

Linux 进程控制操作应用编程



实验报告

- 实验课题作业 4: Linux 进程控制操作应用编程
- 姓名: 梁嘉嘉
- 学号: 23125240
- 课程: 高级操作系统
- 提交日期: 2024/10/24

1. 实验目的

本实验旨在通过编写 C 语言程序，掌握Linux操作系统中的进程控制机制，包括进程的创建、进程的同步、进程间的通信以及信号处理等核心概念。通过具体的实验任务，学习如何使用 `fork()` 函数创建父子进程，如何使用信号和软中断控制进程的终止，以及如何利用无名管道实现进程间的数据通信，深入理解Linux操作系统中进程管理的基本操作和工作原理。

2. 实验环境

硬件环境:

- 虚拟机平台: VMware Workstation Pro 17
- 处理器: AMD Ryzen 7 8845H with Radeon 780M Graphics
 - 架构: x86_64
 - CPU 核心数: 2 个核心
 - 每个核心的线程数: 4
 - L1 缓存: 64 KiB (2 实例)
 - L2 缓存: 2 MiB (2 实例)
 - L3 缓存: 32 MiB (2 实例)
- 内存: 4 GiB
 - 可用内存: 2.6 GiB
 - 交换分区: 3.7 GiB
- 存储:

- 硬盘大小：80 GB
- 分区：单一分区 / 挂载在 sda2

软件环境：

- 操作系统：Ubuntu 24.04.1 LTS (noble)
- 内核版本：6.8.0-45-generic
- 编译工具：
 - GCC 版本：13.2.0 (Ubuntu)
 - Make 版本：4.3 (GNU Make)
 - Git 版本：2.43.0

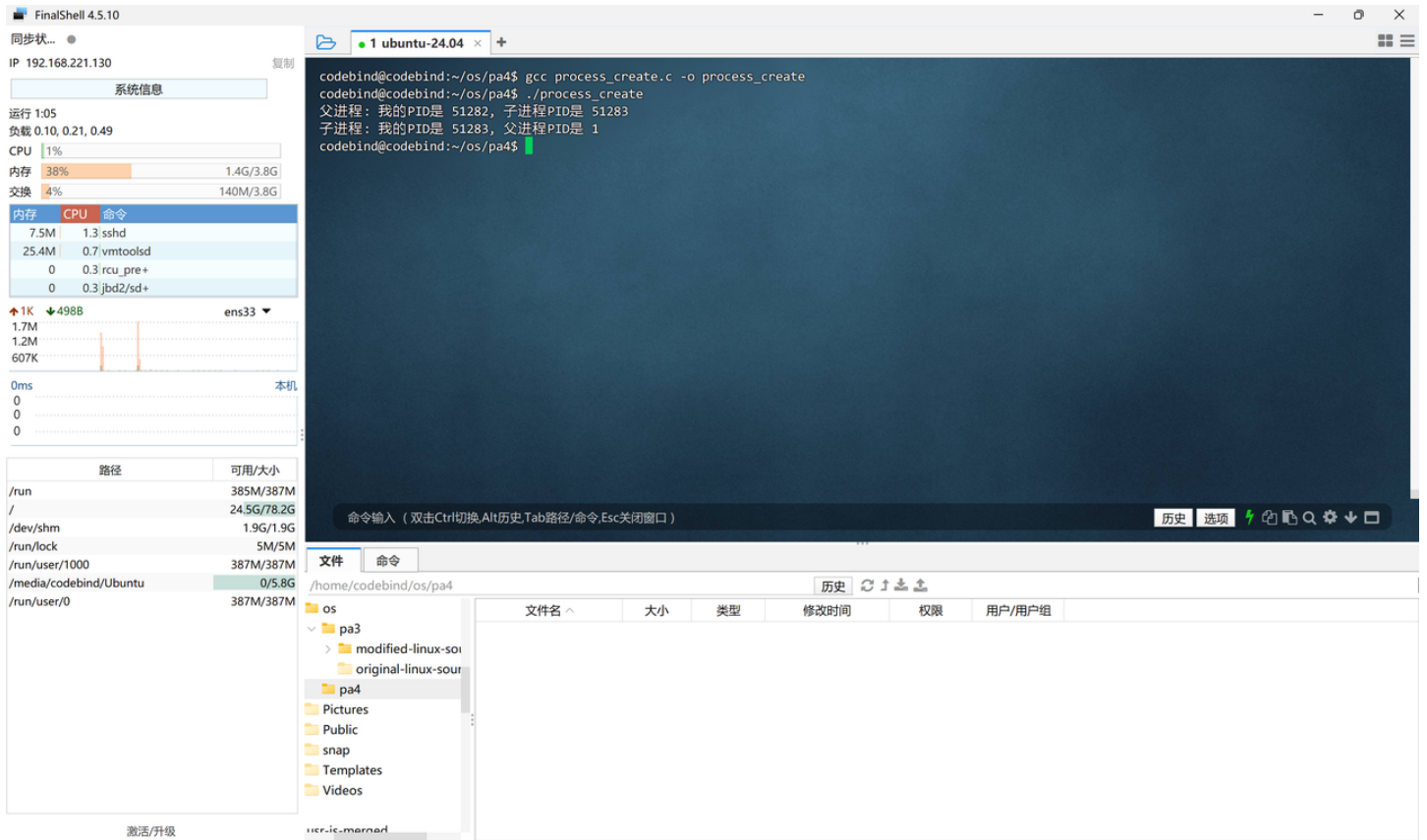
3. 实验步骤

3.1 进程创建实验

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4
5 int main() {
6     pid_t pid = fork(); // 创建子进程
7
8     if (pid < 0) {
9         // 创建子进程失败
10        printf("Fork failed!\n");
11        return 1;
12    } else if (pid == 0) {
13        // 子进程
14        printf("子进程：我的PID是 %d，父进程PID是 %d\n", getpid(), getppid());
15    } else {
16        // 父进程
17        printf("父进程：我的PID是 %d，子进程PID是 %d\n", getpid(), pid);
18    }
19
20    return 0;
21 }
22
```

```
1 $ gcc process_create.c -o process_create
```

```
2 $ ./process_create
```



3.2 字符串循环显示与终止

1. 声明了 `sig_atomic_t` 类型的全局变量 `stop`，用于表示是否需要停止循环。

```
1 volatile sig_atomic_t stop;
```

`volatile` 关键字告诉编译器，变量值可能会在程序运行时异步改变，因此每次访问该变量时，都需要重新读取值而不是使用缓存。

2. 信号处理函数：当程序收到 `SIGINT` 信号（即用户按下 `Ctrl+C` 时），系统会调用此函数。在该函数中，`stop` 被设置为1，这将用于终止主循环。

```
1 void handle_sigint(int sig) {  
2     stop = 1;  
3 }
```

3. 主函数

```

1 int main() {
2     signal(SIGINT, handle_sigint); // 捕捉 Ctrl+C 信号
3
4     while (!stop) {
5         printf("Viva la chine!\n");
6         sleep(1); // 延迟1秒
7     }
8
9     printf("\nAu revoir!\n");
10    return 0;
11 }

```

当用户按下 `Ctrl+C` 时，信号处理函数将 `stop` 设置为 1，这会使循环结束。

随后，程序打印 `Au revoir!` 并正常终止。

```

1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 volatile sig_atomic_t stop;
6
7 void handle_sigint(int sig) {
8     stop = 1;
9 }
10
11 int main() {
12     signal(SIGINT, handle_sigint); // 捕捉 Ctrl+C 信号
13
14     while (!stop) {
15         printf("Viva la chine!\n");
16         sleep(1); // 延迟1秒
17     }
18
19     printf("\nAu revoir!\n");
20     return 0;
21 }
22

```

```

1 $ gcc loop_string.c -o loop_string
2 $ ./loop_string

```

FinalShell 4.5.10

同步状态... IP 192.168.221.130

系统信息

运行 1:06
负载 0.06, 0.18, 0.46
CPU 1%
内存 37% 1.4G/3.8G
交换 4% 140M/3.8G

内存	CPU	命令
0	1	rcu_pre+
7.5M	1	sshd
5.5M	0.7	top
7.2M	0.3	systemd+

ens33

47K
32K
16K

0ms
0
0

本机

路径	可用/大小
/run	385M/387M
/	24.5G/78.2G
/dev/shm	1.9G/1.9G
/run/lock	5M/5M
/run/user/1000	387M/387M
/media/codebind/Ubuntu	0/5.8G
/run/user/0	387M/387M

激活/升级

1 ubuntu-24.04

```
codebind@codebind:~/os/pa4$ gcc loop_string.c -o loop_string
codebind@codebind:~/os/pa4$ ./loop_string
Viva la chine!
Viva la chine!
Viva la chine!
Viva la chine!
Viva la chine!
Viva la chine!
^C
Au revoir!
codebind@codebind:~/os/pa4$
```

命令输入 (双击Ctrl切换,Alt历史,Tab路径/命令,Esc关闭窗口)

历史 选项

文件 命令

/home/codebind/os/pa4

历史

文件名	大小	类型	修改时间	权限	用户/用户组
os					
pa3					
modified-linux-soi					
original-linux-sour					
pa4					
Pictures					
Public					
snap					
Templates					
Videos					

3.3 父子进程同步

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <signal.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 volatile sig_atomic_t stop = 0;
8
9 void handle_sigusr1(int sig) {
10     stop = 1;
11 }
12
13 int main() {
14     signal(SIGUSR1, handle_sigusr1); // 捕捉软中断信号
15
16     pid_t pid = fork(); // 创建子进程
17
18     if (pid < 0) {
19         printf("Fork failed!\n");
20         return 1;
21     } else if (pid == 0) {
22         // 子进程
23         while (!stop) {
24             printf("I am fine, thanks, and you?\n");
```

```

25         sleep(1);
26     }
27     printf("Child exited!\n");
28 } else {
29     // 父进程
30     printf("Program started...\n");
31     for (int i = 0; i < 5; i++) {
32         printf("How are you?\n");
33         sleep(1);
34     }
35     kill(pid, SIGUSR1); // 给子进程发送软中断信号
36     wait(NULL); // 等待子进程终止
37     printf("I am fine too! Program ended!\n");
38 }
39
40 return 0;
41 }

```

```

1 $ gcc synchronized_process.c -o synchronized_process
2 $ ./synchronized_process

```

The screenshot shows a terminal window with the following content:

```

codebind@codebind:~/os/pa4$ gcc synchronized_process.c -o synchronized_process
codebind@codebind:~/os/pa4$ ./synchronized_process
父进程: Program started...
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: I am fine, thanks, and you?
父进程: How are you?
子进程: Child exited!
父进程: I am fine too! Program ended!
codebind@codebind:~/os/pa4$

```

On the left side of the terminal, there is a system monitoring panel with the following data:

系统信息
运行 1:15
负载 0.10, 0.10, 0.29
CPU 1%
内存 37% 1.4G/3.8G
交换 4% 140M/3.8G

Below the system information, there is a table showing the top processes:

内存	CPU	命令
7.5M	1	sshd
5.5M	0.7	top
0	0.7	kworker+
0	0.3	rcu_pre+

At the bottom of the terminal, there is a file manager window showing the directory structure:

```

/home/codebind/os/pa4
├── os
│   ├── pa3
│   │   ├── modified-linux-soi
│   │   └── original-linux-sour
│   └── pa4
│       ├── Pictures
│       ├── Public
│       ├── snap
│       ├── Templates
│       └── Videos
└── user-is-mornd

```

3.4 基于无名管道的父子进程通信



`pipe(fd)` 用于创建一个无名管道，`fd` 是一个大小为2的整数数组，其中：

- `fd[0]` 表示管道的读端。
- `fd[1]` 表示管道的写端。

程序的整体流程：

1. **父进程** 先创建一个无名管道和一个子进程。
2. **父进程** 通过管道写入字母 `A` 到 `Z`，每个字母写 60 次。
3. **子进程** 从管道中读取数据并显示收到的字母。
4. **父进程** 完成所有字母的写入后，发送 `SIGUSR1` 信号给子进程，通知其停止工作。
5. **子进程** 接收到信号后，停止读取管道数据并终止，输出 “Child exited!”。
6. **父进程** 等待子进程结束后，输出 “Program ended!” 并终止。

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <signal.h>
4 #include <string.h>
5 #include <sys/wait.h>
6
7 volatile sig_atomic_t stop = 0;
8
9 void handle_sigusr1(int sig) {
10     stop = 1;
11 }
12
13 int main() {
14     int fd[2];
15     char ch;
16     pipe(fd); // 创建无名管道
17
18     signal(SIGUSR1, handle_sigusr1); // 捕捉软中断信号
19
20     pid_t pid = fork(); // 创建子进程
21
22     if (pid < 0) {
23         printf("Fork failed!\n");
24         return 1;
25     } else if (pid == 0) {
26         // 子进程：读取管道数据
27         close(fd[1]); // 关闭写端
28         while (!stop) {
```

```

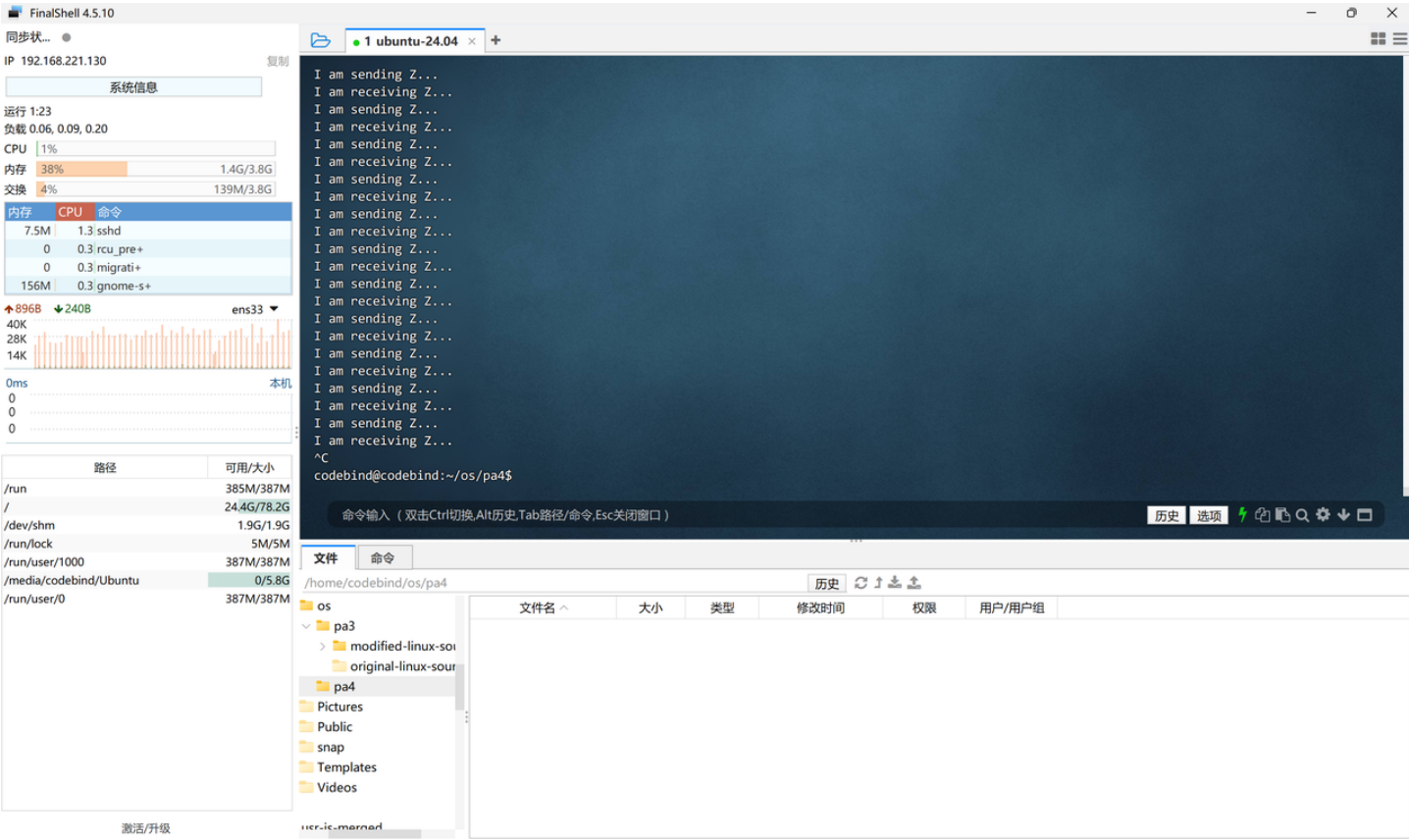
29         if (read(fd[0], &ch, 1) > 0) {
30             printf("I am receiving %c...\n", ch);
31         }
32     }
33     printf("Child exited!\n");
34     close(fd[0]); // 关闭读端
35 } else {
36     // 父进程：写入管道数据
37     close(fd[0]); // 关闭读端
38     char letters[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
39     for (int i = 0; i < 26; i++) {
40         for (int j = 0; j < 60; j++) {
41             write(fd[1], &letters[i], 1);
42             printf("I am sending %c...\n", letters[i]);
43             usleep(10000); // 延迟10ms
44         }
45     }
46     kill(pid, SIGUSR1); // 给子进程发送软中断信号
47     wait(NULL); // 等待子进程终止
48     printf("Program ended!\n");
49     close(fd[1]); // 关闭写端
50 }
51
52 return 0;
53 }
54

```

```

1 $ gcc pipe_communication.c -o pipe_communication
2 $ ./pipe_communication

```

4. 疑难解惑与经验教训

无

5. 结论与体会

通过本次实验，我加深了对 Linux 进程控制机制的理解，学会了如何在父子进程间进行同步、通信以及如何通过信号控制进程的行为。使用 `fork()` 创建进程、`pipe()` 实现管道通信、以及捕捉和处理信号等操作让我更好地掌握了 C 语言在操作系统级别的编程技巧。实验中遇到的难点主要集中在父子进程间的同步和通信方面，经过调试，成功实现了进程之间的协作。此次实验不仅巩固了进程管理的理论知识，也提升了我解决实际编程问题的能力。