# 第四部分 文件管理（任务十九—二十）

### 任务十九 proc文件系统的实现

### 1.实验目的

掌握proc文件系统的实现原理。

掌握文件、目录、索引节点等概念。

### 2.实验内容

编写代码，实现proc文件系统

### 3.实验步骤

**3.1 准备实验**

使用浏览器登录平台领取本次实验对应的任务，从而在平台上创建个人项目（Linux 0.11 内核项目），然后使用VSCode将个人项目克隆到本地磁盘中并打开。

**3.2实现proc文件系统**

①在MINIX 1.0文件系统中为proc文件增加一个新文件类型。在include/sys/stat.h文件的第25行后面添加新文件类型的宏定义，其代码如下：

#define S\_IFPROC 0050000

新文件类型宏定义的值应该在0010000到0100000之间，但后四位八进制数必须是0，而且不能和已有的任意一个S\_IFXXX相同

②为新文件类型添加相应的测试宏。在第37行后面添加测试宏，代码如下：

#define S\_ISPROC(m) (((m)& S\_IFMT) == S\_IFPROC)

③psinfo结点要通过mknod系统调用建立，所以要让它支持新的文件类型。在fs/namei.c文件中只需修改mknod函数中的一行代码即可，即将第647行代码修改为如下：

if(S\_ISBLK(mode) || S\_ISCHR(mode) || S\_ISPROC(mode))

④在init/main.c文件的第13行包含头文件stat.h ：  
#include<sys/stat.h>

⑤在第42行后面添加如下代码，定义mkdir和mknod两个系统调用函数：

\_syscall2(int,mkdir,const char\*,name,mode\_t,mode)

\_syscall3(int,mknod,const char\*,filename,mode\_t,mode,dev\_t,dev)

⑥在第234行复制句柄语句的后面添加如下代码，调用mkdir函数创建/proc目录，调用mknod函数创建psinfo节点：

mkdir("/proc",0755);

mknod("/proc/psinfo",S\_IFPROC|0444,0);

⑦找到fs/read\_write.c文件中的sys\_read函数，此函数中有一系列的if判断语句，再添加一个if语句，代码如下：

if(S\_ISPROC(inode->i\_mode))

return psread(inode->i\_zone[0], buf,count, &file->f\_pos);

⑧由于这里调用了psread函数，所以需要在第32行添加如下代码声明此函数：

extern int psread(int dev,char \* buf,int count,off\_t \* f\_pos);

⑨在VSCode的“文件资源管理器”窗口中，右键点击“fs”文件夹节点，在弹出的快捷菜单中选择“Reveal in File Explorer”，打开项目所在文件夹。

⑩将“学生包”本实验对应文件夹下的proc.c文件拖动到步骤1中打开的fs文件夹下，修改文件名称为procfs.c。

生成项目，确保没有语法错误。

按F5启动调试，待Linux 0.11启动以后，输入命令：ls -l /proc可以查看/proc目录的属性信息，如下图所示：



图57

输入命令：cat /proc/psinfo 即可得到当前所有进程的信息，如下图所示：

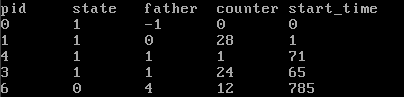


图58

结束调试。

**3.3 调试proc文件系统的工作过程**

①在fs/procfs.c文件中psread函数的第一个if语句处（第70行）添加一个断点。

②按F5启动调试，待Linux 0.11启动后，输入命令“cat /proc/psinfo”后按回车，会在刚刚添加的断点处中断

③按F10单步调试到第81行，这个过程中调用了五次addTitle函数将标题添加到了psbuffer缓冲区中。接下来第85行开始的for循环，会遍历进程控制块数组，将有效的进程信息写入psbuffer缓冲区中。

⑤在psread函数的最后一个for循环语句处（第102行）添加一个断点。按F5继续执行，程序就会命中此断点。此for循环的目的是将psbbuffer缓冲区内的数据逐字节的读取到用户空间中的buf缓冲区中。

⑥在psread函数最后的return语句处添加一个断点，按F5继续调试，程序会在此处中断。

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

int main(int argc, char\* argv[])

{

char buf[513] = {'\0'};

int nread;

int fd = open(argv[1], O\_RDONLY, 0);

while(nread = read(fd, buf, 512))

{

buf[nread] = '\0';

puts(buf);

}

return 0;

}

**3.4 简化fs/procfs.c文件中的源代码**

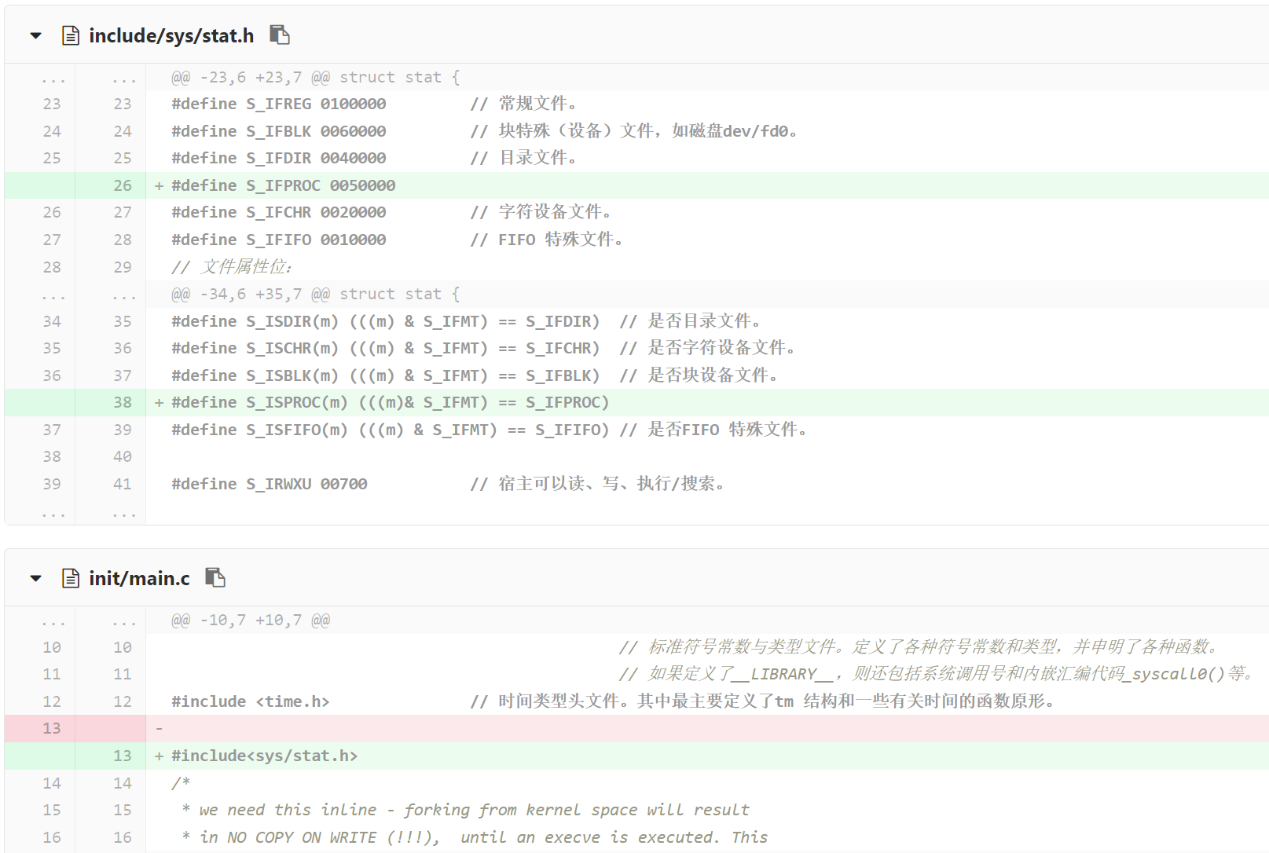
文件fs/procfs.c中psread函数的源代码调用了itoa和addTitle函数将内容写入psbuffer缓冲区，造成源代码比较复杂。读者可以使用sprintf函数替换掉itoa和addTitle函数，源代码就会简单很多。但是Linux 0.11没有实现sprintf函数，读者可以参考init/main.c文件中第200行的printf函数，在fs/procfs.c文件中实现一个sprintf函数。

**3.5 提交作业**

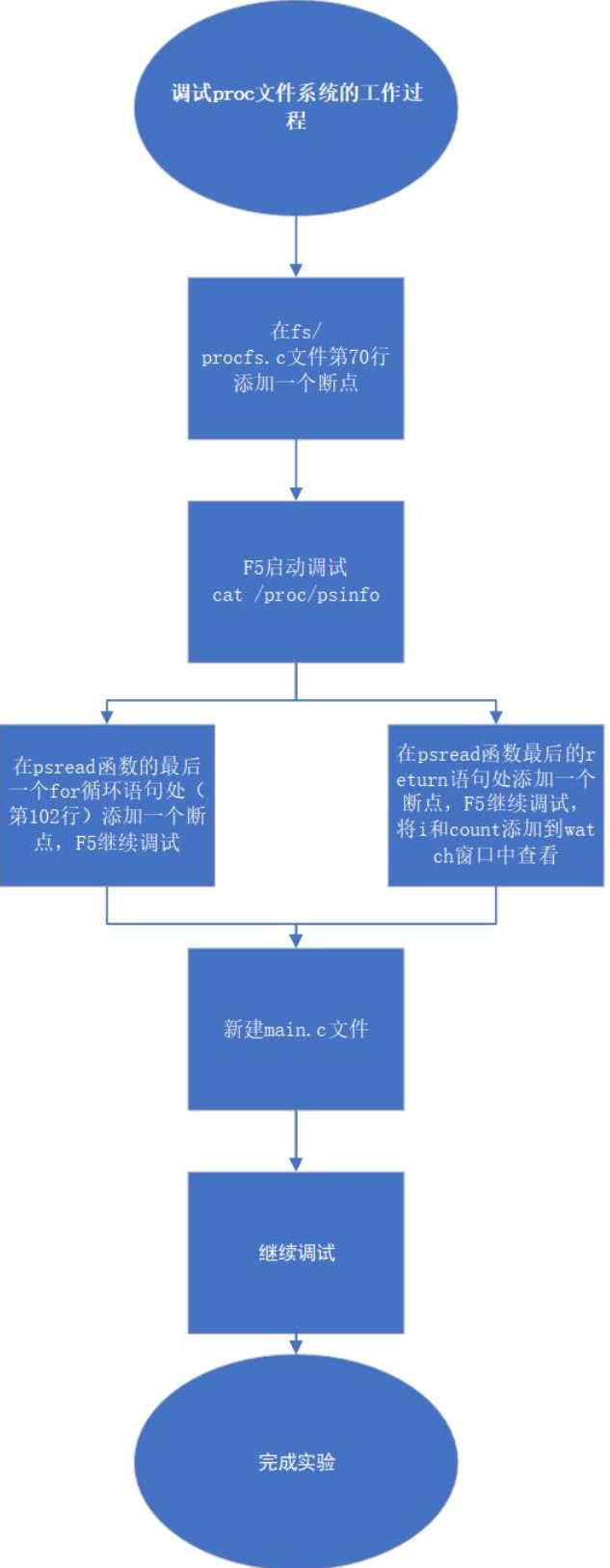
实验结束后先使用VSCode左侧的“源代码版本控制窗口”查看文件变更详情，确认无误后再将本地项目提交到平台的个人项目中。

### 4.运行结果

共修改个文件6个文件

图59

### 5.流程图

图60

### 6.实验体会

在这次的实验经历中，我深入探索了proc文件系统的运作原理，并增进了对文件、目录和索引节点等核心概念的理解。proc文件系统作为一个独特的存在，它并非基于物理磁盘，而是在系统内存中构建，为用户提供了一个实时访问系统状态信息的窗口，如进程详情、内存占用状况等。

在Linux环境下，proc文件系统展现为一种虚拟文件系统，其内部的文件与目录并非物理实体，而是由内核在需要时动态创建。这一特性使得proc文件系统能够高效地为用户提供实时的系统信息。

通过对proc文件系统的学习，我掌握了其在内存中如何模拟文件和目录的创建过程，以及如何通过标准的文件操作接口读取和更新这些“虚拟”文件的内容。同时，我也对文件、目录和索引节点这些基本文件系统元素有了更深入的认识。文件作为数据的载体，可以包含文本、图片、音频等多种形式；目录则是一种特殊的文件，用于组织和管理其他文件和目录；而索引节点则存储了文件的元数据信息，如访问权限、所有者以及文件大小等。

这次实验不仅让我对操作系统的文件系统部分有了更为全面的了解，还锻炼了我修改和扩展系统源码的能力。由于实验中的源代码部分打印指令尚未实现，我通过直接修改源码的方式添加了必要的指令，并在修改过程中特别注意了代码之间的逻辑关联和依赖关系。这一经验对于我未来进行操作系统相关的开发工作将大有裨益。

### 7.思考与练习

1. 请读者模仿psinfo节点的实现方法，实现一个保存物理内存信息的meminfo节点，该节点保存的数据可以参考实验八打印输出的物理内存的信息。

图61

2. 实现一个可以读取jiffies时钟滴答值的tickinfo节点。时钟滴答jiffies的说明可以参考实验六预备知识的内容。考虑到jiffies的值改变的太快了（10ms变一次），能够直接将Linux 0.11中全局变量jiffies的值转换为字符串并放入一个缓冲区中供用户读取吗？在将jiffies的值转换为字符串的过程中，其值是否有可能发生改变呢？如何解决这个问题。

按照以下步骤实现一个可以读取jiffies时钟滴答值的tickinfo节点：

①创建一个新的/sys文件系统的节点，命名为/sys/kernel/tickinfo

②在该节点下创建一个名为jiffies的文件，用于读取jiffies的值

③在jiffies文件中实现读取jiffies值的函数，该函数可以直接读取全局变量jiffies的值并将其转换为字符串，然后将其放入一个缓冲区中供用户读取

不能直接将其转换为字符串并放入缓冲区中供用户读取。因为在转换的过程中，jiffies的值可能会发生改变，导致读取的字符串不准确。

可以使用自旋锁来保护jiffies的值，确保在转换的过程中jiffies的值不会被改变。具体实现方法如下：

①在jiffies文件中使用自旋锁来保护jiffies的值，即在读取jiffies值之前先获取自旋锁，读取完毕后再释放自旋锁。

②在内核中更新jiffies值的地方也需要使用自旋锁来保护jiffies的值，以确保在更新jiffies值的同时不会被其他进程或中断干扰。

3. 在读取节点psinfo的过程中，函数psread会将所有进程的信息转换为字符串并放入psbuffer缓冲区中，但是在循环遍历所有64个进程的过程中，是否会出现进程的信息发生变化的情况呢？如果会发生变化，读者是否可以参考前一个练习解决jiffies的值变化太快的方法来解决此问题。

在循环遍历所有64个进程的过程中，可能会出现进程的信息发生变化的情况。例如，进程的状态、CPU时间片、内存使用情况等都可能在遍历过程中发生变化。

要避免这种情况，可以采用类似于解决jiffies值变化太快的方法，即在每次遍历时记录当前时间戳，并将其与上次记录的时间戳进行比较，只有当时间戳发生变化时才更新进程信息，否则直接使用上一次的缓存数据。这样可以避免在遍历过程中获取到过时的进程信息。

### 任务二十 MINIX 1.0文件系统的实现

### 1.实验目的

通过查看MINIX 1.0文件系统的硬盘信息，理解MINIX 1.0的硬盘管理方式。

学习MINIX 1.0文件系统的实现方法。

改进MINIX 1.0文件系统的实现方法，加深对MINIX 1.0文件系统的理解。

### 2.实验内容

编写Windows控制台程序，打印输出MINIX 1.0文件系统的目录树

### 3.实验步骤

**3.1准备实验**

使用浏览器登录平台领取本次实验对应的任务，从而在平台上创建个人项目（Windows控制台程序），然后使用VSCode将个人项目克隆到本地磁盘中并打开。

**3.2 打印输出MINIX 1.0文件系统的目录树**

①打开一个新的VSCode，并使用VSCode中“file”菜单中的“open folder”打开一个在前面的实验中克隆到本地的Linux011内核项目文件夹。

②在VScode“文件资源管理器”窗口中的任意一个文件夹或文件节点上点击右键，然后在弹出的快捷菜单中选择“Reveal in File Explorer”，在打开的文件夹中，将harddisk.img文件复制到C:\minix目录下（该路径是main函数中需要打开的harddisk.img文件的path字符串变量指定的磁盘位置）。

③再次打开在本实验中通过领取任务在本地创建的Windows控制台项目。

④生成项目，确保没有语法错误。

⑤打开本项目的文件夹，将文件夹下的minix.exe文件复制到C:\minix目录下。

⑥启动Windows控制台，进入C:\minix目录，然后执行命令“minix.exe > a.txt”。此命令会将应用程序打印输出的目录树重定向到文本文件a.txt中。

⑦打开a.txt文件查看MINIX 1.0文件系统的目录树。

3.3 提交作业

实验结束后先使用VSCode左侧的“源代码版本控制窗口”查看文件变更详情，确认无误后再将本地项目提交到平台的个人项目中。

### 4.运行结果

### 5.流程图

图62

### 6.实验体会

在深入探索MINIX 1.0文件系统的实验中，我主要研究了其硬盘管理方式及实现细节。MINIX 1.0文件系统以磁盘块为基础，每个块固定为1024字节，文件则通过这些块的有序组合来存储。

在学习的过程中，我专注于理解文件系统管理模块的核心作用，它负责处理文件的创建、删除、读写等关键操作。这一模块由文件系统缓存、文件系统操作、以及文件系统内存管理等子模块组成，它们共同协作，确保文件系统的稳定运行。

为了优化MINIX 1.0文件系统的性能，我尝试引入了多级索引结构的概念。这一改进不仅增强了文件系统的访问效率，也加深了我对MINIX 1.0文件系统内部机制的理解。通过实践，我认识到在改进文件系统时，必须综合考虑性能、可靠性和安全性等多方面的因素。

此外，我还通过makefile模拟了minix系统的构建过程，并实现了对逻辑块的增删查改等操作。尽管这些新增的内容目前对我来说还不够熟悉，但它们为我未来深入学习Linux系统和MINIX文件系统奠定了坚实的基础。我期待在后续的学习中，能够更深入地理解这些代码段，并将其运用到实际的系统开发中。

### 7.思考与练习

1. 参考load\_inode\_bitmap函数写出一个可以将硬盘镜像文件中的整个逻辑块位图都加载到内存中的函数。

2. 打印输出类似于Linux 0.11内核命令“df”的输出内容。

3. 将“/usr/root”文件夹下的“hello.c”文件的内容打印输出。

4. 将“/usr/src/linux-0.11.org”文件夹下的“memory.c”的内容打印输出。注意，此文件大于7KB，所以需要使用二次间接块才能访问所有的数据。

5. 删除“/usr/root/hello.c”文件。

6. 删除“/usr/root/shoe”文件夹。

7. 新建“/usr/root/dir”文件夹。

8. 新建“/usr/root/file.txt”文件，并设置初始大小为10KB。

参考load\_inode\_bitmap函数写出一个可以将硬盘镜像文件中的整个逻辑块位图都加载到内存中的函数

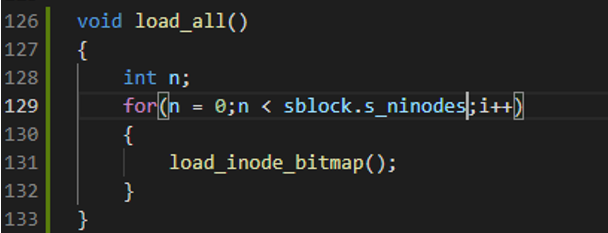


图63

打印输出类似于Linux 0.11内核命令“df”的输出内容。

将“/usr/root”文件夹下的“hello.c”文件的内容打印输出

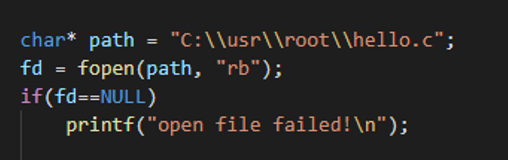


图64