操作系统原理实习四

**【实习题目1】：**

请求分页系统中的 LRU 置换算法，设计一个带分页缓存的文件读写器。

**【题目分析、相关原理及设计思路】：**

**题目分析：**这道题目要求设计一个带分页缓存的文件读写器，并采用 LRU 置换算法来管理缓存，需要将文件划分为大小相等的页，并使用 LRU 算法来管理缓存中的页面，以提高文件的访问效率。

**相关原理：**

分页系统：分页系统是一种操作系统内存管理机制，将物理内存划分为大小相等的页框，而程序所需的内存也被划分为大小相等的页。当程序需要访问内存时，操作系统会将所需页调入内存中。因此，本题要求的带分页缓存的文件读写器，需要将文件划分为大小相等的页，并使用缓存来管理这些页。

LRU 置换算法：LRU 算法是一种页面置换算法，其基本思想是将最近最少使用的页面置换出去。

文件读写器：文件读写器是一个用于读写文件的程序，它可以从文件中读取数据并将数据写入文件。

**设计思路：**首先，需要读取文件并将其分页，并将文件中的每个数据分为一页，分页后，将页号存储，接下来，需要实现 LRU 算法来管理缓存中的页面，先存储缓存中的页面，然后可以查找某个页面是否在缓存中，如果该页面已经在缓存中，则需要将其从缓存中删除并重新插入到缓存的开头，否则，需要将缓存中最后一个页面删除，并将新页面插入到缓存的开头。这样就保证了缓存中最近访问的页面总是在开头。在 main 函数中，读取命令行参数，并调用上述两个函数来实现带分页缓存的文件读写器。最后输出缺页数即可。

【详细步骤和相应截图】：

**1、LRU算法实现函数；**

int lru(vector<int>& pages, int num\_frames) {

int page\_faults = 0;

vector<int> frames(num\_frames, -1);

for (int i = 0; i < pages.size(); i++) {

int page = pages[i];

vector<int>::iterator it = find(frames.begin(), frames.end(), page);

if (it != frames.end()) {

frames.erase(it);

}

else {

page\_faults++;

frames.pop\_back();

}

frames.insert(frames.begin(), page);

}

return page\_faults;

}

**2、主函数编写，读取文件并管理，输出缺页数；**

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

{

if (argc != 4)

{

cout << "Usage: ./lru <num\_phys\_pages> <page\_size> <filename>\n";

return 1;

}

int num\_phys\_pages = atoi(argv[1]);

int page\_size = atoi(argv[2]);

const char\* filename = argv[3];

ifstream infile(filename);

vector<int> pages;

if (infile.is\_open())

{

string line;

while (getline(infile, line)) {

int page\_num = atoi(line.c\_str()) / page\_size;

pages.push\_back(page\_num);

}

}

else

{

cout << "Unable to open file.\n";

return 1;

}

infile.close();

int page\_faults = lru(pages, num\_phys\_pages);

cout << "Number of physical pages = " << num\_phys\_pages << ", page size = " << page\_size << "\n";

cout << "Reading memory trace from " << filename << "... Read " << pages.size() << " memory references\n";

cout << "Results: " << page\_faults << " page faults\n";

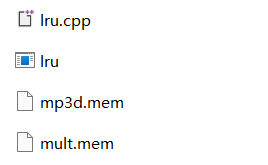
return 0;

}

**3、在Windows PowerShell中将 lru.cpp 文件编译成一个名为 lru 的可执行文件,然后使用mp3d.mem和mult.mem两文件测试文件读写器。**

文本

描述已自动生成

****

【实习中遇到的困难及解决方法、实习心得或课程建议】：

**困难：**如何具体实现LRU算法。

**解决办法：**可以使用一个 vector 来存储缓存中的页面，该 vector 的大小应该等于系统中可用的物理页面数，然后使用 find 函数在 vector 中查找某个页面是否在缓存中。如果该页面已经在缓存中，则需要将其从 vector 中删除并重新插入到 vector 的开头，否则，需要将缓存中最后一个页面删除，并将新页面插入到 vector 的开头。这样就保证了缓存中最近访问的页面总是在 vector 的开头。

**实习心得：**通过该实习，动手操作了 LRU置换算法的实现以及如何实现一个带分页缓存的文件读写器，让我我深入了解了计算机操作系统的分页机制和页面置换算法的工作原理：最近最少使用的页面予以淘汰，通过编写代码实现了算法功能，同时使用可执行文件导入文件的方式验证了算法的正确性。

【实习题目2】：

开发一个能够进行几何计算的虚拟设备驱动。

【题目分析、相关原理及设计思路】：

**题目分析：**需要在Linux系统下实现一个可以进行几何计算的字符驱动设备，即编写字符驱动代码中的函数实现，再将其加载到内核中，同时创建设备驱动文件，就能在应用层使用该字符虚拟设备了。

**相关原理：**

注册字符设备：在Linux内核中，所有的设备都是通过主设备号和次设备号进行管理的。字符设备的主设备号一般由内核分配，开发者需要在驱动程序中实现字符设备的注册函数，以便在系统启动时注册字符设备并获取主设备号。

实现文件操作函数：在字符设备被打开、关闭、读取、写入等操作时，内核会调用相应的文件操作函数。开发者需要实现这些函数，以便对字符设备进行相应的操作。

实现字符设备的初始化和清理函数：开发者还需要实现字符设备的初始化函数和清理函数。初始化函数在设备被加载时调用，用于初始化设备的状态；清理函数在设备被卸载时调用，用于清理设备的状态。

注册字符驱动设备：开发者需要实现字符驱动设备的注册函数，将字符设备的文件操作函数和初始化、清理函数与字符驱动设备关联起来。

编译并安装驱动程序：最后，开发者需要编译驱动程序，并将其安装到Linux内核中，以便在系统启动时加载该驱动程序。

Makefile是一种文本文件，主要用于描述一个软件项目中各个源文件之间的依赖关系，以及如何编译和链接这些源文件来生成可执行文件或库文件等程序。

**设计思路：**首先编写驱动代码，在驱动代码的read和write函数中实现几何信息读入和计算，然后编写Makefile进行自动化编译和构建驱动程序，编写完成后编译和加载驱动，然后为驱动创建设备文件，最后编写应用程序测试驱动是否能够进行正确的几何运算。

【详细步骤和相应截图】：

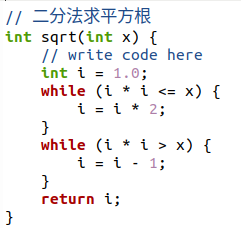
**1、创建和编写驱动代码ly\_dev.c和Makefile文件；**

**文本

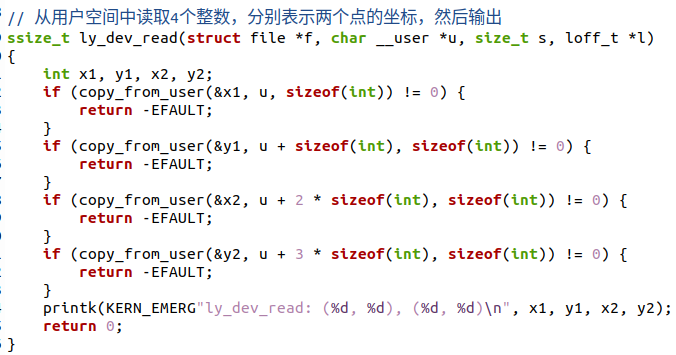
描述已自动生成**

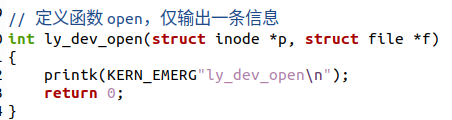
**ly\_dev.c具体代码：**

****

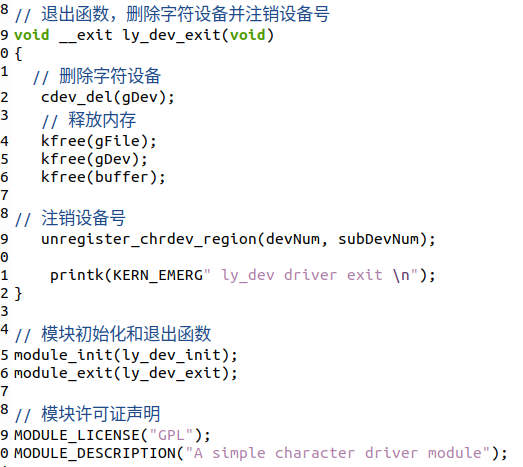
****

****

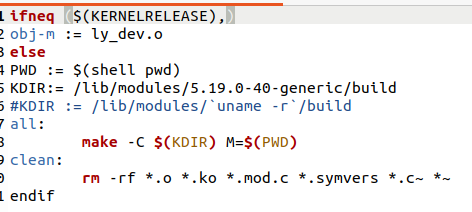
****

****

****

****

**2、编写Makefile文件：**

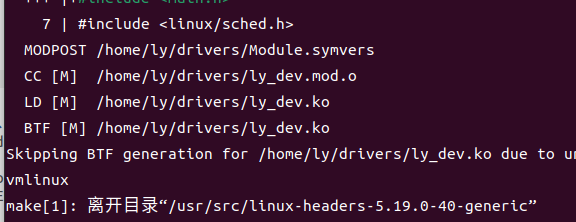
****

**3、编译并加载驱动；**

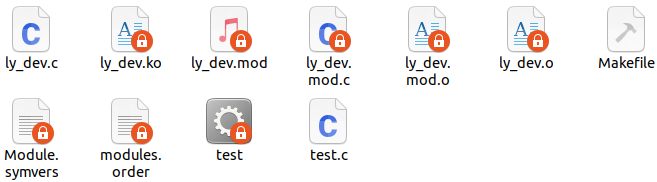
**先安装所需依赖包；**



**在dev和Makefile文件目录下执行make；**



**自动编译出各项所需文件：**

****

**4、将驱动加载到内核：**

手机屏幕截图

描述已自动生成

**使用dmesg查看驱动程序加载信息：**

**文本

描述已自动生成**

**5、编写一个应用层的程序对ly\_dev驱动进行测试；**

**test.c代码：**

**文本

描述已自动生成**

**6、将test.c编译成可执行文件并运行；**

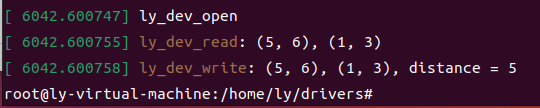
文本

描述已自动生成

**图形用户界面, 文本, 应用程序, 聊天或短信, Teams

描述已自动生成**

**7、再使用dmesg命令查看驱动程序运行结果，成功得到点之间的距离；**

****

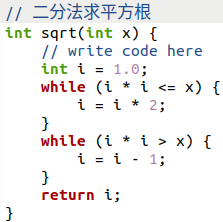
【实习中遇到的困难及解决方法、实习心得或课程建议】：

**困难1：**在ly\_dev.c编写时，计算两点距离需要使用平方根函数，但即使包含math.h头文件，在编译时仍会报错；

文本

描述已自动生成

**解决方法：**查了各种资料，发现需要在代码编译时对指令进行更改但由于这里使用了Makefile文件来自动编译，操作不方便，故直接使用二分法自己写了一个求平方根的函数，同时由于对Linux系统对浮点数的特性以及字节编码不熟悉，这里直接使用了int来保存距离，方便后续测试：



**困难2：**由于对ly\_dev.c代码进行了多次修改和重编译，修改重编译后再将驱动加载到内核中时报错；



**解决方法：**根据报错提示文件已经存在，得知是上次加载的驱动没有卸载，需要使用rmmod指令对上一次加载的 驱动进行卸载，再重新把更改好的驱动加载到内核即可。

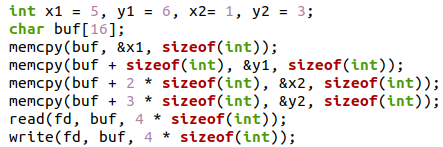
**困难3：**在测试代码中传入时使用：

int x1 = 5, y1 = 6, x2= 1, y2 = 3;

int ret1 = read(fd, &x1, sizeof(int));

但输出顺序变成：[ 5491.033059] ly\_dev\_read: (5, 3), (6, 1)

**解决方法：**这是因为用该方法在调用 read() 函数时，读取的数据先存储到了 &x1 的内存地址中，再根据数据类型的大小按顺序依次存储到下一个内存地址中，因此输出的顺序发生了变化，&x1对应的内存地址为 0x7fffXXXXX001，而 &y1对应的内存地址为 0x7fffXXXXX005，因为sizeof(int)的大小为4，所以读取的数据先存储到 0x7fffXXXXX001中，再存储到 0x7fffXXXXX002、0x7fffXXXXX003、0x7fffXXXXX004 中，最后存储到 0x7fffXXXXX005 中，因此输出的顺序为 (5, 3), (6, 1)，因此这种方法会导致顺序改变，因此在测试中直接将数据先存储到一个buf数组中，用buf作为参数调用read和write（如图），这样操作就能防止顺序发生错位；



**实习心得：**通过这个题目，我掌握了Linux系统下开发字符驱动设备的方法，熟悉了字符设备驱动程序的编写，包括驱动初始化、设备注册、文件操作、设备控制等函数的实现，知道了字符驱动设备中每个函数的调用时机，由于能力和时间限制只在实习中实现了读取两个点信息并计算距离，后续的学习中会增加更多功能，同时在开发中遇到了各种各样的问题，除了我列出的问题以外还遇到了加载驱动提示内核污染的问题以及设备号资源占用等问题，但是都使用了较为暴力的方式解决，在后续学习中也会深入了解这些问题的原理和具体解决办法。