组号： 40

****

操作系统课程设计报告

题 目： GeekOS操作系统的研究与实现

院（系）： 计算机与信息安全学院

专 业： 计算机科学与技术

组 长：

组 员：

指导老师：

职 称：

2021年5月10日

小组分组安排

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 团队成员 | 姓名 | 性别 | 学号 | 分工 | 小组评分 | 工作量 |
| 组长 |  | 男 |  | 项目0和1，报告撰写 | A | 30% |
| 组员 |  | 女 |  | 项目2，报告撰写 | A | 25% |
| 组员 |  | 男 |  | 项目3，报告撰写 | C | 15％ |
| 组员 |  | 男 |  | 项目4 | B | 20％ |

报告评分标准及得分

|  |  |  |
| --- | --- | --- |
| 序号 | 评分标准 | 得分 |
| 1 | 报告规范性（15分） |  |
| 2 | 论文工作量（20分） |  |
| 3 | 设计方案及应用能力（40分） |  |
| 3 | 团队组织形式（10分） |  |
| 4 | 实验结果及分析（15分） |  |
| 总分 | |  |

摘 要

操作系统（operation system，简称OS）是管理计算机硬件与软件资源的计算机程序。操作系统提供一个让用户与系统交互的操作界面以及各种服务,合理组织计算机工作流程，为用户有效使用计算机提供了一个良好的运行环境。《操作系统》课程作为计算机专业的核心课程,要求学生能够将理论与实践相结合，因此《操作系统课程设计》的存在就显得十分有必要了。

GeekOS是一个基于X86架构的PC上运行的微操作系统内核，由美国马理兰大学的教师开发，主要用于操作系统课程设计，目的是使学生能够实际动手参与到一个操作系统的开发工作中。在这次的操作系统课程设计中，我们基于Linux平台环境，完成了五个项目的开发，分别是：熟悉GeekOS系统环境的项目编译、调试与运行过程，熟悉并实现键盘操作函数；了解ELF文件格式以及GeekOS系统如何将ELF文件的可执行程序装入内存并建立和运行内核进程；扩充GeekOS操作系统内核实现用户级进程的动态创建和执行；扩充GeekOS进程调度算法——基于时间片轮转的进程多级反馈调度算法，并使用信号量实现进程的协作；实现分页虚拟存储管理，以替代在项目1和项目2中采用的分段存储管理。

该报告将会围绕上述五个项目进行展开，主要从绪论、系统分析、项目设计与实现、项目测试与编译运行、心得体会等几个方面进行具体的展开。

关键词：GeekOS；用户级进程；多级反馈调度算法；分页虚拟存储管理

目录

[1. 绪论 1](#_Toc22134)

[1.1 GeeKos操作系统概述 1](#_Toc32745)

[1.2 GeekOS系统源代码结构 1](#_Toc23689)

[1.3 实验开发环境与技术简介 3](#_Toc2418)

[2. 系统分析 3](#_Toc4426)

[2.2.1 项目0 4](#_Toc2355)

[2.2.2 项目1 5](#_Toc406)

[2.2.3 项目2 5](#_Toc3268)

[2.2.4 项目3 5](#_Toc4132)

[2.2.5 项目4 5](#_Toc20858)

[3. 项目设计与实现 6](#_Toc15495)

[3.1 项目0 6](#_Toc31204)

[3.1.1 项目设计目的 6](#_Toc25397)

[3.1.2 项目设计原理 6](#_Toc11321)

[3.1.3 项目具体实现 6](#_Toc18105)

[3.1.4 项目编译与结果分析 7](#_Toc18754)

[3.1.5 遇到的问题以及解决方法 8](#_Toc29563)

[3.2 项目1 9](#_Toc10114)

[3.2.1 项目设计目的 9](#_Toc23237)

[3.2.2 项目设计原理 9](#_Toc23721)

[3.2.3 项目具体实现 10](#_Toc9961)

[3.2.4 项目编译与结果分析 10](#_Toc12181)

[3.2.5 遇到的问题以及解决方法 12](#_Toc6588)

[3.3 项目2 13](#_Toc24433)

[3.3.1 项目设计目的 13](#_Toc4034)

[3.3.2 项目设计原理 13](#_Toc4273)

[3.3.3 项目具体实现 14](#_Toc21937)

[3.3.4 项目编译与结果分析 14](#_Toc30455)

[3.3.5 遇到的问题以及解决方法 17](#_Toc11078)

[3.4 项目3 18](#_Toc19242)

[3.4.1 项目设计目的 18](#_Toc25976)

[3.4.2 项目设计原理 18](#_Toc22346)

[3.4.3 项目具体实现 20](#_Toc29584)

[3.4.4 项目编译与结果分析 20](#_Toc28092)

[3.4.5 遇到的问题以及解决方法 23](#_Toc26030)

[3.5 项目4 24](#_Toc4135)

[3.5.1 项目设计目的 24](#_Toc18623)

[3.5.2 项目设计原理 24](#_Toc27167)

[3.5.3 项目具体实现 25](#_Toc2809)

[3.5.4 项目编译与结果分析 26](#_Toc9768)

[3.5.5 遇到的问题以及解决方法 28](#_Toc30036)

[4. 心得体会 28](#_Toc6107)

[参考文献 29](#_Toc27025)

1. **绪论**

计算机操作系统是管理计算机系统软件、硬件资源、控制程序运行、改善人机界面、提供各种服务、合理组织计算机工作流程和为用户有效使用计算提供良好的运行环境的系统软件，它为用户使用计算机提供一个方便、灵活、安全、可靠的工作环境。计算机操作系统是计算机应用软件赖以存在的基础，《操作系统》作为计算机专业的核心课程，要求学生能够掌握课本理论知识的同时也能够将其进行实践，将理论知识应用到具体实践中，真正实现理论与实践的相结合。而本次操作系统课程设计通过实现五个项目，让学生对GeekOS操作系统有进一步的了解与认识。

## GeeKos操作系统概述

GeekOS是一个基于X86架构的PC上运行的微操作系统内核，由美国马理兰大学的教师开发，主要用于操作系统课程设计，目的是使学生能够实际动手参与到一个操作系统的开发工作中。出于教学目的，这个系统内核设计简单，却又兼备实用性，它可以运行在真正的X86 PC硬件平台。作为一个课程设计平台，GeekOS由一个基本的操作系统内核作为基础，提供了操作系统与硬件之间的所有必备接口，实现了系统引导、实模式到保护模式的转换、中断调用及异常处理、基于段式的内存管理，FIFO进程调度算法以及内核进程、基本的输入输出（键盘作为输入设备、显示器作为输出设备），以及一个用于存放用户程序的只读文件系统PFAT。学生可以在Linux或Unix及Windows Cygwin环境下对其进行功能扩充。学生可以在 Linux 或 Unix 环 境下对其进行功能扩充，且其针对进程、文件系统、存储管理等操作 系统核心内容分别设计了 7 个难度逐渐增加的项目供学生选择。

## GeekOS系统源代码结构

GeekOS操作系统源文件geekos-0.3.0.zip可以从[http://geekos.sourceforge.net](http://geekos.sourceforge.net/)下载。解压后的GeekOS目录结构如图1.1所示:



图1.1 GeekOS目录结构

在doc目录下文件hacking.pdf和index.htm是GeekOS系统的参考文档。Scripts目录下有startProject和removeEmptyConflicts两个脚本文件。GeekOS系统的源文件在src目录下，分为7个项目：project0, project1, project2, project3, project4, project5, project6。每个项目的文件结构都类似，以project0为例，结构如图1.2所示：

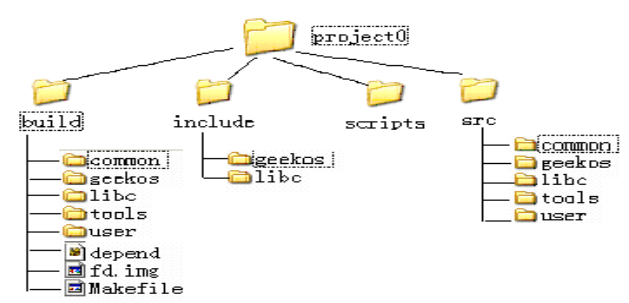


图1.2 project0文件结构

在build文件夹中，包含系统编译后的可执行文件的文件、软盘镜像或是硬盘镜像、makefile项目管理文件。在inculde文件夹中有geekOS和libc两个子目录，在geekOS子目录中有kthread.h,keyboard.h等头文件，在libc中包含有geekOS支持的C语言标准函数string.H头文件。在scripts文件夹是项目编译时要用到的一些脚本文件。Src文件夹中存放系统内核源代码，用户修改geekOS系统时要修改的源代码如main.c都位于这个目录中。在User子目录中一般是用来存放用户的测试文件，在tools子目录中的代码是用来建立PFAT测试文件系统的。

* 1. **实验开发环境与技术简介**

此次操作系统课程设计在Linux开发平台上进行实现，具体用到的实验平台以及软件版本如表1.1所示：

表1.1 操作系统实验开发环境

|  |  |
| --- | --- |
| 实验平台 | 软件版本 |
| 实验代码 | GeekOS-0.3.0 |
| 硬件模拟器 | BOCHS x86 |
| Linux操作系统 | Ubuntu9.04 |
| 虚拟机 | VMware Workstation12.5.7 build-5813279 |
| 主机系统 | Windows10 64-bit |
| 计算机硬件 | X86 PC |

1. **系统分析**
   1. **实验项目开发环境的搭建**
2. 在虚拟机上安装Linux操作系统

操作系统课程设计中的每个项目要求使用Linux环境下实现，因此我们首先要在Windows下下载安装Linux环境。Ubuntu是一个又社区开发的基于Linux的操作系统，它包含了用户一般所需的所有程序：无论是文字处理和电子邮件，还是web服务和编程工具。所以我们选择Ubuntu作为Linux系统进行安装。

首先要安装VMware Workstation，运行VMware，新建虚拟机或者导入一个已建好的虚拟机，以便能够安装Linux系统；接下来在虚拟机上安装好Ubuntu系统，由于系统安装时已经默认安装了GCC，因此后续实验过程中可以直接使用GCC编译而无需重新安装。

1. 下载GeekOS

Ubuntu内置了Firefox浏览器，打开网址 geekos.sourceforge.net，下载GeekOS 0.3到本地。

将其解压到 /home/[username]/geekos-0.3.0目录。

1. 配置Ubuntu中GeekOS开发环境
2. 修改使用国内镜像更新源

把镜像下载下来之后，修改的方法可以在Synaptic  Package Manager图形界面中设置，或者在命令行终端中输入sudo gedit /etc/apt/sources.list，使用文本编辑器编辑源列表。在Synaptic  Package Manager中reload或终端中输入sudo apt-get update 重新获取软件包列表。

1. 在终端中输入 sudo apt-get intall build-essential 下载安装 build-essential包。
2. 在终端中执行 sudo apt-get install gcc-3.4
3. **安装NASM中**

在终端在执行 sudo apt-get install nasm

1. **安装Bochs**

**在终端中**执行 sudo apt-get install bochs

在终端中执行 sudo apt-get install bochs-x

1. **编译GeekOS**

进入geekos-0.3.0/src/project0/build目录

执行 make depend

执行 make

成功之后在build 目录下生成fd.img文件。

1. **配置启动Bochs**

创建boches配置文件

启动bochs

* 1. **课程设计项目与要求**

### 项目0

要求实现一个内核进程，功能是实现从键盘接收一个按键，并在屏幕上显示。主要是让学生熟悉GeekOS的编译、运行过程，了解计算机系统的启动原理。

### 项目1

要求学生熟悉ELF文件格式，并编写代码对ELF文件进行分析，并将分析传送给加载器。主要让学生熟悉可执行链接文件（ELF文件）的结构，并学会加载和运行可执行文件。

### 项目2

要求学生实现对用户态进程的支持。在项目2实现之前，GeekOS一直使用内核进程。对用户态进程执行的支持包括用户态进程结构的初始化、用户进程空间的初始化、用户进程切换和用户程序导入等。该项目中，存储分配依然使用分段分配方式。实现项目2后，用户就可以使用GeekOS提供的命令行解释器Shell运行一些命令来执行PFAT文件系统内的用户测试程序。

### 项目3

要求学生改进GeekOS的调度程序，实现基于4级反馈队列的调度算法（初始GeekOS系统仅提供了FIFO进程调度算法），并实现信号量，支持进程间通信。

### 项目4

要求学生实现分页虚存管理，以替代在项目1和项目2中采用的分段存储管理。实现分页虚存管理后。系统在内存不够的情况下就可以将部分页调到硬盘，以释放内存实现虚拟存储技术。了解虚拟存储器管理设计原理，掌握请求分页虚拟存储管理的具体实现技术。

1. **项目设计与实现**

## 项目0

### 项目设计目的

熟悉GeekOS的项目编译、调试和运行环境，掌握GeekOS运行工作过程。

### 项目设计原理

项目要求从键盘输入，因此要用到键盘处理函数。GeekOS的键盘处理函数定义在keyboard.h与keyboard.c 文件中。键盘的初始化是在Main函数中调用Init\_Keyboard进行的，Init\_Keyboard主要功能是设置初始状态下存放键盘扫描码的缓冲区，并为键盘中断设置处理函数。而键盘中断处理过程是：首先从相应的I/O端口读取键盘扫描码，根据是否按下Shift键，分别在键值表中寻找扫描码对应的按键值，经过处理后将键值放入键盘缓冲区，最后通知系统重新调度进程。若用户进程需要从键盘输入信息，可调用Wait\_For\_Key()函数,进程调用该函数后，会阻塞进入按键操作的等待队列，直到按键操作结束，进程才会被唤醒。Start\_Kernel\_Thread函数主要功能就是建立一个内核线程。

本项目主要要求设计一个函数对键盘的中断进行响应。这主要通过使用Geekos提供的键盘响应函数Wait\_For\_Key()进行键盘中断的响应及返回键值。该函数首先检查键盘缓冲区是否有按键，如果有，就读取一个键码，如果此时键盘缓冲区中没有按键，就将线程放入键盘事件等待队列。于是可分两步完成:

1. 编写函数EchoCount，函数功能是：接受键盘输入的按键，并将键值显示到显示器，当输入Ctrl+D退出。
2. 在Main函数体内调用Start\_Kernel\_Thread函数，将编写的函数地址传递给startFunc，建立一个内核进程。

### 项目具体实现

### 项目编译与结果分析

1. 编译：在Linux环境下编译系统得到GeekOS镜像文件。具体如图3.1、图3.2所示:  
   进入Build目录下:

stu@stu-desktop:~/桌面/GeekOS /project0/build$ make depend

stu@stu-desktop:~/桌面/GeekOS /project0/build$ make

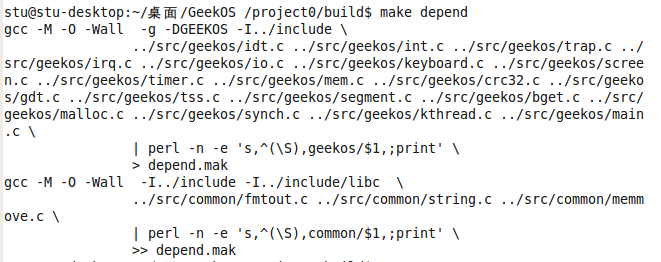


图3.1 make运行结果



图3.2 make depend运行结果

1. 运行：在bochs中运行GeekOS系统显示结果。运行结果如图3.3所示：  
   运行Bochs模拟器，执行GeekOS内核。

stu@stu-desktop:~/桌面/GeekOS /project0/build$ bochs

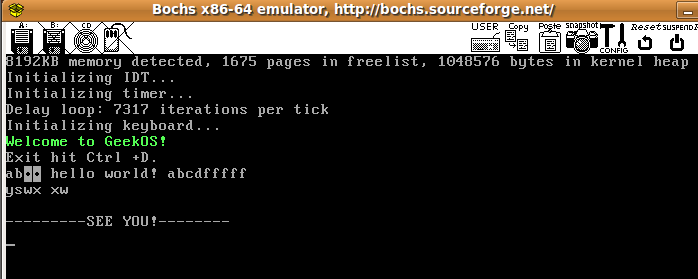


图3.3 项目0运行结果

1. 结果分析

由项目运行结果可知，输入bochs后运行出现”Exit Hit Ctrl +D”提示语句。

通过键盘向虚拟机输入任意字符，输入完毕后按“Ctrl+d”组合键可退出程序。

### 遇到的问题以及解决方法

在做项目过程中，程序在编译运行时出现了挺多的错误，不能出现实验要求的结果，经过多次运行测试并查看错误提示信息，发现是环境配置出现了问题，正确安装环境之后，可以正确得到图3.3所示运行结果。

## 项目1

### 项目设计目的

熟悉ELF文件格式，了解GeekOS系统如何将ELF格式的用户可执行程序装入内存，建立内核进程并运行的实现技术。

### 项目设计原理

ELF是UNIX系统实验室作为应用程序二进制接口而开发和发布的。有两个平行视图，具体如表3\_1所示：

表3\_1 两个平行视图

|  |  |
| --- | --- |
| **连接程序视图** | **执行程序视图** |
| **ELF 头部** | **ELF 头部** |
| **程序头部表（可选）** | **程序头部表** |
| **节区1** | **段 1** |
| **...** |
| **节区 n** | **段 2** |
| **...** |
| **...** | **...** |
| **节区头部表** | **节区头部表（可选）** |

GeekOS中的用户程序全部在系统的编译阶段完成编译和连接，形成可执行文件，用户可执行文件保存在PFAT文件系统中。本项目要完成的就是在系统启动后，从PFAT文件系统将可执行文件装入内存，建立进程并运行得到相应的输出。在磁盘中的ELF文件的映像和在内存中执行程序镜像间的对应关系如图3.4所示：

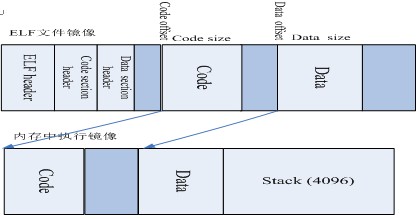


图3.4 磁盘中的ELF文件的映像和在内存中执行程序镜像间的对应关系

而此过程主要由Spawner函数实现，其主要经过简要概述为：先调用Read\_Fully函数将文件读入内存，后调用Parse\_ELF\_Executable函数分析ELF文件，最后调用Spawn\_Program函数将可执行程序的代码段和数据段等装入内存，此后便可以开始运行一个内核级进程了。如下图3.5所示：



图3.5 运行内核级进程过程

在本项目中，我们要完成Parse\_ELF\_Executable函数，此函数的作用为根据ELF文件格式，从exeFileData指向的内容中得到ELF文件头，继续分析可得到程序头和程序代码段等信息。

### 项目具体实现

### 项目编译与结果分析

1. 编译：在Linux环境下编译系统得到GeekOS镜像文件。具体如图3.6、图3.7所示

进入Build目录下:

stu@stu-desktop:~/桌面/GeekOS /project1/build$ make depend

stu@stu-desktop:~/桌面/GeekOS /project1/build$ make



图3.6 make运行结果

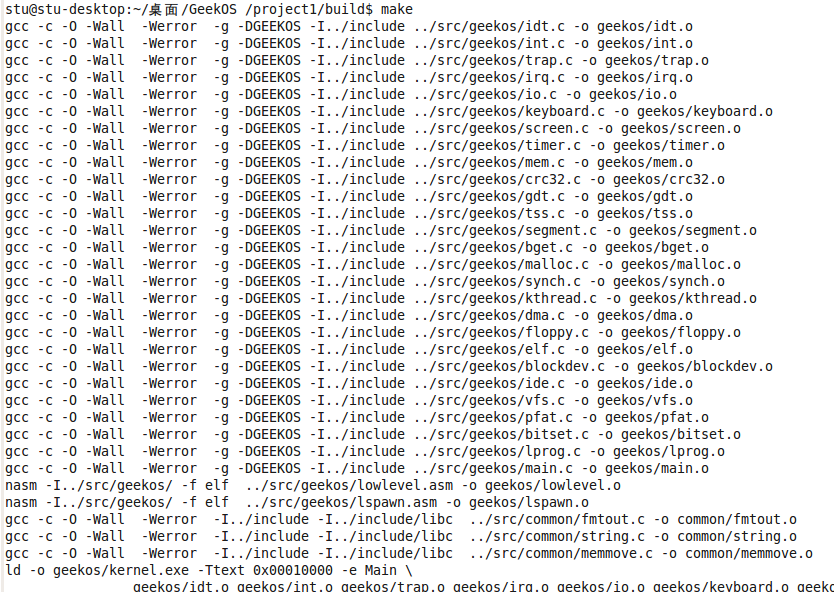


图3.7 make depend运行结果

1. 运行：进入“/project1/build/”目录，在终端中执行“bochs”命令，启动运行后得到结果如图3.8所示：

stu@stu-desktop:~/桌面/GeekOS /project1/build$ bochs

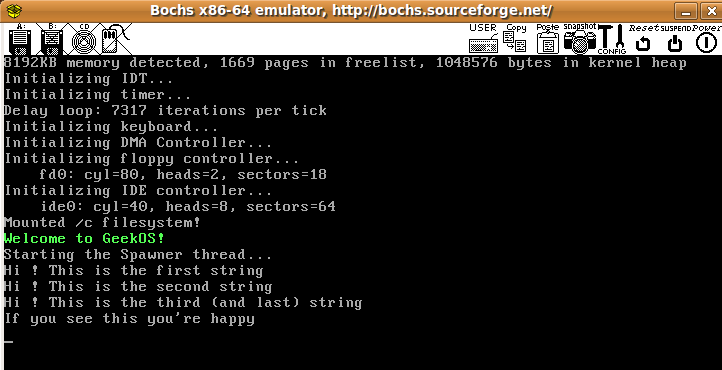


图3.8 项目1运行结果

1. 结果分析

由项目运行结果可知，运行结束后，屏幕成功显示a.c文件中的输出内容以及spawner函数中的输出内容。在源文件a.c文件中，既有局部变量也有全局变量，在实验过程中，我们已经将/桌面/GeekOS /project1/src/geekos/lprog.c文件中的Spawn\_Program()函数中的virtSize这一局部变量修改为了静态的全局变量，所以实验结果中会将a.c文件中的包括局部变量和全局变量在内的所有输出内容进行输出显示在屏幕上。代码开始执行后，首先会读取elf得到第一个elf的字符串，接下来是第二个，依次类推，最后得到最后一个串。最终得到如图3.8所示的运行结果。

### 遇到的问题以及解决方法

在做项目过程中，按照项目要求在相应的文件中对应的函数添加代码后，对项目进行编译运行，发现实验结果中只输出了second和last string这两个字符串，而first string这个字符串没有输出。经过对代码进行分析后，发现是没有进入/geekos/lprog.c中的Spawn\_Program()函数，将virtSize这一局部变量修改为全局变量。经过修改之后再编译运行发现a.c文件以及spawner函数中的输出内容中能够全部正确进行输出。

## 项目2

### 项目设计目的

扩充GeekOS操作系统内核，使得系统能够支持用户级进程的动态创建和执行。

### 项目设计原理

Geekos的初始系统不支持用户态进程，但提供了用户态进程上下文接口和实现用户态进程需要用到的数据结构。所以，用户态进程及相关函数都要开发者实现。在Geekos中为了区分用户态进程和内核进程，在Kernel\_Thread结构体中设置了一个字段userContext,指向用户态进程上下文。对于内核进程来说，这个指针为空，而用户态进程都拥有自己的用户上下文。因此在Geekos中要判断一个进程是内核进程还是用户态进程，只要通过userContext字段是否为空来判断就可以了。GeekOS的进程结构图如图3.9所示：

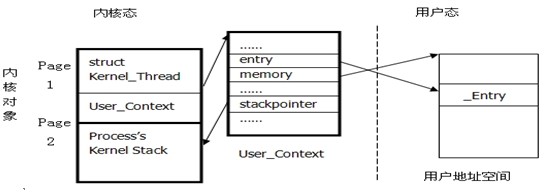


图3.9 用户态进程结构图

每个用户态进程都拥有属于自己的内存段空间，如：代码段、数据段、堆栈段等，每个段有一个段描述符（segment descriptor），并且每个进程有一个段描述符表（Local Descriptor Table），用于保存该进程的所有段描述符。操作系统中还设置一个全局描述符表（GDT，Global Descriptor Table），用于记录了系统中所有进程的ldt描述符。GDT、LDT和User\_Context的关系如图3.10所示

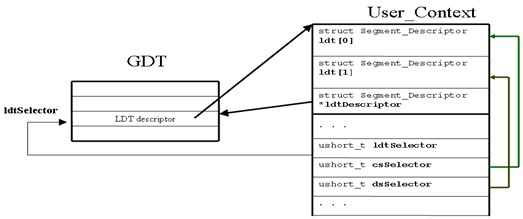


图3.10 GDT、LDT和User\_Context的关系

程序流程图如图3.11所示：

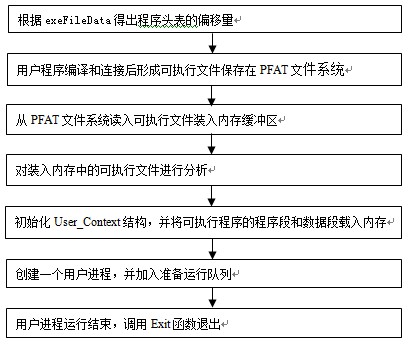


图3.11 程序流程图

GeekOS的用户级进程创建过程可以描述如下：

1. Spawn函数导入用户程序并初始化：调用Load\_User\_Program进行User\_Context的初始化及用户级进程空间的分配及用户程序各段的装入；
2. Spawn函数调用Start\_User\_Thread()，初始化一个用户态进程，包括初始化进程Kernel\_Thread结构以及调用Setup\_User\_Thread初始化用户级进程内核堆栈；
3. 最后Spawn函数退出，这时用户级进程已被添加至系统运行进程队列，可以被调度。

### 项目具体实现

### 项目编译与结果分析

1. 编译：在Linux环境下编译系统得到GeekOS镜像文件。具体如图3.12、图3,13所示：  
   进入Build目录下:

stu@stu-desktop:~/桌面/GeekOS /project2/build$ make depend

stu@stu-desktop:~/桌面/GeekOS /project2/build$ make



图3.12 make运行结果

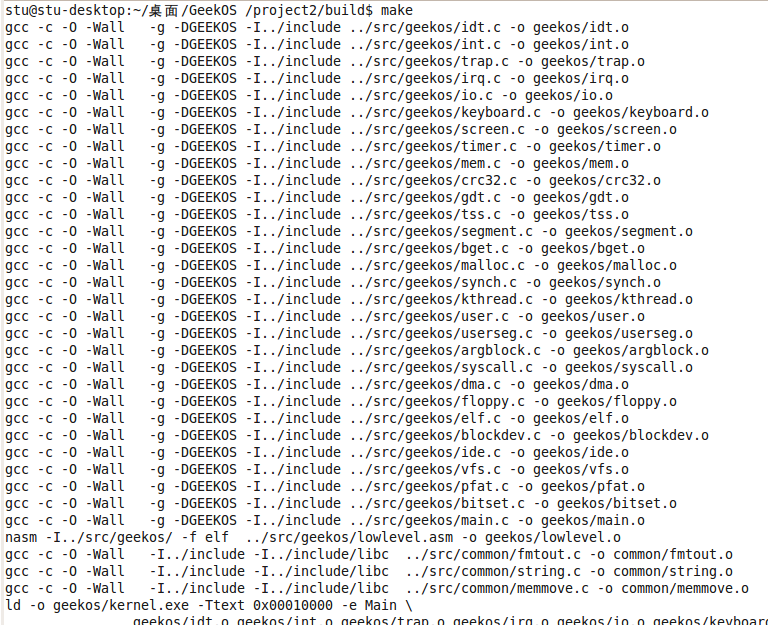


图3.13 make depend运行结果

1. 运行：bochs运行后，输入相关命令，即可执行project2/src/user/下的各个可执行文件。运行结果如图3.14、图3.15所示：

stu@stu-desktop:~/桌面/GeekOS /project2/build$ bochs

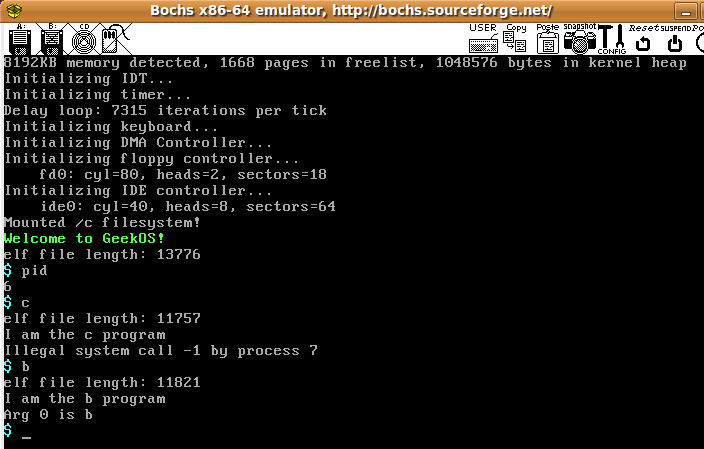


图3.14 项目2运行结果（1）

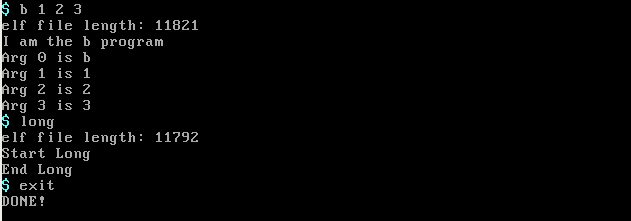


图3.15 项目2运行结果（2）

1. 结果分析
2. 输入exit，退出；
3. 输入pid，查询内核建立进程数。由项目运行结果可知，运行pid程序后在屏幕上输出为6，这是因为在系统初始化之前已经运行了5个进程，因此我们的第一个用户程序的进程 ID 是从 6 开始的。这五个进程分别为： Idle、Reaper、Init\_Floppy()、 Init\_IDE()、Main。而Idle、Reaper和Main三个进程，它们是由Init\_Scheduler函数创建，是执行进程，空闲进程，进程死掉时回收；Init\_Floppy()初始化软盘， Init\_IDE()初始化硬盘。
4. b程序带参数，这里需要注意，创建用户态进程时，除了分配用户态内存堆栈，还给参数专门分配了空间。
5. 输入设置以外的命令字符，到指令路径找相应程序，新建立进程执行，如：b,c,long,null（均放在project2/src/user/下）。

由于Spawn\_Init\_Process(void)里面填写的是shell程序，本项目中真正建立的第一个用户级进程其实是shell，通过它创建其他进程。从shell开始执行，输入shell程序中的相关命令，即可得到不同结果。shell是一个简单的命令解析器，它能分析用户的输入，shell中预制了很多命令，比如pid命令可以实现查询当前进程的pid。当系统运行时启动 shell 程序，将 shell.exe 作为可执行文件传递给 spawn 函数的 program 参数，创建用户态进程。运行后， geekos 就可以挂载在 shell 上，并能测试用户程序 b.exe 和 c.exe。

### 遇到的问题以及解决方法

在做项目过程中，最后对项目进行编译运行时，当进行读取文件时提示读取失败，这是因为shell在动态创建用户进程时，默认用户输入的exe文件的路径，所以当你在shell中输入c（/c目录下有c.exe文件）时，会先提示读取文件失败，然后再在/c和/a目录下搜索文件，如有这个文件就加载它。当输入/c/c.exe时就不会出现读取文件失败的错误提示。

## 项目3

### 项目设计目的

改进GeekOS的调度程序，实现基于4级反馈队列的调度算法(初始GeekOS系统仅提供了FIFO进程调度算法)，并实现信号量，支持进程间通信。

### 项目设计原理

1. Set\_Scheduling\_Policy(policy,quantum),policy是设置的调度策略，quantum是设置的时间片。例如policy为1说明设置的是多级反馈队列调度算法，g\_schedulingPolicy设置为1（g\_schedulingPolicy为系统添加的标识算法的变量，初始化为0）。如果policy为0，则说明设置的是轮转调度， g\_schedulingPolicy设置为0，把4个队列变成一个队列，即所有的线程都在队列0上了。
2. 在系统调用Sys\_GetTimeOfDay（）中，只需要返回g\_numTicks就可以了。在Sys\_SetSchedulingPolicy（）中，如果state->ebx是1，则设置的是MLF算法，为0则是RR算法。如果state->ebx为其他值，则返回-1。
3. 在Init\_Thread（）中都是把队列放在0队列上的，并且blocked变量为false。
4. 在Get\_Next\_Runnable（）中，从最高级的队列开始，调用Find\_Best（）来找线程优先级最大的线程，直到在某级队列中找到符合条件可以运行的线程。
5. 在Wait（）函数中，线程被阻塞，所以blocked变量被设置为true，并且如果是MLF算法，则该进程的currentReadyQueue加一，下次运行的时候进入高一级的线程队列。
6. 信号量定义

GveekOS定义了信号量的结构体：

Struct Semaphore

1. 信号量PV操作

信号量操作：

Semaphore\_Create()

Semaphore\_Acquire（P操作）

Semaphore\_Release（V操作）

Semaphore\_Destroy()

Semaphore\_Create()函数首先检查请求创建的这个信号量的名字是否存在，如果存在，那么就把这个线程加入到这个信号量所注册的线程链表上；如果不存在，则分配内存给新的信号量，清空它的线程队列，把当前的这个线程加入到它的线程队列中，设置注册线程数量为1，初始化信号量的名字，值和信号量的ID，并把这个信号量添加到信号量链表上，最后返回信号量的ID。

P操作Semaphore\_Acquire（）中，首先检查传入的信号量ID是否存在，如果存在，接着检查当前线程是否注册使用了这个信号量，如果这两项检查任意一项失败了，那么就返回-1。如果成功了，就把信号量的值减去1，如果减去1后信号量的值小于0，那么就把当前线程放入这个信号量的等待队列上。

V操作Semaphore\_Release（）中，首先也是检查传入的信号量ID是否存在，如果存在，接着检查当前线程是否注册使用了这个信号量，如果这两项检查任意一项失败了，那么就返回-1。如果成功了，那就把信号量的值加上1，如果加上1后信号量的值小于或等于0，则要把该信号量里等待队列上的一个线程唤醒。

Semaphore\_Destroy（）中，首先也是检查传入的信号量ID是否存在，如果存在，接着检查当前线程是否注册使用了这个信号量，如果这两项检查任意一项失败了，那么就返回-1。如果成功了，就把该线程从这个信号量的注册的线程数组中删除，并把注册的线程数量减去1。如果这个信号量的注册线程为0了，则把这个信号量从信号量链表中删除，并释放它的内存。

1. 多级反馈队列调度队列模型如下图3.16、图3.17所示：

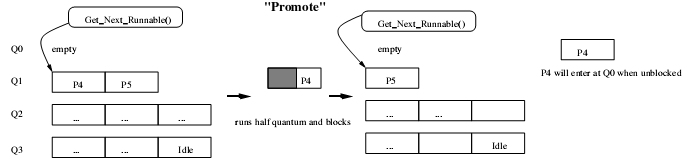


图3.16 多级反馈队列调度队列模型（1）

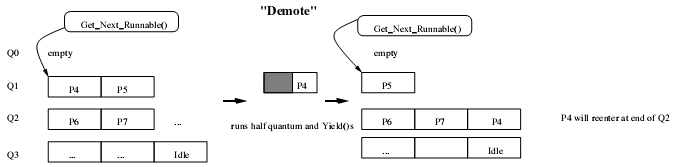


图3.17 多级反馈队列调度队列模型（2）

1. 多级反馈队列与分时调度进程队列的转换如图3.18所示：

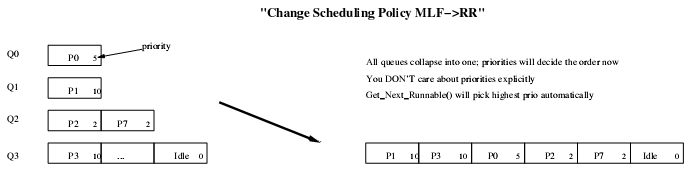


图3.18 多级反馈队列与分时调度进程队列的转换

### 项目具体实现

### 项目编译与结果分析

1. 编译：在Linux环境下编译系统得到GeekOS镜像文件。具体如图3.19、图3.20所示：  
   进入Build目录下:

stu@stu-desktop:~/桌面/GeekOS /project3/build$ make depend

stu@stu-desktop:~/桌面/GeekOS /project3/build$ make



图3.19 make运行结果

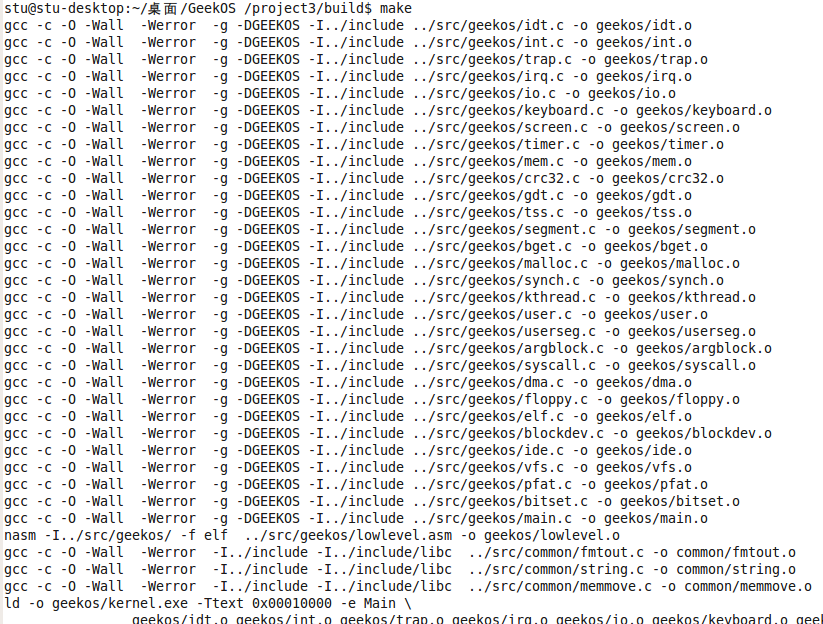


图3.20 make depend运行结果

1. 运行：进入“/project2/build/”目录，在终端中执行“bochs”命令，启动运行后得到结果如图3.21、图3.22所示，信号量测试结果如图3.23、图3.24、图3.25所示：

stu@stu-desktop:~/桌面/GeekOS /project3/build$ bochs

运行结果：

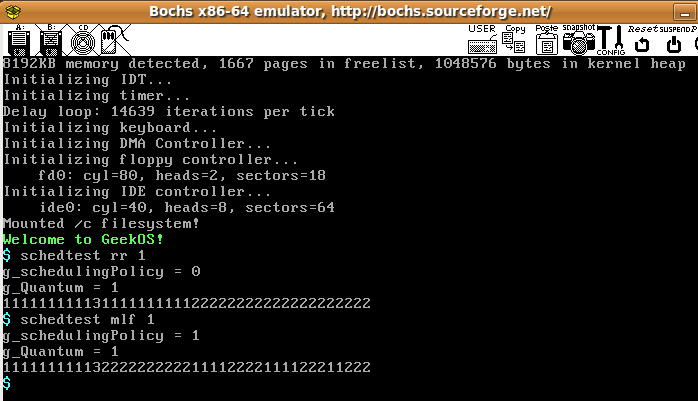


图3.21 项目3运行结果（1）

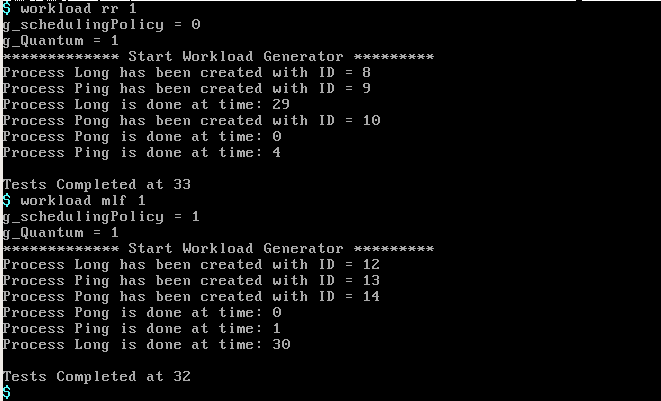


图3.22 项目3运行结果（2）

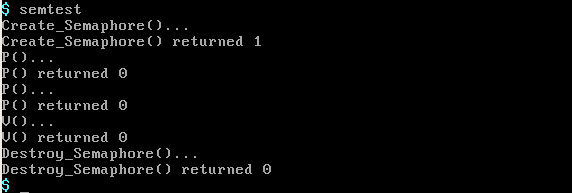


图3.23 项目3信号量测试运行结果（1）

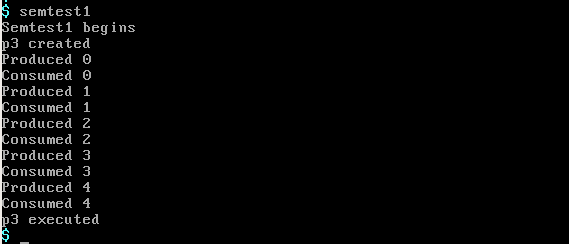


图3.24 项目3信号量测试运行结果（2）

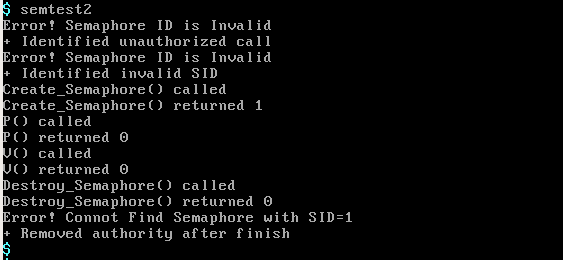


图3.25 项目3信号量测试运行结果（3）

1. 结果分析

本项目是为了测试不同的调度方式在时间片下运行多个进程的影响以及对信号量的测试。调度方式有多级反馈策略和轮询调度策略。从图3.21中可以看出，在schedtest的两次运行中，1总是运行完10次后再运行3，这是因为1和3有个共用的信号量，而1在3之前创建，所以实验结果符合预期。从我们的测试结果来看，这两种调度算法都是基于时间片轮转，输出2和输出1的数量基本一样多（多队列优先级调度算法的时间片都是一样的）。进程的周转时间还包括进程切换的时间。同样的调度算法，同样的时间片，程序执行的周转时间可能不一样（进程切换花费的时间可能不一样）。

对于信号量部分，从实验结果中可以看出，我们实现的信号量相关操作（创建、获取、 释放和销毁）都能够正确的在系统中执行，而且在semtest2.exe的测试中对于不正确的信号量操作（线程对未授权信号量进行操作、无效信号量ID和销毁信号量后再对其进行获取操作）系统也能够进行相应的处理和反馈，从此证明了信号量部分相关代码的正确性。

### 遇到的问题以及解决方法

在运行Project3时，发现在一个模拟器中无法运行完所有的测试命令，原因是在执行命令结束之后，没有释放内存，内存不足以执行完所有命令，而模拟器的megs最大容量为32。也无法执行完所有命令，所以最后是通过开启多个模拟器来执行完所有命令。

## 项目4

### 项目设计目的

要求实现分页虚拟存储管理，以替代在项目1和项目2中采用的分段存储管理。实现分页虚存管理后。系统在内存不够的情况下可以将部分页调到硬盘，以释放内存实现虚拟存储技术。

### 项目设计原理

系统增加了一个新的寄存器CR3作为指向当前页目录的指针。这样，从线性地址到物理地址的映射过程为：

1. 从CR3取得页目录的基地址；
2. 以线性地址中的页目录位段为下标，在目录中取得相应页表的基地址；
3. 以线性地址中的页表位段为下标，在所得到的页表中取得相应的页面描述项；
4. 将页面描述项中给出的页面基地址与线性地址中的页内偏移位段相加得到物理地址。

上述映射过程可用图直观地表示，具体如下图3.26所示：

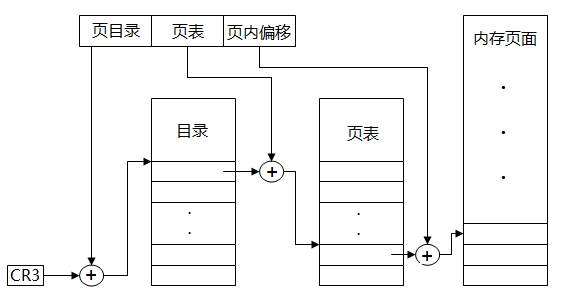


图3.26 线性地址到物理地址的映射

在GeekOS中，所有内核级进程共享一个页表，而每个用户级进程都有各自的页表。此外，用户模式进程的页表也包含访问内核模式内存的入口。GeekOS的内存布局如图3.27所示：

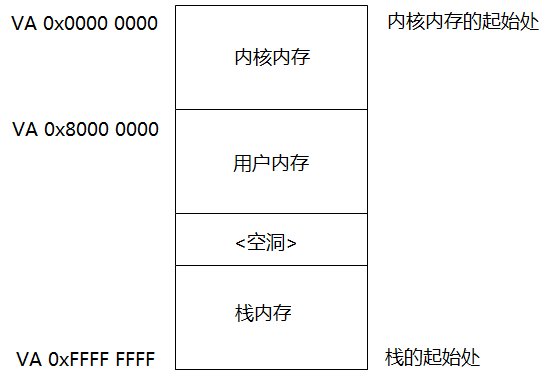


图3.27 GeekOS的内存布局

通过编写一个初始化页表和允许在处理器中使用分页模式的函数来为内核级进程创建一个页目录和页表入口，这个函数就是<project4\src\geekos\paging.c>中的Init\_VM函数。在<paging.c>的Init\_VM的Hints（提示）中，用户可以看到此函数的功能主要有以下三个：

（1）建立内核页目录表和页表；

（2）调用Enable\_Paging函数使分页机制有效；

（3）加入一个缺页中断处理程序，并注册其中断号为14。

用户设计的缺页中断处理程序应该能够认识到此页在page file中是存在的，并将其从磁盘读进内存。而当用户将一页从磁盘调入内存时，需要释放这一页占用的空间。

缺页中断处理程序应做的工作如下表3\_2所示。

表3\_2 缺页处理表

|  |  |  |
| --- | --- | --- |
| **缺页情况** | **标识** | **相应处理** |
| 堆栈生长到新页 | 超出原来分配一页的限制 | 分配一个新页进程继续 |
| 此页保存在磁盘上 | 数据标识这一页在page file中存在 | 从page file读入需要的页继续 |
| 因为无效地址缺页 | 非法地址访问 | 终止用户进程 |

### 项目具体实现

### 项目编译与结果分析

1. 编译：在Linux环境下编译系统得到GeekOS镜像文件。具体如图3.28、图3.29所示：  
   进入Build目录下:

stu@stu-desktop:~/桌面/GeekOS /project4/build$ make depend

stu@stu-desktop:~/桌面/GeekOS /project4/build$ make



图3.28 make运行结果

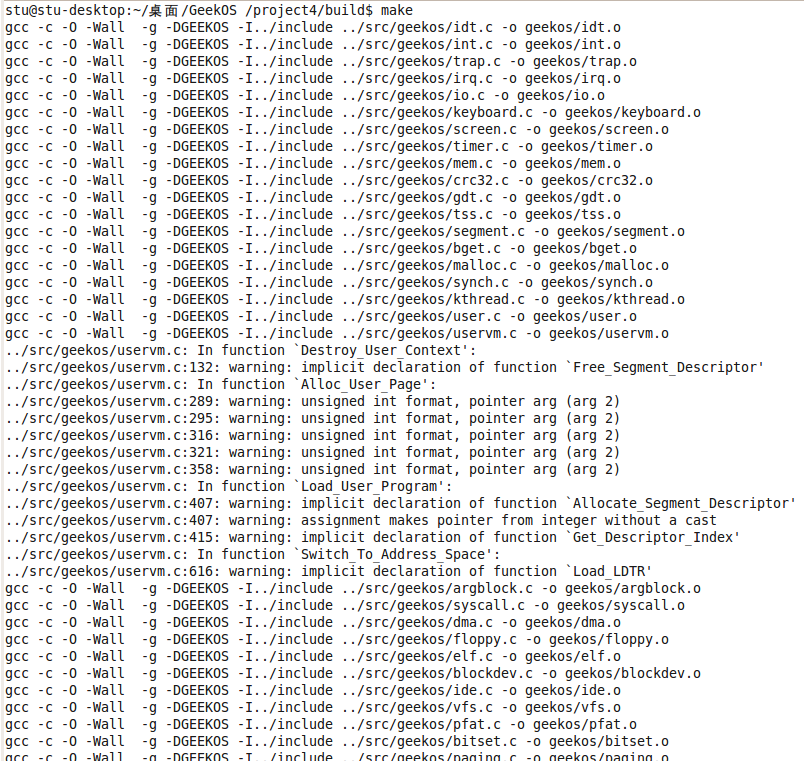


图3.29 make depend运行结果

1. 运行：进入“/project2/build/”目录，在终端中执行“bochs”命令，启动运行后得到结果如图3.30所示：

stu@stu-desktop:~/桌面/GeekOS /project4/build$ bochs

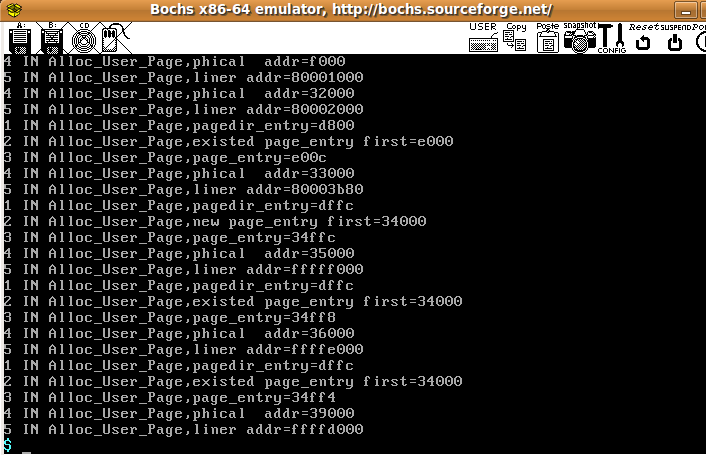


图3.30 项目4运行结果图（1）

在bochs中输入rec 4，得到结果如图3.31所示：

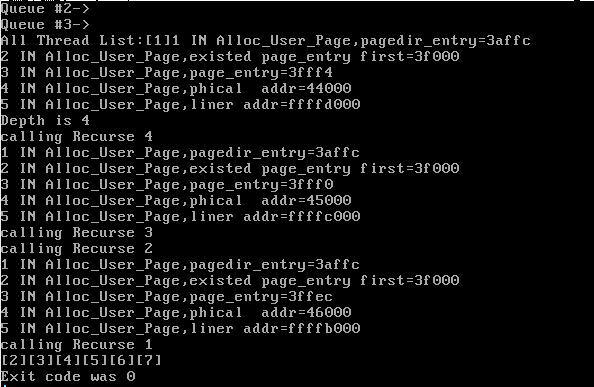


图3.31 项目4运行结果图（2）

1. 结果分析

由图3.30的运行结果可以看出，当前系统有进程号为1~5的进程正在运行，进程1打印当前页目录表的入口起始地址，进程2打印当前页表当前起始地址，进程3打印当前所在页表位置，进程4打印物理地址起始地址，进程5打印线性地址，再运行一次进程4打印当前线性地址对应物理地址。

输入命令rec 4后，得到图3.31的运行结果，由运行结果可以看出，当前进程队列下共有7个进程，进程1显示出当前页目录表入口为3affc，进程2显示已存在页表的入口地址为3f000，进程3显示该线性地址空间下对应查找到的页表下标地址为3eff0，进程4显示对应物理地址为45000，进程5显示当前线性地址为ffffc000；最后输出此次迭代递归的搜索深度为4，在这个过程中一共递归调用过4次进程，分别是调用递归进程4、进程3、进程2，以及进程1，最后结束递归调用，程序运行结束。

### 遇到的问题以及解决方法

项目4要做的工作量比较大,有一定的难度。要建立分页调度就必须要了解请求分页虚拟存储管理的原理以及分页系统的地址转换机制，这样才能更好地掌握请求分页虚拟存储管理的具体实现技术。在做此项目的过程中，遇到了各种各样的报错问题，通过查找资料以及与组员以及同学探讨，慢慢解决了在实现过程中的各种报错问题，最后成功实现了该项目。

1. 心得体会

通过实验学习，使我更加全面的加深了对操作系统理论知识的理解，用户模式及内核模式、内存分配、内核进程等操作系统的重要内容，更深刻的理解这些技术的实现思想有助于更好地理解和开发完善操作系统；加强了动手实践能力，对整个操作系统的架构、组织及功能的部署有了初步了解。

在做所有项目时，早期出现的问题主要集中在进程管理的实现上，包括混淆用户进程对象的内核堆栈与用户空间的内核堆栈，以至不断出现用户堆栈错误。这更让我体会到，代码的编写并不难，难在调试。后期，对于rr和mlf两种调度算法，一开始我们不是很明白在在本次实验中是如何实现的，经过我们团队的讨论后能更清楚其相同点和差别，即它们都是基于时间片轮转，输出2和输出1的的数量基本一样多（多队列优先级调度算法的时间片都是一样的）；进程的周转时间还包括进程切换的时间；同样的调度算法，同样的时间片，程序执行的周转时间可能不一样（进程切换花费的时间可能不一样）；信号量测试程序测试的是生产者消费者的互斥操作，生产一个消费一个。

此次课设如果靠自己单独完成是比较困难的，但有队友的帮助就容易很多了。团队之间的合作不仅能提高效率，还可以互相学习彼此的优点和技术，促进每个人的成长。在项目的实践过程中遇到了很多的困难，但也正是因为这些困难，加强了与团队沟通交流的能力。学习是一个相互的过程，通过组队的形式，可以了解到别人的不同理解方式，学习方法，学习的态度，研究的精神。更能了解到自己的不足之处，使自己在今后的学习生活中，获得更大的提高。

# 参考文献

1. 黄廷辉,王宇英.计算机操作系统实践教程.北京：清华大学出版社，2007.
2. 费翔林,骆斌.操作系统教程[第5版].北京：高等教育出版社，2014.2.
3. 胡元义,徐甲同.操作系统实践教程[M].西安：西安电子科技大学出版社,2001.