8:0]
3	2	0	SW	0	0.0	0	0.0	[ksoftirqd/0]
19	2	0	SW	0	0.0	3	0.0	[ksoftirqd/3]
24	2	0	SW	0	0.0	3	0.0	[kworker/u8:1]
23	2	0	SW	0	0.0	1	0.0	[kdevtmpfs]
409	2	0	SW	0	0.0	1	0.0	[kworker/1:1]
328	2	0	SW	0	0.0	3	0.0	[kworker/3:1]
2	0	0	SW	0	0.0	1	0.0	[kthreadd]
411	2	0	SW	0	0.0	2	0.0	[kworker/2:1]
9	2	0	SW	0	0.0	0	0.0	[migration/0]
14	2	0	SW	0	0.0	2	0.0	[migration/2]

4. 实验代码

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <time.h>
5 #include <fcntl.h>
6 #include <string.h>
7 #include <sys/stat.h>
8 #include <sys/klog.h>
9
10 #define FALLBACK_KLOG_BUF_SHIFT 17 /* CONFIG_LOG_BUF_SHIFT in kernel */
11 #define FALLBACK_KLOG_BUF_LEN (1 << FALLBACK_KLOG_BUF_SHIFT)
12
13 #define KLOG_CLOSE 0
14 #define KLOG_OPEN 1
15 #define KLOG_READ 2
16 #define KLOG_READ_ALL 3
17 #define KLOG_READ_CLEAR 4
18 #define KLOG_CLEAR 5
19 #define KLOG_CONSOLE_OFF 6
20 #define KLOG_CONSOLE_ON 7
21 #define KLOG_CONSOLE_LEVEL 8
22 #define KLOG_SIZE_UNREAD 9
23 #define KLOG_SIZE_BUFFER 10
24
25 /* we use 'Linux version' string instead of Oops in this lab */
26 // #define OOPS_LOG "Oops"
27 #define OOPS_LOG "Linux version"
28
29 int save_kernel_log(char *buffer)
30 {
31     char path[128];
32     time_t t;
```

```

33     struct tm *tm;
34     int fd;
35
36     t = time(0);
37     tm = localtime(&t);
38
39     snprintf(path, 128, "/mnt/%d.%d.%d.%d.%d.log", tm->tm_year+1900,
40              tm->tm_mon+1, tm->tm_mday, tm->tm_hour,
41              tm->tm_min, tm->tm_sec);
42     printf("%s\n", path);
43
44     fd = open(path, O_WRONLY|O_CREAT, 0644);
45     if(fd == -1) {
46         printf("open error\n");
47         return -1;
48     }
49     write(fd, buffer, strlen(buffer));
50     close(fd);
51
52     return 0;
53 }
54
55 int check_kernel_log()
56 {
57     char *buffer;
58     char *p;
59     ssize_t klog_size;
60     int ret = -1;
61     int size;
62
63     printf("start kernel log\n");
64
65     klog_size = klogctl(KLOG_SIZE_BUFFER, 0, 0);
66     if (klog_size <= 0) {
67         klog_size = FALLBACK_KLOG_BUF_LEN;
68     }
69
70     printf("kernel log size: %d\n", klog_size);
71
72     buffer = malloc(klog_size + 1);
73     if (!buffer)
74         return -1;
75
76     size = klogctl(KLOG_READ_ALL, buffer, klog_size);
77     if (size < 0) {
78         printf("klogctl read error\n");
79         goto done;
80     }
81
82     buffer[size] = '\0';
83
84     /* check if oops in klog */
85     p = strstr(buffer, OOPS_LOG);
86     if (p) {
87         printf("we found '%s' on kernel log\n", OOPS_LOG);
88         save_kernel_log(buffer);
89         ret = 0;
90     }
91 done:
92     free(buffer);
93     return ret;
94 }
95
96 int main(void)

```

```
97 {
98     if(daemon(0,0) == -1) {
99         printf("daemon error");
100         return 0;
101     }
102
103     while(1) {
104         check_kernel_log();
105
106         sleep(5);
107     }
108
109     return 0;
110}
```