**福州大学数学与计算机科学学院**

课程实验报告

**课程名称： 计算机操作系统**

**学 号： 031402207**

**姓 名： 陈奕程**

**专 业： 计算机科学与技术**

**年 级： 2014级**

**学 期： 2014学年 第2学期**

2015年 7 月 5日

**实验一 Linux操作系统的使用和分析**

**一、实验目的**

本实验主要学习和掌握Linux操作系统的基本应用。通过本实验，学生能够熟练掌握Linux环境下各种基本操作命令接口的应用。从系统安全角度出发，学习掌握系统的基本安全优化和配置，在操作系统层次进行有效安全加固，提高Linux系统的安全性能。通过本次实验，有助于学生进一步理解操作系统原理中的相关内容，加深认识。

**二、实验要求**

1、熟练掌握Linux系统的基本操作命令。

2、熟悉Linux 系统的基本配置。

3、实现Linux系统的安全加固。

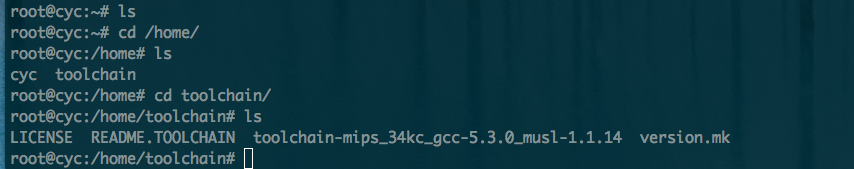
4、掌握一种以上的网络应用软件的安装、配置与应用。

**三、实验内容**

**1、Linux基本操作与管理**

* + - 1. **文件管理及磁盘管理**

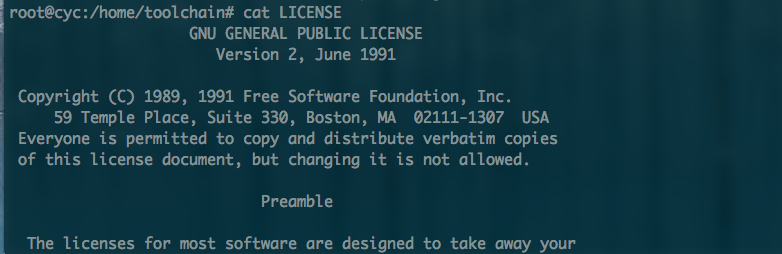
1. ls: 这个命令能列出你当前工作目录下的文件名与目录名，默认只显示非隐藏文件名。



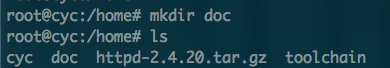
1. wget 参数+链接：从ftp下载



1. cat: 显示文本文件内容,具体用法：cat 文件名称。Cat命令是只查看但不能修改文件内容。



1. cd：此命令用来进出目录的，可以改变你的工作目录,具体用法：cd 目录名称。
2. mkdir,rmdir：分别是用来建立新目录和删除已建的目录。



1. pwd：显示当前所在的目录位置。



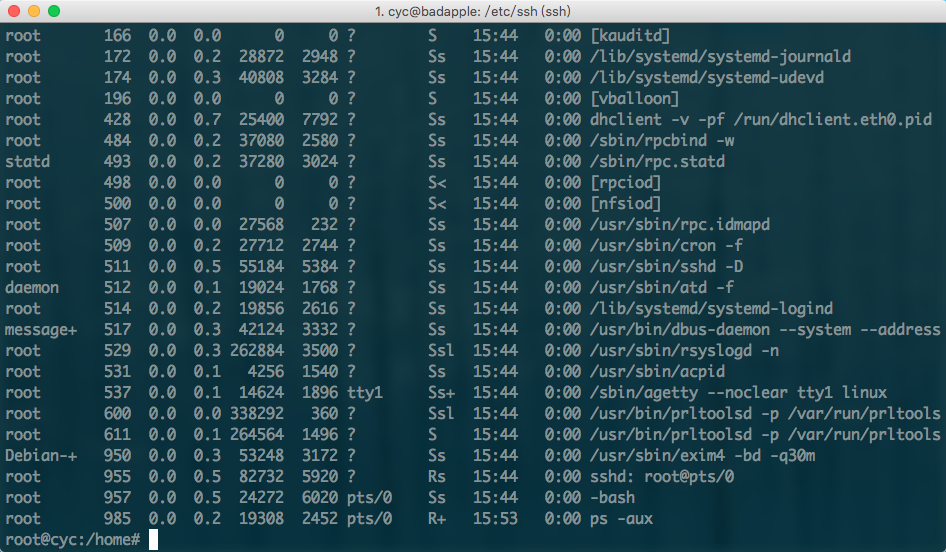
1. cp：复制文件 具体用法是：cp –r 源文件(source) 目的文件(target)

参数r是指连同源文件中的子目录一同拷贝。

1. rm：删除文件或者文件夹。使用格式：rm filename 或者rm -r 目录。
2. mv：文件更名或者搬移。
3. df：显示磁盘的使用。使用格式：df



1. ps -aux：查询正在执行的进程。

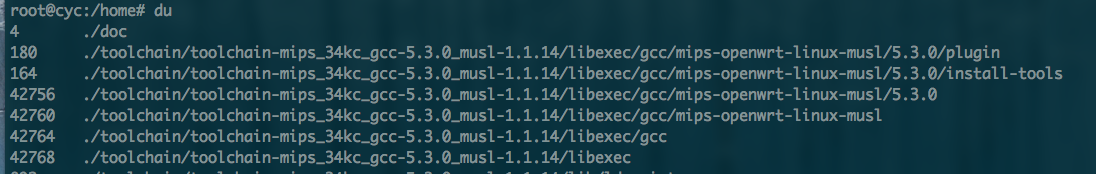


1. kill：终止正在执行的进程。使用格式：kill 进程号
2. find:搜寻文件或文件的路径

* find /-name filel 查找文件，但速度较慢。
* find /root/-name filel 指定了相对小的范围查找，提高了速度。
* find /-name filel -print 查找filel的路径。

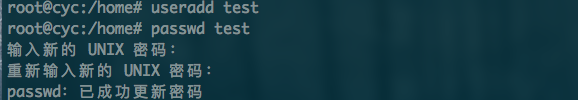


1. du：显示目录的使用。使用格式：du



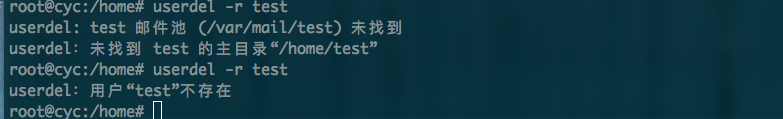
* + - 1. **用户管理**

1. # reboot 需要超级用户的权限，#超级用户提示符 $普通用户提示符。
2. Su root 进入超级用户
3. useradd：新增使用者账号的命令，如果想新增一个叫做test的用户，命令为useradd test。
4. passwd：该命令修改特定用户的口令，使用格式：passwd 用户名



1. 删除用户：

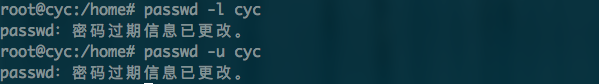
方法一：userdel test ，这种方法删除了用户但是文件夹还在。

方法二：userdel -r test，这种方法删除了用户以及用户的主目录。

1. 锁定不必要的账户

使用命令passwd -l <用户名>锁定不必要的账户

使用命令passwd-u <用户名>解锁需要恢复的账号。



1. 禁用root之外的超级用户
2. 使用命令passwd -l <用户名>锁定不必要的超级账户
3. 使用命令passwd-u <用户名>解锁需要恢复的超级账号。

风险：需要与管理员确认此超级用户的用途。

* + - 1. **系统管理**

1. 关机命令：shutdown –h now
2. 重启 reboot
3. **掌握一种以上的网络应用软件的安装、配置与应用。**

**apache的安装、配置与应用**

1. 安装依赖包pcre

apt-get install libpcre3 libpcre3-dev

1. 编译安装依赖包apr

wget http://archive.apache.org/dist/apr/apr-1.5.1.tar.gz

tar -zxvf apr-1.5.1.tar.gz

cd apr-1.5.1/

./configure --prefix=/home/cyc/apr/

make

make install

1. 安装依赖包apr-util

cd ..

wget <http://archive.apache.org/dist/apr/apr-util-1.5.4.tar.gz>

tar -zxvf apr-util-1.5.4.tar.gz

cd apr-util-1.5.4/

./configure --prefix=/home/cyc/apr-util \

--with-apr=/home/cyc/apr/

make

make install

1. 安装apache

wget http://mirror.bit.edu.cn/apache//httpd/httpd-2.4.20.tar.gz

tar zxvf httpd-2.4.20.tar.gz //解压

cd cd httpd-2.4.20/

./configure -h

./configure --prefix=/home/cyc/apr-util \

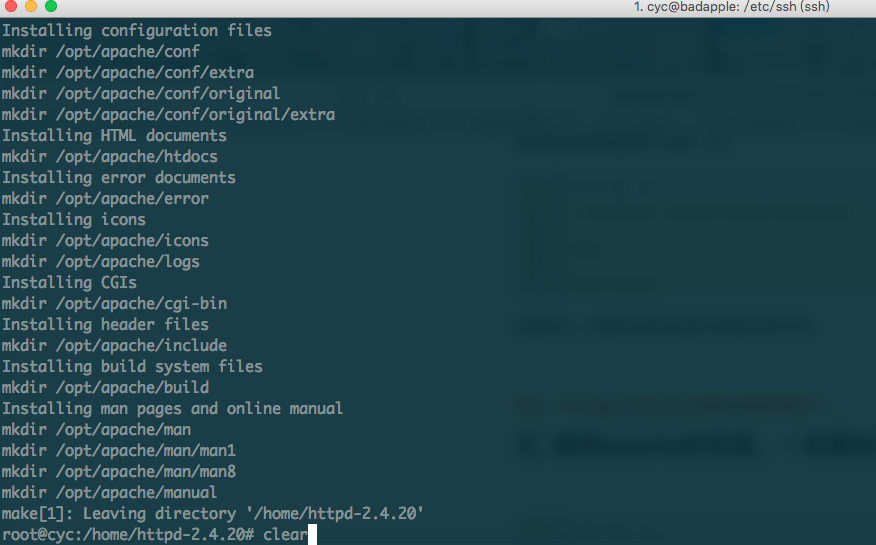
--with-apr=/home/cyc/apr/ \

--with-apr-util=/home/cyc/apr-util

// 配置编译参数

make //编译

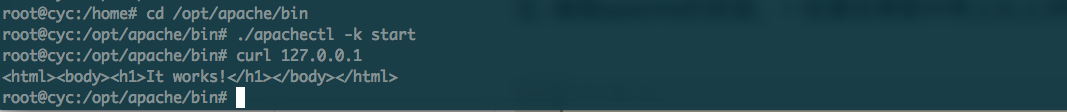
make install

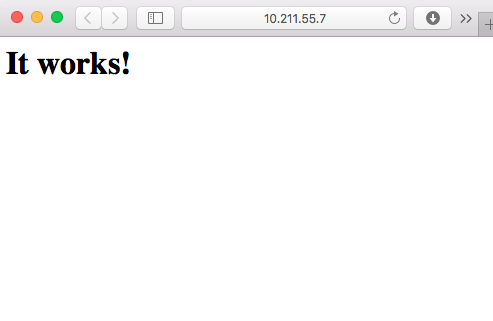


(5)运行

cd /opt/apache/bin

./apachectl -k start





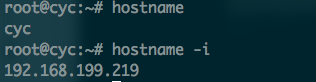
**2、熟悉Linux 系统的基本配置。**

(1)修改机器名

暂时修改机器名：hostname 主机名  
 注：hostname命令可以临时修改机器名但机器重新启动之后就会恢复原来的

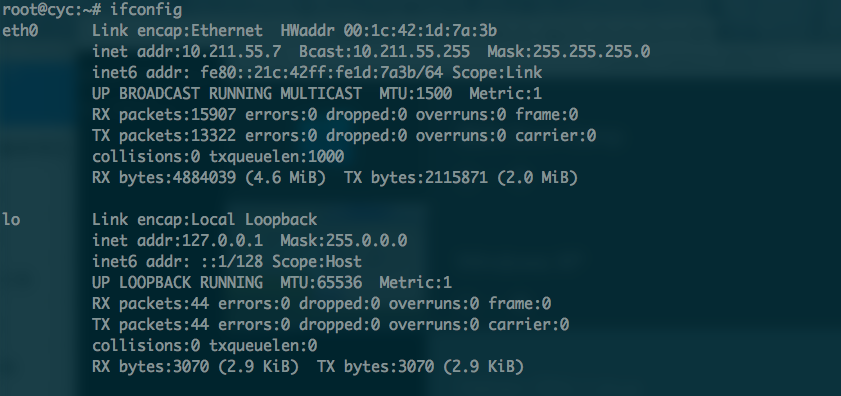
值。  
 #hostname //查看机器名

#hostname -i //查看本机器名对应的ip地址





1. ifconfig   显示或设置网络设备



**四、实验总结**

实验一的内容是比较简单的，由于平时使用的是类Linux的Unix的MacOSX系统，使用上较为顺手。过程中我使用ssh连接进行Linux实验以得到较好的操作体验。在编译安装Apache的过程中我遇到了缺少依赖包的问题，需要我们一次去安装一个个缺失的依赖包与配置编译参数。这不像windows安装软件大部分时候只需要使用安装包一路下一步那么直接。需要一定的耐心和善用-h查看帮助指令，更需要一定的专研。熟能生巧，相信通过这次操作会大大加大我对Linux的熟悉程度。实验过程中，我们通过合作和讨论，熟练掌握了各项Linux常用命令及网络配置与软件安装的技巧，受益颇多，为之后的各项试验打下了坚固的基础。

**参考文献**

1. 鸟哥的Linux私房菜 （基础学习篇 第三版）鸟哥 著；王世江 编
2. C Primer Plus（第5版 中文版）[美] 普拉塔（Prata S.） 著；云巅工作室 编
3. 刘晓明； 黄磊； 成新选；用U盘安装428XL Linux系统

4. 商周,米守防,苏飞: Linux网络驱动程序分析

5.C程序设计（第三版） 谭浩强 著

**实验二 Linux系统进程通信**

**一、实验目的**

1）了解有关Linux系统调用；

2）学习有关Linux的进程创建，理解进程创建后两个并发进程的执行。

**二、实验内容**

在Linux环境下，用C/C++语言编程，使用系统调用fork创建进程多个子进程。

**(1) 调试并完成下列程序，完成实验要求：**

#include "stdio.h"

#include "sys/types.h"

#include "unistd.h"

int main()

{

pid\_t pid1;

pid\_t pid2;

pid1 = fork();

pid2 = fork();

printf("pid1:%d, pid2:%d\n", pid1, pid2);

}

要求：

1. 请说出执行这个程序后，将一共运行几个进程。
2. 观察运行结果，并给出分析与解释。

答：

A:

执行后有四个进程

B:

理论上来说 有24种结果

设F(x)={

if(x==0) return 0;

else return 1;

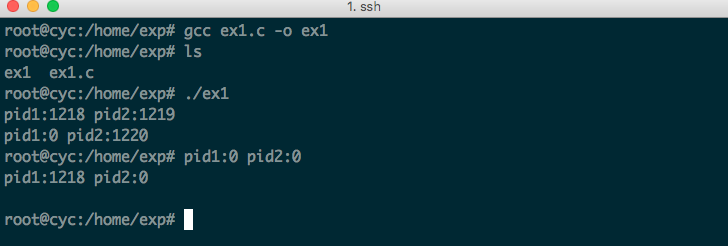
}

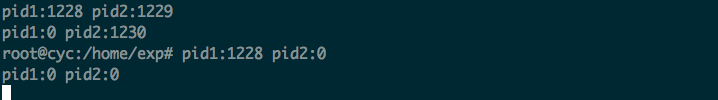
则F（pid1)有两种状态 F(pid2)也有两种状态

所以可将状态编码为二进制数 0 ，1，2，3

由于执行顺序的不确定性 所以结果有4！=24种

以下是程序部分执行结果





分析：

(1)一个进程调用fork()函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

(2)fork调用的一个奇妙之处就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：

1、在父进程中，fork返回新创建子进程的进程ID；

2、在子进程中，fork返回0；

3、如果出现错误，fork返回一个负值。

(3)我们可以根据返回值的不同来判断先执行父进程还是子进程。

(4)由结果可以得出执行进程的结果

程序运行过程：

原进程（进程1）

第1步pid1=1218 第2步pid2=1219

父进程（进程2）

ID为3539

父进程（进程1）

第3步pid2=1220

子进程

**(2)参考下面的相关程序实例，编写一个管道实验程序，实现两个进程之间的数据通信。**要求：父进程顺序写入若干个字符串，子进程顺序读出内容，并写入文件piple.txt，并显示出来。

（写出设计步骤、数据结构、关键代码、实验结果及其分析说明）

主要代码：

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

char msg[]="abcdefghijklmnopqrstuvwxyz";

int main()

{

int chan[2];

int pid;

char buf[100];

if (pipe(chan)==-1) return -1; //初始化管道

pid = fork();

if (pid!=0) //父进程

{

close(chan[0]);//关闭读取

write(chan[1],msg,sizeof(msg));

printf("parent msg sent\n");

}else{//新进程

close(chan[1]);//关闭写入

read(chan[0],buf,sizeof(buf));//将收到数据放入缓存

printf("child msg recevied\n");

//写入文件

FILE \*fp=fopen("pipo.txt","w");

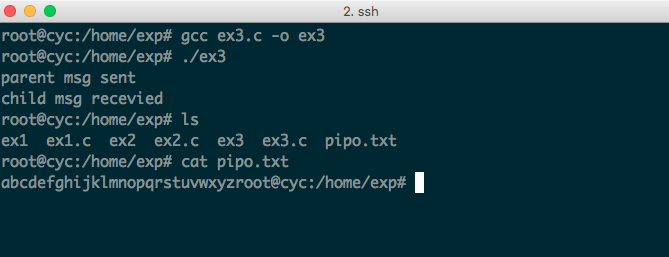
fprintf(fp,"%s",buf);

}

return 0;

}

运行截图：



**分析：**

无名管道是Linux中管道通信的一种原始方法具有以下特点：

1、它只能用于具有亲缘关系的进程之间的通信（也就是父子进程或者兄弟进程之间）；

2、它是一个半双工的通信模式，具有固定的读端和写端；

3、管道也可以看成是一种特殊的文件，对于它的读写也可以使用普通的 read()、write()等函数。但它不是普通的文件，并不属于其他任何文件系统并且只存在于内存中。

用 pipe() 创建的管道两端处于同一个进程中，由于管道主要是用于在不同的进程间通信的，因此，在实际应用中没有太大意义。实际上，通常先是创建一个管道，再调用fork()函数创建一个子进程，该子进程会继承父进程所创建的管道。此时父子进程分别拥有自己的读写通道，实现父子进程之间的读写，只需把无关的读端或写端的文件描述符关闭即可。

**三、实验总结**

通过本次实验，我初步理解了多进程程序设计及Linux进程通信的基本原理，熟练掌握了fork（）及管道传输的使用。一个进程调用fork函数后系统会把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。管道是从一端写数据一端读出数据，从而实现两个进程之间的通信。

同时在本次实验中熟练掌握了在Linux下编写及编译运行C语言程序的方法，对我的学习有着深刻的启发。今后我将更加深入学习Linux系统下的程序设计。

**参考文献**

[1] Linux中fork（）函数详解

http://blog.csdn.net/jason314/article/details/5640969

[2] Linux进程间通信(二)---管道通信之无名管道及其基础实验

http://blog.csdn.net/mybelief321/article/details/9073895

[3] 鸟哥的Linux私房菜 （基础学习篇 第三版）鸟哥 著；王世江 编

[4] C Primer Plus（第5版 中文版）[美] 普拉塔（Prata S.） 著；云巅工作室 编

**实验三 动态分区管理方法**

**一、实验目的**

本实验主要对操作系统中内存管理方法及其应用中的一些关键算法进行模拟。学生通过设计与实现相关算法，能够加强对相应理论的理解，并对了解操作系统内存管理子系统的相关处理原理与过程。

**二、实验要求**

基本要求：编码实现动态分区管理的三种分配算法和回收算法（相邻空闲区要合并）。

1）初始化：输入若干个空闲分区和已占用分区信息，建立空闲区表和已分配表。

2）分配功能：输入一个作业请求，可选最先、最佳和最坏算法实现该作业的内存分配；

3）回收功能：输入进入完成态的作业，回收其占用的分区，并进行检查合并相邻空闲分区。

4）显示当前分区情况：显示已占用分区、空闲区表的内容。

**三、实验内容**

1. **基本原理**

**最先适应算法**：

从用户区的低地址开始查找，找到能满足程序需求的第一个空闲去用于分配。如果该空闲空间与所需空间大小一样，则从空闲表中取消该项；如果还有剩余，则余下的部分仍留在空闲表中，但应修改分区大小和分区始址。

**最佳适应算法：**

查找一个能满足程序需求的，长度最短的空闲分区用于分配。

**最坏适应算法：**

查找一个能满足程序需求的，长度最大的空闲分区用于分配。

**内存回收：**

在一个进程被撤销时，系统需要回收其所占的分区。在回收一个分区时，需要判断是否有相邻的空闲区，以便把相邻的空闲区合并成一个搭的分区，有利于程序装入新的分配。

1. **算法流程设计**

**最先适应算法：**

N

N

N

Y

Y

查找空闲分区表第一项

检索完？

分区大小>=size?

分区大小==size?

继续检索下一项

划出size大小的分区

将分区从链表中移除

修改有关数据结构

返回

Y

**最佳适应算法：**

先将空闲[分区表](http://baike.baidu.com/view/1412956.htm" \t "_blank)（空闲区链）中的空闲分区要按从小到大进行排序，自表头开始查找到第一个满足要求的自由分区分配。

分配同最先适应算法。

**最坏适应算法：**

该算法要求将所有的空闲分区按其容量从大到小的顺序形成一空闲分区链，查找时只要看第一个分区能否满足作业要求。

1. **主要函数实现**

1、程序结构定义

class Progress:

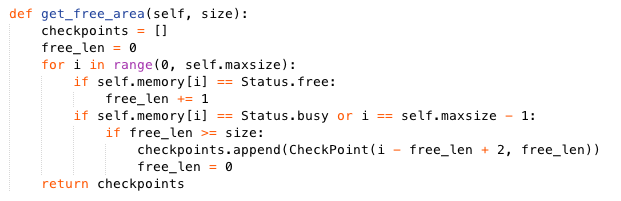
def \_\_init\_\_(self, start, size):

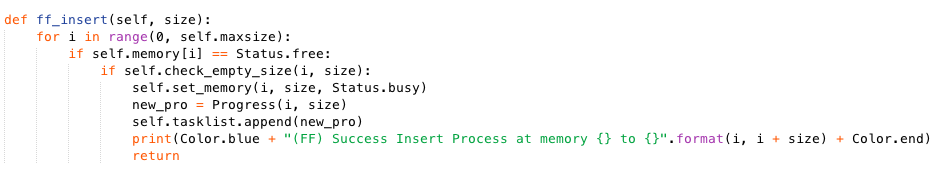
self.start = start

self.size = size

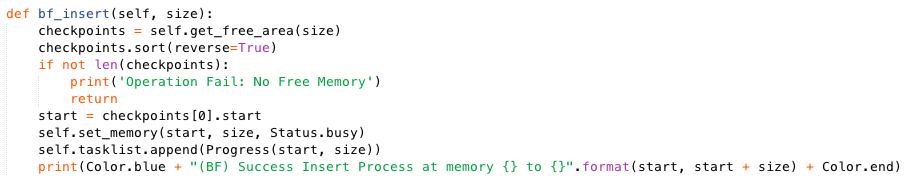
self.status = Status.busy

2、计算空闲区

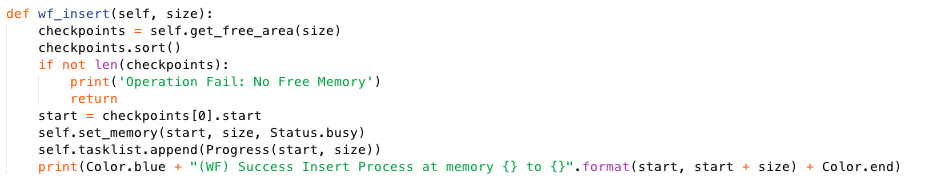


3、最先适应算法

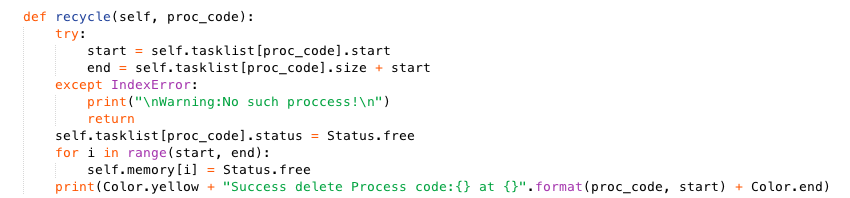
4、最佳适应算法

****

5、最坏适应算法

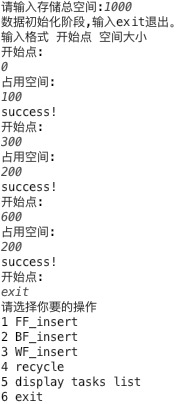
****

6、回收功能

****

1. **测试数据与结果分析**

初始化：根据要求输入预先分配的分区情况，完成后显示主菜单。选择查看结果5

以下为重新开始的新一轮分配，空间大小为200

1. **最先适应算法：分配100**

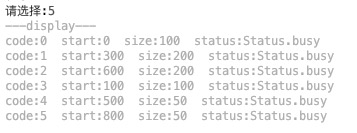


1. **最佳适应法：分配50**

****

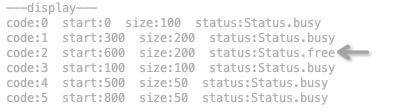
****

1. **最坏适应法：分配50**



1. **回收功能**





**四、实验总结**

本次的实验通过模拟动态分区的过程，让我对动态分区有了深刻的了解，并且了解到了存储分配策略：最先适应法，最佳适应法，最坏适应法的各自的优缺点,事实上，这几种方法实现收代码大同小异，只要考虑全面即可。本次实验过程中由于我们无法操作内核分配内存的过程，只是进行分配的模拟，但即使是这样，这次试验也大大加深了我对动态分区管理方法的各种算法的理解。俗话说“眼过千遍，不如手过一遍”，感谢这次实验给了我这次机会。

**参考文献**

1、鸟哥的Linux私房菜 （基础学习篇 第三版）鸟哥 著；王世江 编

2、数据结构（C语言版）严蔚敏、吴伟民编著 清华大学出版社

3、O'Reilly：深入浅出 Python（中文版） [美] 巴里（Barry P.） 著；林琪 等 译

4、操作系统课件

5、C程序设计（第三版） 谭浩强 著

**实验四 页面置换算法**

**一、实验目的**

本实验主要对操作系统中请求分页式内存管理及其应用的一些关键算法进行模拟。学生通过设计与实现Clock算法，能够加强对相应理论的理解，并对了解操作系统内部的基本处理原理与过程也有很多益处。

**二、实验要求**

基本要求：描述Clock算法的基本原理、必要的数据结构、算法执行流程图、编码实现。

1）初始化：输入作业可占用的总页框数，初始化置空。

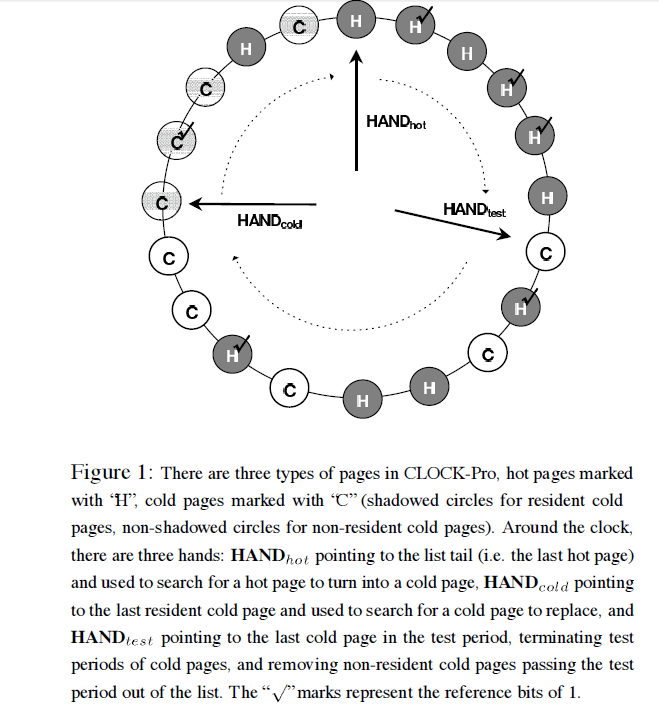
2）输入请求序列：输入一个作业页号访问请求序列，依次占用相应页框，直至全部占用；

3）Clock算法：当页框全部占用后，对于后续新的页号访问请求，执行Clock算法，淘汰1个页面后装入新的页号。

4）显示当前分配淘汰序列：显示淘汰的页号序列。

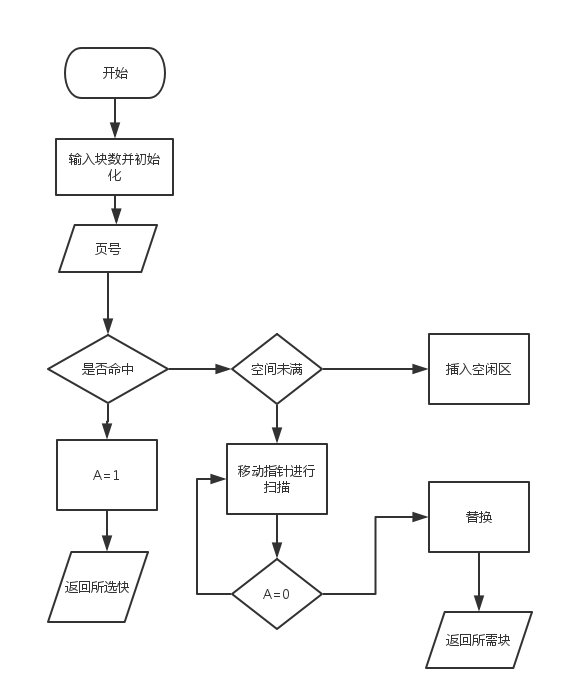
描述Clock算法的基本原理、必要的数据结构、算法执行流程图、编码实现。

**三、实验内容**

**1、基本原理**

时钟替换算法（Clock）：给每个页帧关联一个使用位。当该页第一次装入内存或者被重新访问到时，将使用位置为1。每次需要替换时，查找使用位被置为0的第一个帧进行替换。在扫描过程中，如果碰到使用位为1的帧，将使用位置为0，在继续扫描。如果所谓帧的使用位都为0，则替换第一个帧。

**2、算法流程设计**

**流程图：**

**3、主要函数实现**

**class** MemoryPage:

**1）初始化函数**  
 **def** \_\_init\_\_(self, size):  
 self.maxsize = size  
 self.pages = [**None**] \* size  
 self.pointer = 0  
**2）指针移动函数，构成循环队列** **def** move\_pointer(self,confirm = **True**):  
 pointer = (self.pointer + 1) % (self.maxsize)  
 **if** confirm:  
 self.pointer = pointer  
 **return** pointer

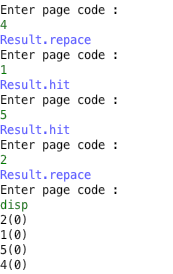
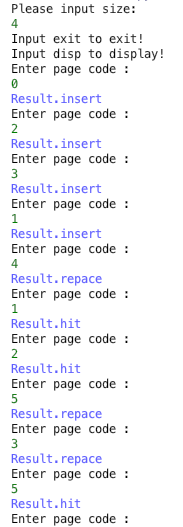
**3）检测指针所选项是否符合要求函数，主要对A进行判断** **def** match(self):  
 **if not** self.pages[self.move\_pointer(**False**)]:  
 **return False  
 if** self.pages[self.pointer].A == 0:  
 **return True  
 else**:  
 self.pages[self.pointer].A = 0  
 **return False**

**4）检测所需块是否命中函数  
 def** check\_exsit(self, page):  
 **for** i **in** range(0, self.maxsize):  
 **if** self.pages[i] **and** self.pages[i].code == page:  
 self.pages[i].A = 1  
 **return True  
 return False**

**5）处理用户输入块号及进行状态返回函数  
 def** get(self, page):  
 **if not** self.pages[self.pointer]:  
 self.insert(page)  
 **return** Result.insert  
 **if** self.check\_exsit(page):  
 **return** Result.hit  
 **if not** self.pages[self.pointer]:  
 self.insert(page)  
 **return** Result.insert  
 **if** self.match():  
 self.insert(page)  
 **return** Result.repace  
 **else**:  
 self.move\_pointer()  
 self.get(page)

**6）对指定块号进行插入、替换函数。** **def** insert(self, page):  
 self.pages[self.pointer] = PageStatus(page)  
 self.move\_pointer()

**4、测试数据与结果分析**



**四、实验总结**

通过本次实验，我深刻理解了Clock算法的基本原理，巩固了理论课上所学的知识。本次实验最重要的一部分是实现一个循环队列，实验中我选择使用数组的方式模拟循环队列。同时我使用递归的方式在循环队列中进行所需块的检测、插入、替换。在实验的过程中，要注意每一个细节都应当进行充分的考虑，否则便会出现严重的差错。这也是本次试验的难点之一。经历这次试验，对我的算法与编码能力有了进一步的提升，受益匪浅。

同时，经历本次试验后，我对操作系统中虚拟存储,页面置换等等算法与实现有了更加深入的理解，原本感觉生涩难懂的课本概念也变得容易了许多。总而言之，经历了本次试验，使我有了很大的收获。

**参考文献**

1、Clock\_pro算法描述

http://blog.csdn.net/hs794502825/article/details/9123469

2、数据结构（C语言版）严蔚敏、吴伟民编著 清华大学出版社

3、O'Reilly：深入浅出 Python（中文版） [美] 巴里（Barry P.） 著；林琪 等 译

4、操作系统课件

5、操作系统虚拟内存中的四种典型页替换算法(OPT,LRU,FIFO,Clock)

http://blog.csdn.net/wanghao109/article/details/13003479