



中南大學  
CENTRAL SOUTH UNIVERSITY

# Linux 文件系统

## 实验报告

学生姓名	王雅蓉
学 号	8206200602
专业班级	计科 2003
指导教师	沈海澜
学 院	计算机学院
完成时间	2023 年 06 月 10 日

计算机学院  
2023 年 06 月

## 目录

第 1 章 实验概述 .....	2
1.1 实验目的 .....	2
1.2 准备知识 .....	2
1.3 实验内容 .....	2
1.4 实验要求 .....	2
第 2 章 实验设计 .....	3
2.1 /proc 下有关文件分析 .....	3
2.2 CPU 类型和型号的获取 .....	3
2.3 Linux 版本获取 .....	4
2.4 从启动到当前经过的时间获取 .....	4
2.5 当前内存状态获取 .....	5
2.6 内容修改 .....	6
第 3 章 实验代码 .....	7
3.1 文件内容的获取代码 read_info.c .....	7
3.2 文件内容修改代码 alter_info.c .....	9
第 4 章 总结与收获 .....	10

## 第 1 章 实验概述

### 1.1 实验目的

- (1) 熟悉利用 `proc` 文件系统在用户态下获得内核参数的方法;
- (2) 熟悉文件系统常用系统调用;

### 1.2 准备知识

- (3) Linux 的虚拟文件系统;
- (4) `proc` 文件系统;
- (5) Linux 文件系统的主要系统调用;

### 1.3 实验内容

编写一个 C 程序, 通过 Linux 文件系统调用, 用户态下访问 `/proc` 文件系统, 获得内核参数 (CPU 的类型、型号、所使用的 Linux 的版本、从启动到当前时刻经过的时间、当前内存状态等或其他参数), 修改可以修改的参数。

- (1) 本实验总共完成了以下内容:
- (2) CPU 类型与型号的获取
- (3) Linux 版本的获取
- (4) 从启动到当前时刻经过的时间的获取
- (5) 当前内存状态的获取
- (6) 修改了 kernel 的 `hostname` 主机名

### 1.4 实验要求

- (1) 掌握 Linux 环境下应用程序的编辑、运行、调试方法
- (2) 完成实验内容要求实现的任务, 并记录实验过程
- (3) 撰写实验报告, 包括实验目的、实验内容、实验要求、实验代码及运行截图、实验心得等。

## 第 2 章 实验设计

### 2.1 /proc 下有关文件分析

Linux 内核提供了一种通过 /proc 文件系统，在运行时访问内核内部数据结构、改变内核设置的机制。proc 文件系统是一个虚拟文件系统，它只存在内存当中，而不占用外存空间。它以文件系统的方式为访问系统内核数据的操作提供接口。该目录下有许多重要的文件，在本实验中用到的文件及其功能如下：

表 2.1 /proc 下文件及其功能解析

文件名	文件内容解释
cpuinfo	CPU 的硬件信息（型号、家族、缓存大小）
meminfo	物理内存、交换空间等的消息，系统内存占用情况等
version	指明了当前正在运行的内核版本
uptime	记录了操作系统从启动到现在的时间；系统空闲的时间
sys/kernel	用于调整和监控 linux 内核操作中的一些杂项和普通项

### 2.2 CPU 类型和型号的获取

CPU 类型和型号的获取需要进入 cpuinfo 文件，通过 cat 命令可以看到本机中 cpuinfo 文件的内容的格式如图：


```
felia@felicia-virtual-machine:/proc$ cat cpuinfo
processor       : 0
vendor_id      : AuthenticAMD
cpu family     : 23
model          : 96
model name     : AMD Ryzen 7 PRO 4750U with Radeon Graphics
stepping       : 1
microcode      : 0x8600109
cpu MHz        : 1696.814
cache size     : 512 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 2
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 16
wp             : yes
```

图 2.2.1 文件内部格式图

由图可以看出，cpu 的型号所属类别为 model name，因此找到 model 所在行，

读取冒号后的信息即可。读取时的核心代码及运行结果如下，

```
while(getdelim(&arg, &size, 0, cpuinfo) != -1) {
    char *match = strstr(arg, "model name");
    if(match) {
        match += strlen("model name");
        while(*match == ' ' || *match == '\t' || *match == ':')
            ++match;
        char *end = match;
        while(*end != '\n') ++end;
        *end = 0;
        printf("%s\n", match);
    }
}
```



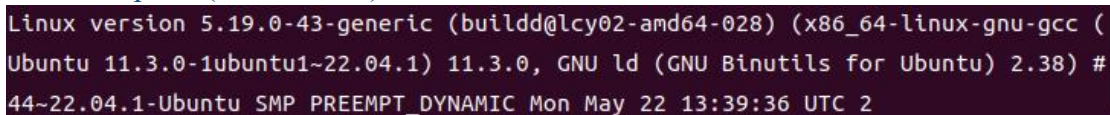
```
felicia@felicia-virtual-machine:~/Linux_research/research3$ ./read_info
AMD Ryzen 7 PRO 4750U with Radeon Graphics
```

图 2.2.2 运行图 1

### 2.3 Linux 版本获取

Version 文件中存放了 Linux 版本，因此只需获取该文件内容即可，核心代码及运行结果如下：

```
//linux 的版本
char *line=0;
ssize_t read=getline(&line,&size,version);
if(read==-1)
{
    printf("Failed to read from /proc/version\n");
    fclose(version);
    return 1;
}
printf("%s\n", line);
```



```
Linux version 5.19.0-43-generic (buildd@lcy02-amd64-028) (x86_64-linux-gnu-gcc (
Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #
44~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon May 22 13:39:36 UTC 2
```

图 2.3 运行图 2

### 2.4 从启动到当前经过的时间获取

uptime 文件中存放了系统从启动到现在一共经历的时间，以及空闲的时间，

这两个数据可以计算系统的空闲比。获取该文件中的内容即可，核心代码及运行结果如下：

```
//从启动到现在经历的时间
if (uptime == NULL) {
    printf("Failed to open /proc/uptime\n");
    return 1;
}
double up_time;
if (fscanf(uptime, "%lf", &up_time) != 1) {
    printf("Failed to read from /proc/uptime\n");
    fclose(uptime);
    return 1;
}
printf("Uptime: %.2lf seconds\n", up_time);
```



图 2.4 运行图 3

## 2.5 当前内存状态获取

当前内存的状态及占用信息存放在 meminfo 文件中，获取 meminfo 文件中的内容即可，核心代码及运行结果如下：

```
//当前内存的状态
if(meminfo==NULL)
{
    printf("Failed to read from /proc/meminfo\n");
    fclose(meminfo);
    return 1;
}
char *line2=0;
ssize_t read2;
while ((read2 = getline(&line2, &size, meminfo)) != -1) {
    printf("%s\n", line2);
}
```

```
MemTotal:      3982184 kB

MemFree:       994452 kB

MemAvailable:  1683492 kB

Buffers:       54780 kB

Cached:        837788 kB

SwapCached:    0 kB

Active:        349120 kB

Inactive:      1154076 kB

Active(anon):  2200 kB
```

图 2.5 运行图 4

## 2.6 内容修改

/proc 中存在部分文件可以修改,如/proc/sys/kernel 文件。在实验设计中,允许传入两个参数,第一个为修改的文件,第二个为更新后的内容。比如可以修改该文件中的主机名 hostname。修改运行时 Linux 系统的 hostname,即不需要重启系统,运行后立即生效,但是在系统重启后会丢失所做的修改,如果要永久更改系统的 hostname,就要修改相关的设置文件。但是需要注意,写入时需要获得 root 权限,核心代码及运行结果如下:

```
int fd = open(argv[1], O_WRONLY);
if (fd == -1) {
    perror("Failed to open file");
    return 1;}
ssize_t written = write(fd, argv[2], strlen(argv[2]));
if (written == -1) {
    perror("Failed to write to file");
    close(fd);
    return 1;
}

printf("Successfully wrote %ld bytes to %s\n", written, argv[1]);

root@felicia-virtual-machine:/home/felicia/Linux_research/research3# ./alter_info /proc/sys/kernel/hostname "new-hostname"
Successfully wrote 12 bytes to /proc/sys/kernel/hostname
```

图 2.5 运行图 5

## 第 3 章 实验代码

### 3.1 文件内容的获取代码 read\_info.c

```
#include <stdio.h>
#include <string.h>

int main() {
    FILE *cpuinfo = fopen("/proc/cpuinfo", "rb");
    FILE *version = fopen("/proc/version", "rb");
    FILE *uptime = fopen("/proc/uptime", "rb");
    FILE *meminfo = fopen("/proc/meminfo", "rb");
    char *arg = 0;
    size_t size = 0;
    //cpu 的型号信息
    while(getdelim(&arg, &size, 0, cpuinfo) != -1) {
        char *match = strstr(arg, "model name");
        if(match) {
            match += strlen("model name");
            while(*match == ' ' || *match == '\t' || *match == ':')
                ++match;
            char *end = match;
            while(*end != '\n') ++end;
            *end = 0;
            printf("%s\n", match);
        }
    }
    //linux 的版本
    char *line=0;
    ssize_t read=getline(&line,&size,version);
    if(read==-1)
    {
        printf("Failed to read from /proc/version\n");
        fclose(version);
    }
}
```



```
        return 1;
    }
    printf("%s\n", line);
    //从启动到现在经历的时间
    if (uptime == NULL) {
        printf("Failed to open /proc/uptime\n");
        return 1;
    }
    double up_time;
    if (fscanf(uptime, "%lf", &up_time) != 1) {
        printf("Failed to read from /proc/uptime\n");
        fclose(uptime);
        return 1;
    }
    printf("Uptime: %.2lf seconds\n", up_time);
    //当前内存的状态
    if(meminfo==NULL)
    {
        printf("Failed to read from /proc/meminfo\n");
        fclose(meminfo);
        return 1;
    }
    char *line2=0;
    ssize_t read2;
    while ((read2 = getline(&line2, &size, meminfo)) != -1) {
        printf("%s\n", line2);
    }
    free(line2);
    free(line);
    free(arg);
    fclose(meminfo);
    fclose(uptime);
    fclose(version);
    fclose(cpuinfo);
```

```
return 0;
```

```
}
```

### 3.2 文件内容修改代码 alter\_info.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <fcntl.h>
```

```
#include <unistd.h>
```

```
int main(int argc, char *argv[]) {
```

```
    if (argc != 3) {
```

```
        printf("Usage: %s /proc/file/to/modify \"new content\"\n", argv[0]);
```

```
        return 1;
```

```
    }
```

//一共两个参数，第一个参数是需要修改的文件，第二个参数是新的内容

///proc/sys/kernel/hostname "new-hostname"

```
    int fd = open(argv[1], O_WRONLY);
```

```
    if (fd == -1) {
```

```
        perror("Failed to open file");
```

```
        return 1;
```

```
    }
```

```
    ssize_t written = write(fd, argv[2], strlen(argv[2]));
```

```
    if (written == -1) {
```

```
        perror("Failed to write to file");
```

```
        close(fd);
```

```
        return 1;
```

```
    }
```

```
    printf("Successfully wrote %ld bytes to %s\n", written, argv[1]);
```

```
    close(fd);
```

```
    return 0;
```

```
}
```

## 总结与收获

本次有关 Linux 中 `/proc` 文件系统的实验使我收获颇多，在复习的过程中，我仿佛对 Linux 系统的核心和精髓，万物皆文件，理念简单的同时，却能实现强大的系统功能。虚拟文件系统使得多个标准系统可以在 linux 中存在，用户可以直接使用且不用担心底层实现。其中 `procfs` 为用户空间提供了内核及其控制的进程的瞬时状态的快照。在 `/proc` 中，内核发布有关其提供的设施的信息，如中断、虚拟内存和调度程序。此外，`/proc/sys` 是存放可以通过 `sysctl` 命令配置的设置的地方，可供用户空间访问。

通过本次实验，我对于 `/proc` 目录下许多文件的内容及其作用有了深入地了解，也注意到了许多细节，比如要对该目录下的系统文件做出修改时需要获取 `root` 权限，这也是在之前一直被我忽略的一点。

在未来的 Linux 系统与应用的学习中，我会更加深入学习课堂上没有深入讲解的内容，更注重自身知识的深度与厚度积累，积极探索，认真学习，在今后的实验中精益求精的要求自己。