**暨南大学本科实验报告专用纸**

课程名称 操作系统原理实验 成绩评定

实验项目名称 proc文件系统的实现 指导教师 郝振明

实验项目编号 0806015707 实验项目类型 验证 实验地点

学生姓名 张印琪 学号 2018051948

学院 信息科学技术学院 系 计算机科学系 专业 网络工程

实验时间 20XX 年 X 月 X日 上 午～ X 月 X 日 上 午

温度 ℃湿度

一、 实验目的

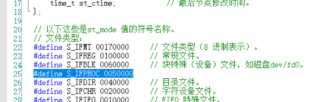
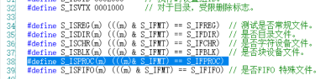
● 掌握proc文件系统的实现原理。 ● 掌握文件、目录、索引节点等概念。

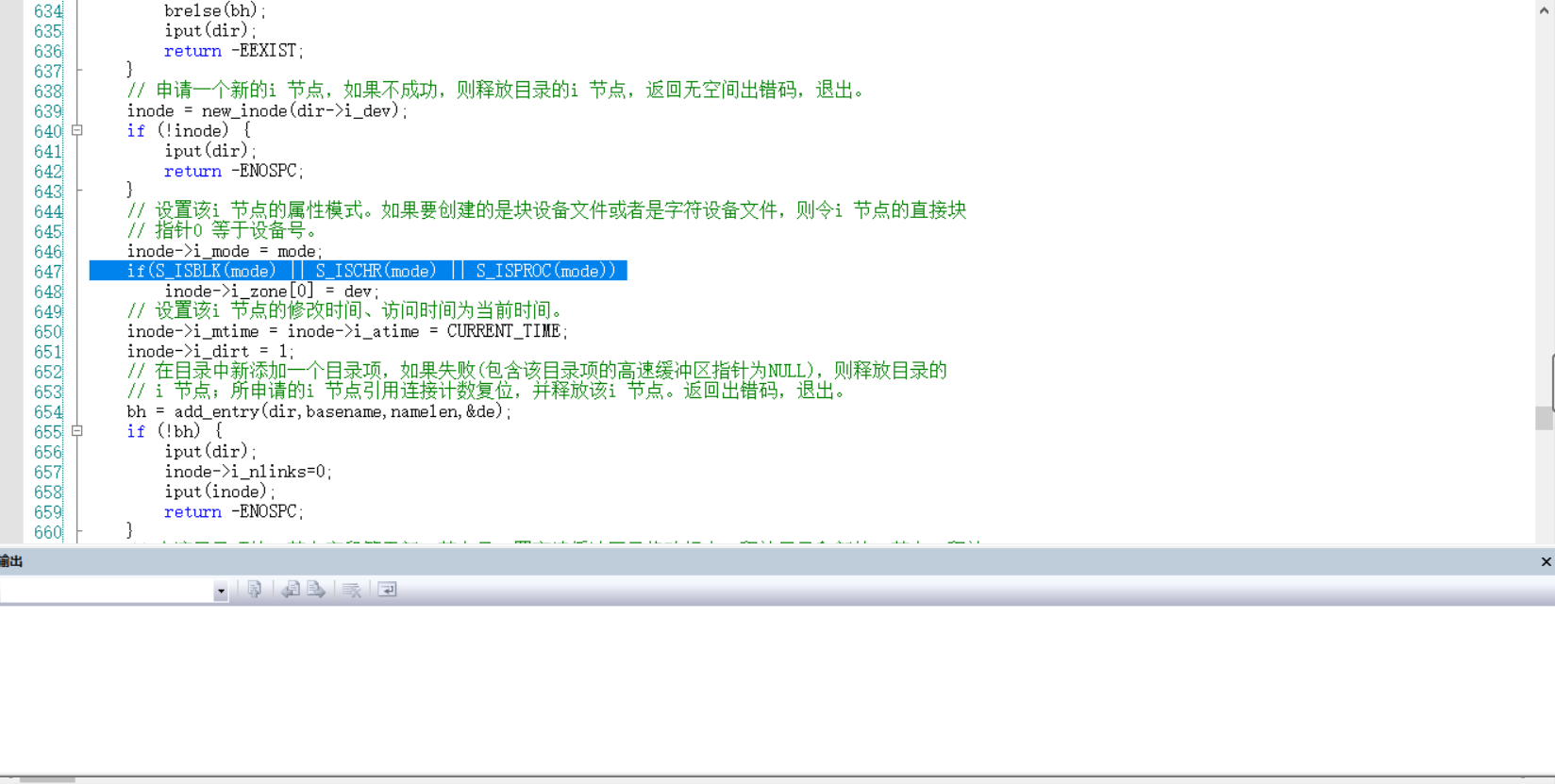
**三、 实验内容**

3.1 准备实验 1. 启动 Engintime Linux Lab。 2. 新建一个 Linux011 Kernel 项目。 3.2 实现 proc 文件系统 下面的内容会引导读者实现一个简单的 proc 文件系统，并在其中添加一个用于显示 Linux 0.11 操作系统进程信息的 psinfo 节点。每当进程信息发生变化时，Linux 0.11 操作 系统的内核负责向 psinfo 节点中写入相关的数据，然后 Linux 0.11 的应用程序就可以从 psinfo 节点中读取数据了。读取数据的方式与从普通文件中读取数据的方式完全相同。 请读者按照下面的步骤进行实验： 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

Linux 内核实验教程

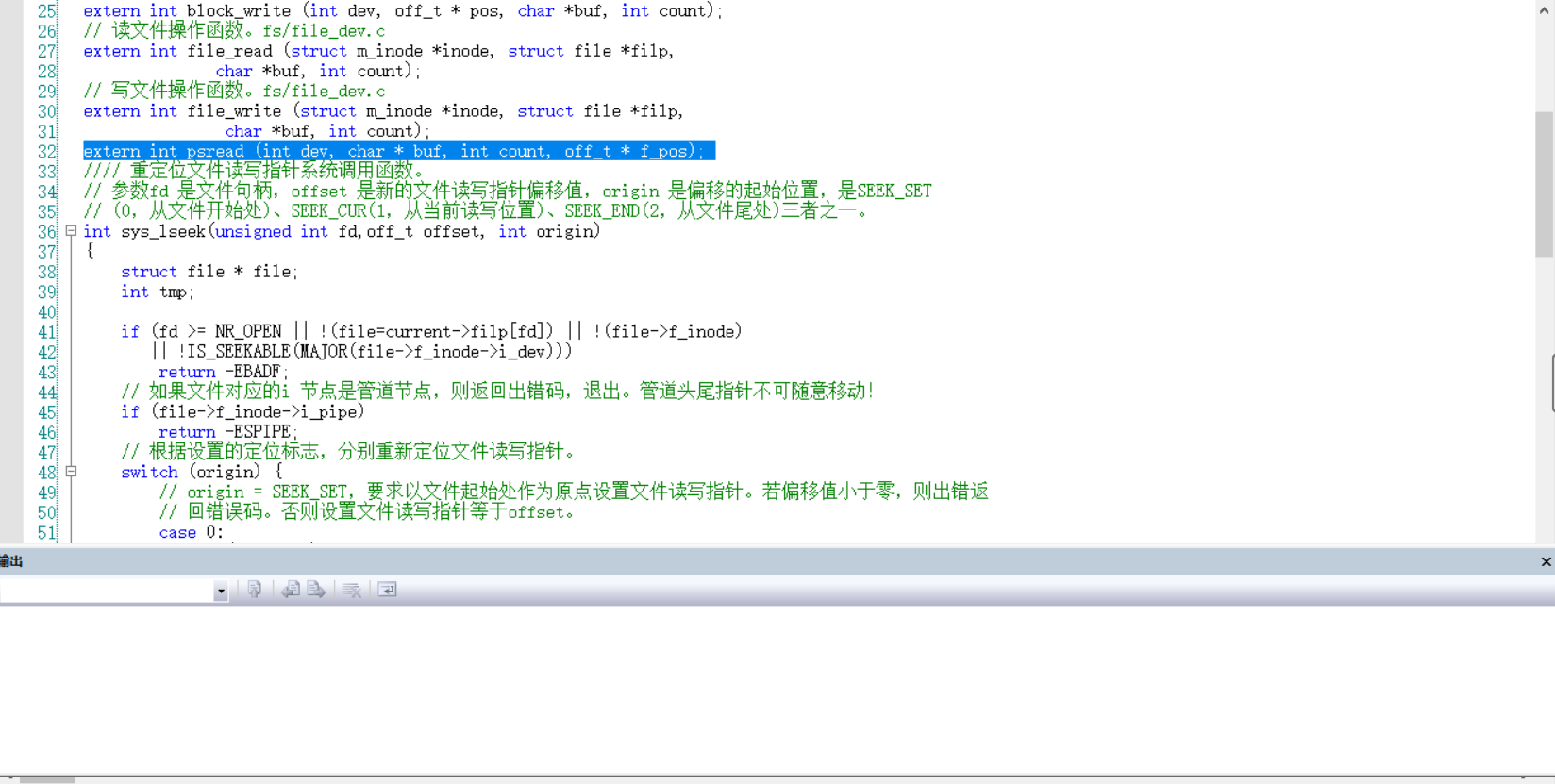
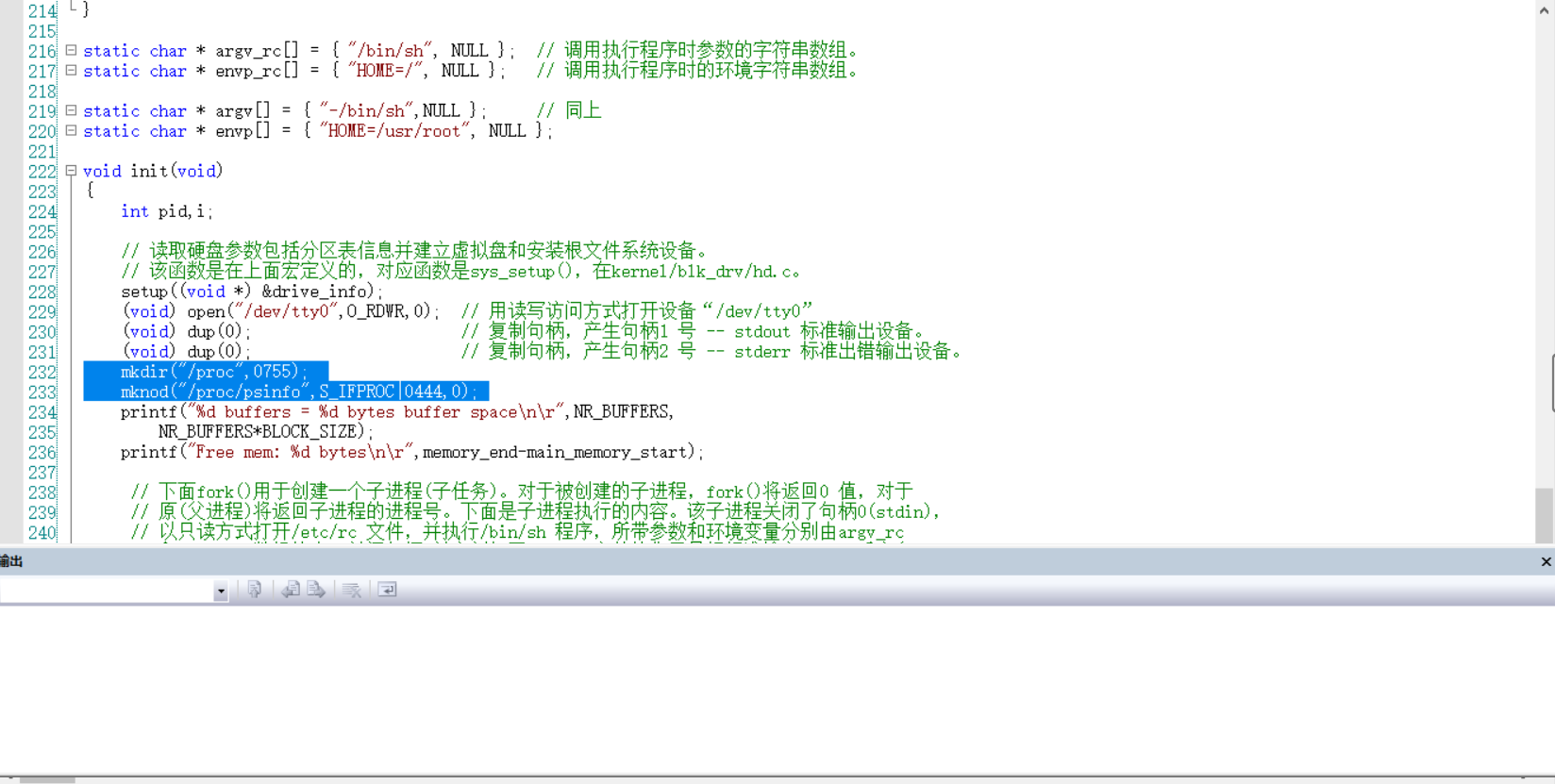
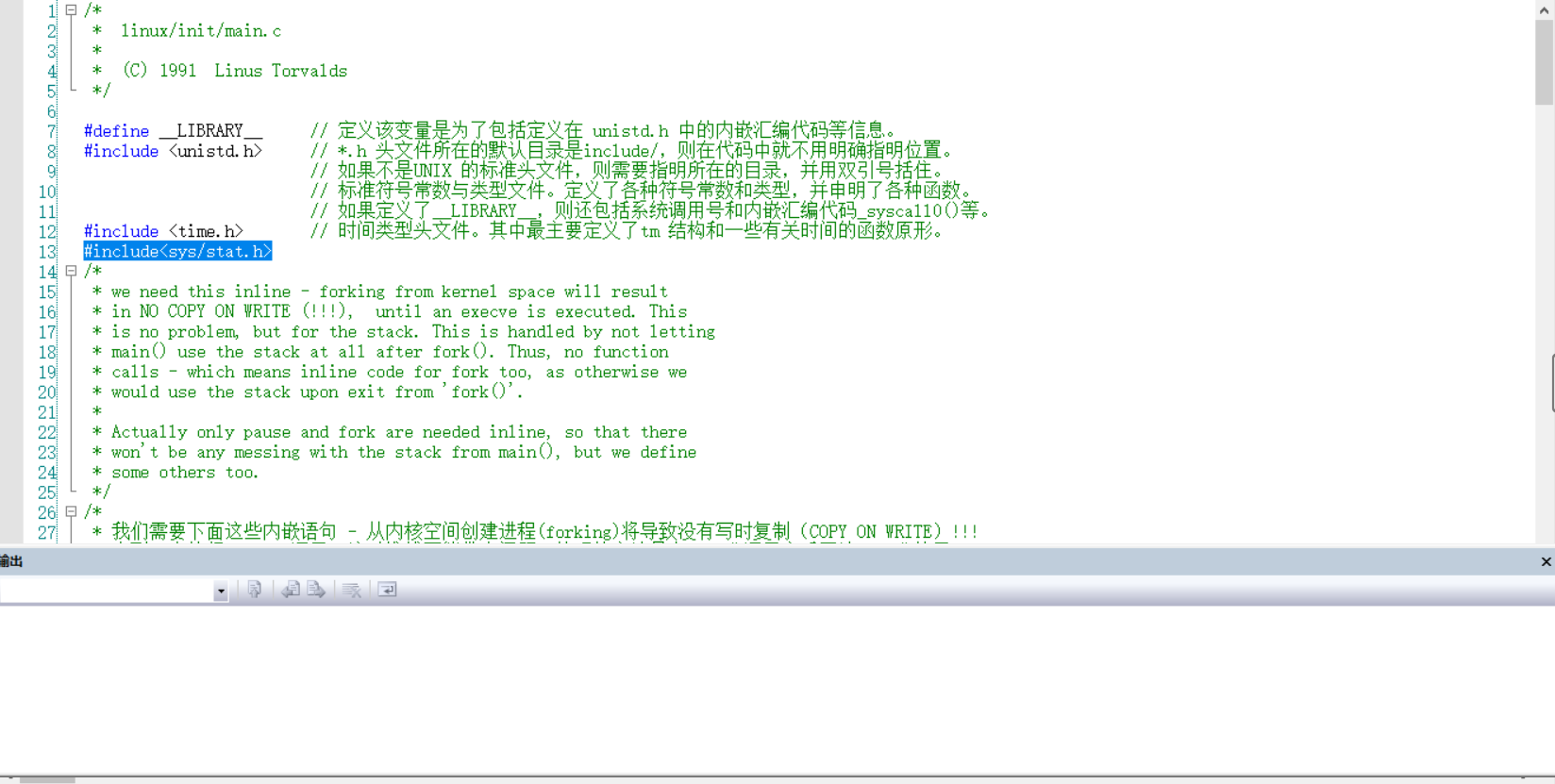
2. 为新文件类型添加相应的测试宏。在第 37 行后面添加测试宏，代码如下： #define S\_ISPROC(m) (((m)& S\_IFMT) == S\_IFPROC) 3. psinfo 结点要通过 mknod 系统调用建立，所以要让它支持新的文件类型。在 fs/namei.c 文件中只需修改 mknod 函数中的一行代码即可，即将第 647 行代码修 改为如下： if(S\_ISBLK(mode) || S\_ISCHR(mode) || S\_ISPROC(mode))

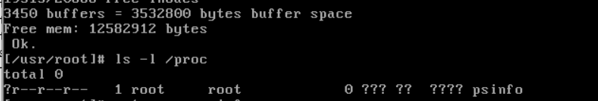
proc 文件系统的初始化工作应该在根文件系统被挂载之后开始，包括两个步骤：建立 /proc 目录和建立/proc 目录下的各个结点(本实验只建立/proc/psinfo 节点)。因为在根文 件系统加载后，初始化过程已经进入用户态，就不能调用 sys\_mkdir 和 sys\_mknod 函数建立 目录和节点了，而必须调用 mkdir 和 mknod 系统调用函数来建立目录和结点。所以，必须在 初始化代码所在文件中实现 mkdir 和 mknod 两个系统调用函数的用户态接口。请读者按下面 的步骤继续修改 Linux 0.11 内核的源代码： 1. 在 init/main.c 文件的第 13 行包含头文件 stat.h ： #include<sys/stat.h> 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) 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); 由于 proc 文件系统对用户态程序来说是只读的，所以将 mkdir 函数的第二个参数 mode 的值设为“0755”(rwxr-xr-x)，表示只允许 root 用户改写此目录，其它用 户只能进入和读取此目录。将 mknod 函数的第二个参数 mode 的值设为 “S\_IFPROC|0444”， 表示这是一个 proc 文件，权限为 0444（r--r--r--），对所有 用户只读。 mknod 函数的第三个参数 dev 用来说明结点所代表的设备编号。对于 proc 文件系 统来说，此编号可以完全自定义。proc 文件的处理函数可以通过这个编号决定文 件包含的是什么信息。例如，可以把0对应psinfo， 1对应meminfo， 2对应cpuinfo。 2. 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); 这样，当文件类型为 proc 文件时，就会调用 psread 函数进行处理了。传递给 psread 函数的参数包括：  inode->i\_zone[0]，这就是之前调用mknod函数时指定的dev ——设备编号。  buf，用户空间缓冲区，就是应用程序在调用 read 函数时传入的第二个参数， 用于存放从文件中读取到的数据。  count，就是应用程序在调用 read 函数时传入的第三个参数，用于说明 buf 指向的缓冲区的大小。  &file->f\_pos，f\_pos 是上一次读文件结束时“文件位置指针”的位置。这里 必须传递指针，因为 psread 函数还需要根据读取到的数据量修改 f\_pos 的值。

3. 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：

Linux 内核实验教程



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

psread 函数的源代码在“学生包”本实验对应文件夹下的 proc.c 文件中。请读者按照 下面的步骤将 proc.c 文件中的源代码添加到 Linux 0.11 的内核中： 1. 在Linux Lab的“项目管理器”窗口中，右键点击“fs”文件夹节点，选择菜单“添 加”中的“添加新文件”,打开“添加新文件”对话框。 2. 在“添加新文件”对话框中选中“C源文件（.c）”模板，输入文件名称“procfs.c” 后点击“添加”按钮，添加一个新源文件procfs.c。 3. 将“学生包”本实验对应文件夹下的proc.c文件拖动到Linux Lab中释放，即可打 开此文件。将其中的代码全部复制到刚刚新建的源文件procfs.c中。 4. 按F7键生成项目，确保没有语法错误。 5. 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：

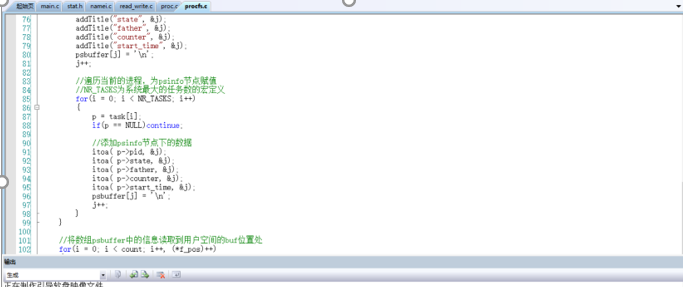


图 11-1 proc 文件夹内的文件属性信息 输入命令：cat /proc/psinfo 即可得到当前所有进程的信息，如下图所示：

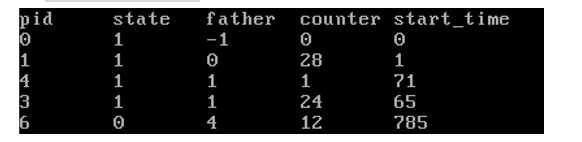
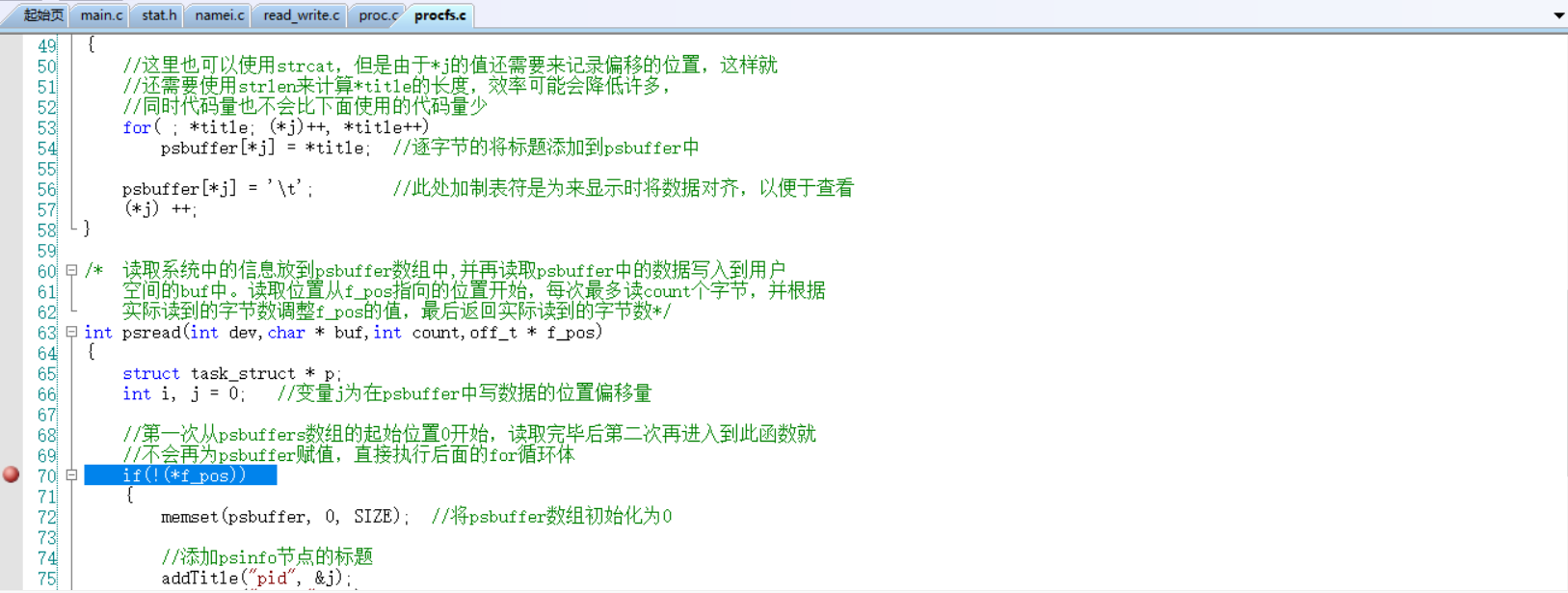
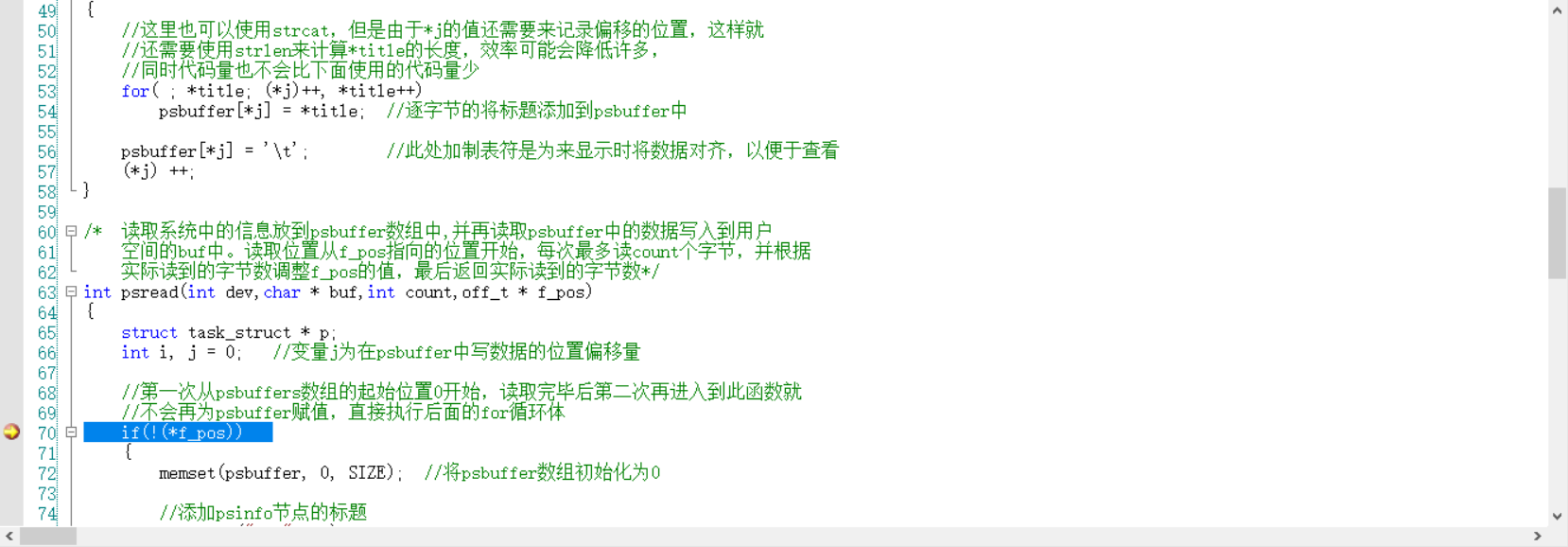


图 11-2 打印输出当前所有进程的信息 3.3 调试 proc 文件系统的工作过程 请读者按照下面的步骤调试 proc 文件系统的工作过程，进而理解相关的源代码： 1. 在 fs/procfs.c 文件中 psread 函数的第一个 if 语句处（第 70 行）添加一个断点。 2. 按 F5 启动调试，待 Linux 0.11 启动后，输入命令“cat /proc/psinfo”后按回 车，会在刚刚添加的断点处中断。此 if 语句的作用是：当第一次调用 psread 函数 时，文件位置指针 f\_pos 的值为 0，就会进入这个 if 语句将进程信息放入缓冲区 psbuffer 中。但是，如果需要再次进入 psread 函数时，由于文件位置指针 f\_pos 的值不为 0，就不再进入这个 if 语句，从而可以直接跳转到后面的 for 循环读取 数据了。 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。 4. 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。 5. 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。将鼠标移动到返回值变量 i 上可以查看此变量的值，再将鼠标移动到 psread 函数的第三个参数 count 上可以查看此变量的值，通过比较这两个值可以 发现，psread 函数实际读取的字节数小于 buf 缓冲区的大小 count，也就是说 psread 函数执行一次就可以读取到所有数据了。 

请读者考虑一下 cat 命令的实现过程（可以参考下面的源代码），虽然调用一次 read 函数就已经读取到了所有数据，但是由于 cat 实现代码中的 while 循环第一次读到的字节数 nread 不为 0，所以会再次执行 read 函数来读取文件，也就会再次进入 psread 函数。 #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; }



请读者按照下面的步骤继续调试： 1. 按 F5 继续调试，程序会在第 70 行的断点处再次中断。此时查看 Bochs 虚拟机的显 示窗口，已经打印输出 psinfo 节点的内容了。 2. 继续按 F10 单步调试，可以发现程序并没有进入第一个 if 语句，而且没有读取任 何数据就从第 111 行的 break 语句跳出了 for 循环，最后返回了 0。 3. 按 F5 继续运行，cat 命令就结束了，也就不会再命中任何断点了。 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。

总结：

通过本次实验我学会掌握了文件系统接口实现了proc文件系统，它是一个虚拟文件系统，在 Linux启动时就被挂接（mount）到了/proc目录上。proc通过虚拟文件和虚拟目录的方式提 供访问系统参数的机会，所以有人称它为“了解系统信息的一个窗口”。这些虚拟的文件和 目录并没有真实的存在于磁盘上，而是在内存中形成了一种对Linux内核数据的直观表示， 并且随着操作系统的运行自动建立、删除和更新。虽然是虚拟的，但它们都可以通过标准的 文件系统调用来访问（包括read、write函数等）。 Linux 0.11还没有实现虚拟文件系统，也就是还没有提供增加新文件系统类型的接口。 所以本实验只能在现有文件系统的基础上，通过打补丁的方式模拟一个proc文件系统。Linux 0.11使用的是MINIX 1.0文件系统，这是一种典型的基于i节点(inode)的文件系统，《Linux 内核完全注释》一书的第12章对它有详细描述。MINIX 1.0文件系统中的每个文件都要对应 至少一个inode，而 inode中记录着文件的各种属性，包括文件类型等。文件类型有普通文件、 目录、字符设备文件和块设备文件等。在Linux 0.11的内核中，每种类型的文件都有不同的 处理函数与之对应。所以，可以在MINIX 1.0文件系统中增加一种新的文件类型——proc文 件，并在相应的处理函数内实现proc文件系统的功能①了解了 Linux 系统调用的执行过程，对系统调用有了深刻的认识。②学会增加系统调用及添加内核函数的方法。③学会运用调用中的监控功能。系统调用是操作系统为应用程序提供的与内核进行交互的一组接口。通过这些接口，用户态应用程序的进程可以切换到内核态，由系统调用对应的内核函数代表该进程继续运行，从而可以访问操作系统维护的各种资源，实现应用程序与内核交互。