

组号: _____



桂林电子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

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: 黄海康

组 员: _____

指导老师: 韦顺忠

职 称: 辅导员

2023 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（%）

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）	

摘 要

摘要部分 300~500 字，主要给出课设设计的背景、目的和意义；课设设计完成的内容及效果。

操作系统作为一种特殊的软件，其在任务调度（多线程、并发），文件管理等等问题的解决方案和思路是典范性的。而且，操作系统对各种数据结构的设计和发明有促进作用，其在各种数据结构使用上也是典范性的。操作系统对解决很多工程问题提供了经典且有效的解决思路。因此，作为计算机专业的学生，学好操作系统是很有必要的。

为了学习操作系统，我们选择了一个既具备基本操作系统核心功能，与实际使用的操作系统比较接近，但又易于理解，规模较小的操作系统——geekOS 作为教学平台。GeekOS 是一个教育操作系统内核，GeekOS 尽可能简单实用。它运行在 x86 PC 上，简单地提供现代操作系统服务所需的最低功能，例如虚拟内存、文件系统和进程间通信

geekOS 提供了七个项目，而我们完成了 project0~project4 五个项目，下面是我们实验的内容及效果

- project0: 添加一个内核线程来从键盘中读取键并将它们回显到屏幕上
- project1: 熟悉 ELF 文件格式，了解 GeekOS 系统如何将 ELF 格式的可执行程序装入到内存，建立内核进程并运行的实现技术

关键词: (3~5 个)

gookOS、操作系统、虚拟内存

目 录

第一章 引言（一级标题）

*****（文字）

操作系统是对可用的硬件资源做了抽象，它使得我们能够以相似的编程的方式调用不同的底层硬件，并且忽略同类硬件的不同实现的区别。所以我们的编写的应用程序是一定要调用操作系统给的接口来进行输入输出和计算的。你当然可以不去理解操作系统做着自己的编程工作，但你不是一直要写 `helloworld`，你会慢慢需要知道什么是进程，进程和线程的区别，开机引导怎么设置等等，哪怕只有一点点那你也是需要了解操作系统。

身为一个计算机专业的学生，应该知道计算机系统包括硬件系统和软件系统，操作系统毫无疑问是软件系统的重要部分，本科阶段的教学目的是为了给未来提供更多的可能性和上升空间，投资的并不是立竿见影的具体操作。操作系统是庞大的全面的，可以在里面见识到各种数据结构的巧妙应用，解决问题的奇思妙想，毫无疑问裨益非凡。

为了学习操作系统，我们选择了 `geekOS`：一个既具备基本操作系统核心功能，与实际使用的操作系统比较接近，但又易于理解，规模较小的操作系统。它运行在 `x86 PC` 上，简单地提供现代操作系统服务所需的最低功能，例如虚拟内存、文件系统和进程间通信。

系统分析

GeekOS 实验项目开发环境的搭建

虚拟机准备

这里我们选择 [virtualbox-6.0.12](#) + [ubuntu-9.04-desktop-i386.iso](#) 搭建实验环境。根据我的亲身体会，不建议使用其他版本，否则会出现各种在 9.04 下不会出现的错误。

此外，安装过程十分漫长，请耐心等待。

环境搭建

1. 由于系统年代久远，其自带的软件包列表已不可用，因此我们需要更新软件包列表：`sudo apt-get update`
2. 下载安装 `build-essential` 包：`sudo apt-get install build-essential`
3. 安装 NASM：`sudo apt-get install nasm`
4. 安装 Bochs：`sudo apt-get install bochs && sudo apt-get install bochs-x`

项目分析

项目 0——GeekOS 系统环境调试及编译

实验内容是设计一个键盘操作函数，编程实现一个内核进程。该进程的功能是：接受键盘输入的字符并显示在屏幕上，当按 `ctrl+d` 时，结束进程的运行

项目 1——内核级线程设计及实现

熟悉 ELF 文件格式，了解 GeekOS 系统如何将 ELF 格式的可执行程序装入到内存，建立内核进程并运行的实现技术。在此项目中，GeekOS 需要从磁盘加载一个可执行文件到内存中，并且在内核线程中执行此程序。我们需要做的就是完成 `project0/src/geekos/elf.c` 中 `ParseELFExecutable` 函数，把 EXE 文件的内容填充到指定的 `Exe_format` 格式的区域。

项目 2——用户级进程的动态创建和执行

GeekOS 系统最早创建的内核进程有 `Idle`、`Reaper` 和 `Main` 三个进程，它们由 `Init_Scheduler` 函数创建：最先初始化一个核态进程 `mainThread`，并将该进程作为当前运行进程，函数最后还调用 `Start_Kernel_Thread` 函数创建了两个系统进程 `Idle` 和 `Reaper`。所以，`Idle`、`Reaper` 和 `Main` 三个进程是系统中最早存在的进程。GeekOS 系统初始的内核是不支持用户态进程，所以本项目的设计目的是扩充 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。在内核进程控制

块 `Kernel_Thread` 中有一个字段 `User_Context` 用于标志进程是内核进程还是用户态进程。若为核心进程，则字段赋值为 0。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（`User_Context`）。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。

项目 3——线程调度算法和信号量功能实现

本项目在 `project2` 了解进程的基础上，进行进行的调度管理以及信号量的实现和操作。该项目主要目的是，研究进程的调度算法，为 `GeekOS` 扩充一种基于时间片轮转的进程多级反馈调度算法，掌握用信号量实现进程间同步的方法，并使用信号量实现进程协作。

项目 4——请求分页虚拟存储管理

项目 4 的目的是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。分页存储管理中有三种地址：逻辑地址、线性地址和物理地址，线性地址到物理地址的映射过程为：取得页目录的基地址；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；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；最后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。

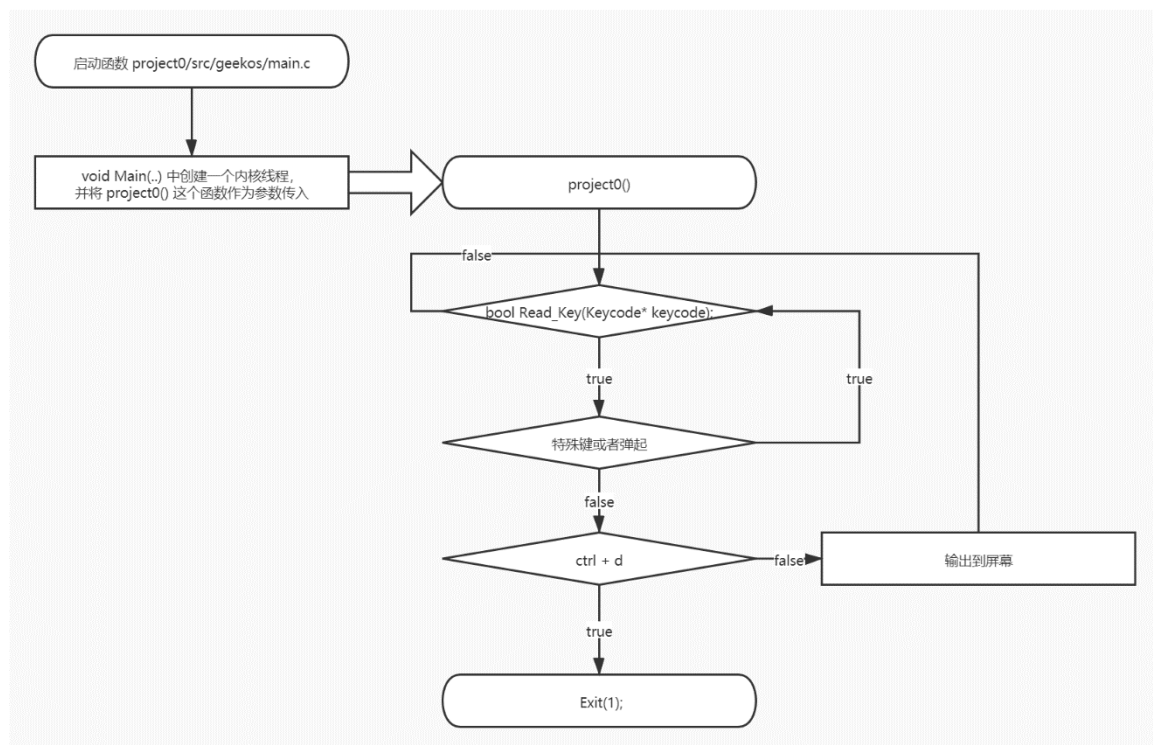
在 `GeekOS` 中，所有内核级进程共享一个页表，而每个用户级进程都有各自的页表。当系统需要运行某个进程时，就把该进程对应的页表调入内存，并使之驻留在内存中，这样就可以运行该用户级线程。所以要实现请求分页虚拟存储管理，首先需要为内核程序空间建立页表，这部分内容在函数 `Init_VM` 中完成，然后为用户进程建立页表，并实现请求分页技术和进程终止处理。

系统设计 with 实现

项目设计流程图和原理

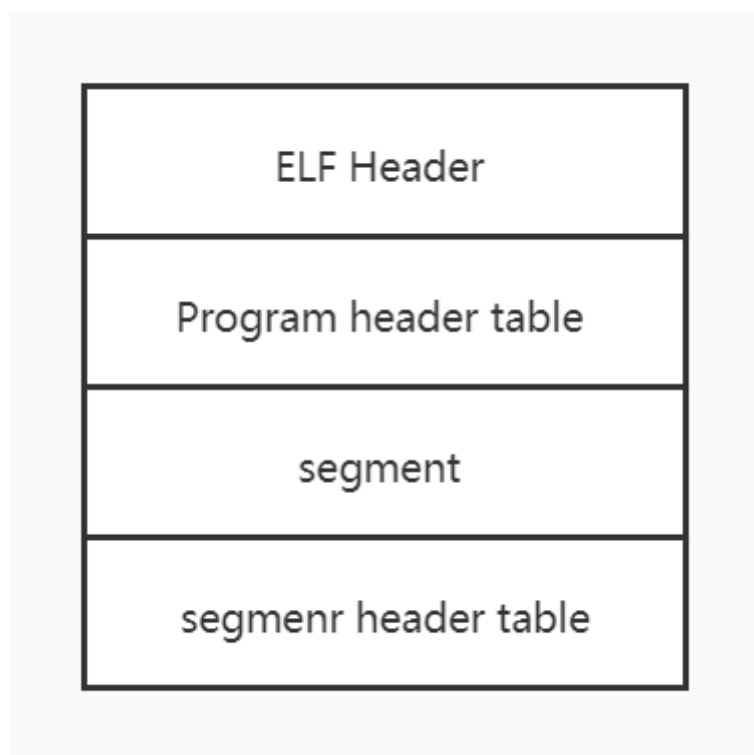
项目 0——GeekOS 系统环境调试及编译

启动 `geekOS` 后调用 `StartKernelThread`，创建了一个接受键盘输入的字符并显示在屏幕上的内核进程 `proect0`。`proect0` 循环调用 `ReadKey()`，读取键码后，判断是否是 `ctrl+d`，若是则退出；否则输出接收到的字符到屏幕，然后继续调用 `ReadKey()` 监听。



项目 1——内核级线程设计及实现

在计算机科学中，是一种用于二进制文件、可执行文件、目标代码、共享库和核心转储格式文件。是 UNIX 系统实验室（USL）作为应用程序二进制接口（Application Binary Interface, ABI）而开发和发布的，也是 Linux 的主要可执行文件格式。其格式如下：



geekOS 启动后进入启动函数 `Main()`, `Main()` 中由 `SpawnInitProcess()` 函数调用 `StartKernelThread()`, 以 `Spawner()` 函数为进程体建立一个内核态进程, 并放入就绪队列。 `Spawner` 函数运行过程为: 通过 `ReadFully` 函数将 *ELF* 可执行文件读入内存缓冲区; 通过 `ParseELFExecutable` 函数解析 *ELF* 文件, 并通过 `SpawnProgram` 函数执行 *ELF* 文件。

启动函数 void Main(struct Boot_Info* bootInfo)



Spawn_Init_Process()



Start_Kernel_Thead()



Spawner()



Read_Fully()



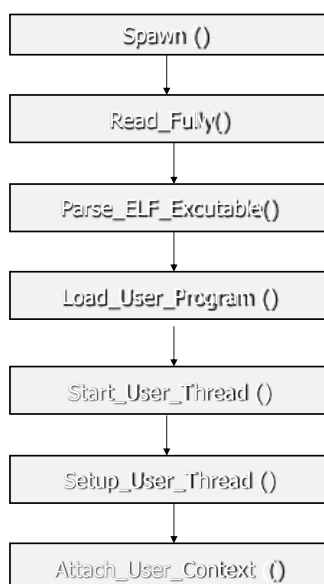
Prese_ELF_Executable()



Spawn_Program()

项目 2——用户级进程的动态创建和执行

用户态进程创建流程



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、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。

用户态进程创建 LDT 的步骤：

- (1)调用函数 `Allocate_Segment_Descriptor()`新建一个 LDT 描述符；
- (2)调用函数 `Selector()`新建一个 LDT 选择子；
- (3)调用函数 `Init_Code_Segment_Descriptor()`初始化一个文本段描述符；
- (4)调用函数 `Init_Data_Segment_Descriptor()`初始化一个数据段描述符；
- (5)调用函数 `Selector()`新建一个数据段选择子；
- (6)调用函数 `Selector()`新建一个文本（可执行代码）段选择子。

项目 3——线程调度算法和信号量功能实现

如图 1.1 是进程调度处理的过程，进程定义结构体中 `numTicks` 变量初始化对象时为 0，此后每次时钟中断，都会加一。用以比较进程执行时间是否超过了系统规定的时间片 `g_Quantum`，超过表明时间片用完，使用 `Make_Runnable` 函数调度新的进程运行，将当前运行进程放入准备运行的进程队列 `s_runQueue` 中，之后调用 `Get_Next_Runnable` 函数找到优先级最高的进程当做当前运行的进程，从准备队列里取出并运行。

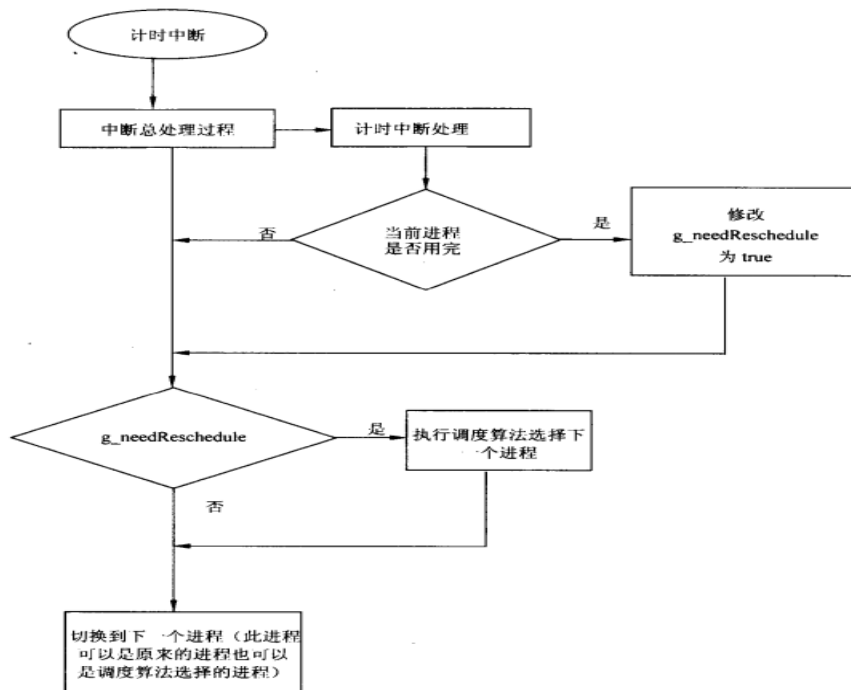
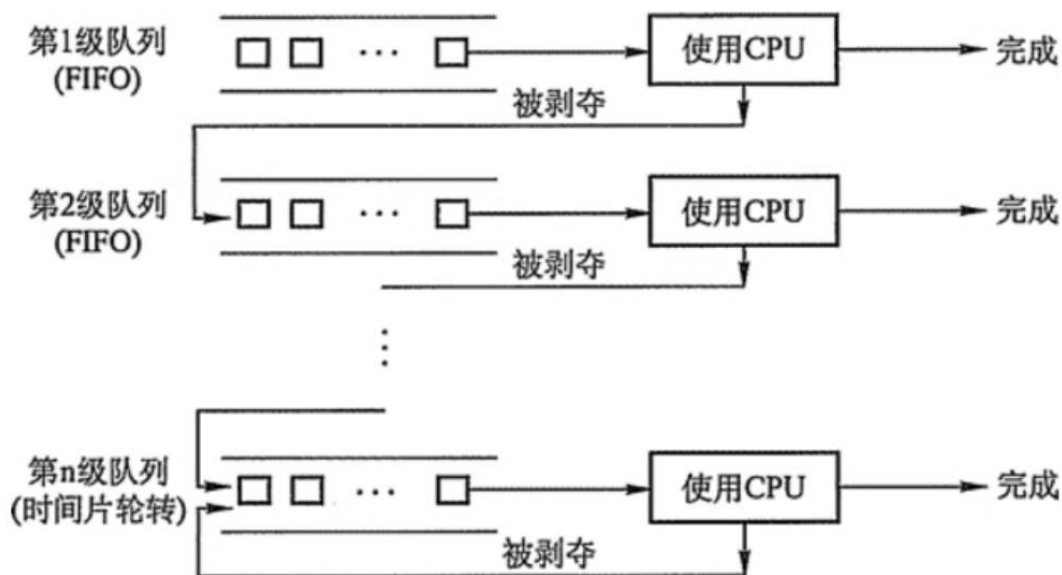


图 1.1 进程调度处理过程

如图 1.2，是多级反馈队列调度的基本流程，分为多个优先级不同的队列，新创建的进程首先进入优先级最高的准备队列，在给定的时间片内无法完成，则进入次一级队列的队尾，以此类推。对于系统中的空闲进程 Idle，应该始终放在最低优先级队列的队尾。



四级反馈队列调度策略实现的方法，首先用四个准备的运行队列替换系统的原来的一个队列，并为队列设置不同的优先级 0-3, 0 为最高优先级。修改 `s_runQueue` 的定义, 将 `MAX_QUEUE_LEVEL` 设置为 4，构成结构体数组，结构体数组用于存放队列的队首指针。进程调度时，使用 `Sys_SetSchedulingPolicy` 函数选择使用 RR 还是 MLF 调度策略。使用 `Get_Next_Runnable` 函数从优先级为 0 的队列里开始寻找，具体使用 `Get_Front_Of_Thread_Queue` 函数找到应该调用的线程。

进程调度的优劣可以使用进程执行时间来衡量，即从创建到结束所花费的时间。在系统中可以通过调用 `Sys_GetTimeOfDay` 得到，分别在进程开始和结束时调用一次，获取全局变量 `g_numTicks` 的值，通过差值计算花费。

通过信号量和 PV 操作实现进程同步，其基本原理是，使用信号量标记存储使用该临界资源的进程信息，信号量只能由 PV 操作来改变。P 用来获取信号量，V 用来释放信号量，当执行 P 操作，信号量值减一，V 操作则相反，当信号量的值为 0 时进程阻塞。

具体实现通过 `Sys_P` 和 `Sys_V` 两个系统调用，当执行这两个系统调用操作信号量是，内核首先检测进程是否有操作权限，有则调用具体相应函数获得或者释放。`Sys_CreateSemaphore` 和 `Sys_DestroySemaphore` 分别用来创建和销毁信号量。

创建信号量时首先通过传入的参数获取信号量的名称，通过名称查找信号量是否在信号量队列，没有则创建信号量并加入队列，然后将进程注册加入到信号量的注册队列。当所有使用该信号量的进程都结束了，则销毁掉这个信号量。

项目 4——请求分页虚拟存储管理

为内核程序空间建立页表。要实现分页系统，第一步是修改用户的项目使用页表和段，这一步在函数 `Init_VM` 中完成，先分配一个页目录表，然后为整个存储区间的内容分配页表，从而给所有线性地址建立映射，每一个线性地址由页表来映射到物理地址。除此之外，对缺页的处理在分页存储管理中必不可少，需在函数 `Init_VM` 中加入缺页中断处理程序，通过调用 `Install_Interrupt_Handler` 函数来安装中断处理程序。

为用户进程建立页表，让用户进程拥有它们自己的线性地址空间一个用户进程的存储格式如图 3.4.1 所示。在 `uservm.c` 中修改用户进程在用户的存储区域使用分页，即对释放进程、装载可执行文件、用户缓冲区与内核缓冲区之间数据传输方式进行修改，这一步参考 `userseg.c` 完成。然后在 `Load_User_Program` 函数中分两步实现用户进程的分页系统，首先复制线性存储空间的低 2GB 的所有核心页目录表项到用户进程的页目录表，从而为进程分配一个页目录表；然后为用户进程的代码、数据和堆栈区间分配页表项，通过函数 `Alloc_Pageable_Page` 来分配这三个区间的页。由于用户模式进程的线性地址的基地址是 `0x8000 0000`，界限地址为 `0xFFFF FFFF`，所以为了能正常使用分页，需切换 PDBR 寄存器的内容，在装入 LDT 之后，函数 `Switch_To_Address_Space` 加入一个 `SET_PDBR` 调用作为上下文切换的一部分。

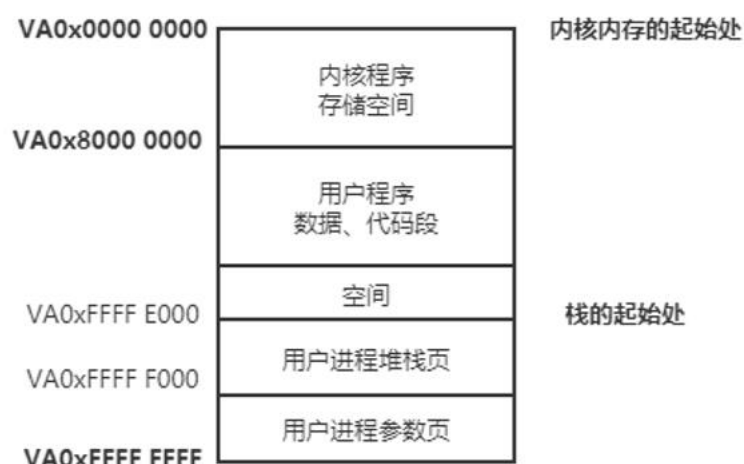


图 3.4.1 用户进程的存储格式

实现请求分页技术。使用虚拟存储技术可以把内存里的页暂时存储到磁盘上，从而使进程免受物理存储空间大小的制约，为了实现这一点操作系统将需要在磁盘设备上创建一个 `page file` 文件暂时保存从内存中替换出去的页，并实现一个类 LRU 算法在内存中选取一个替换页把它写到磁盘的 `page file` 文件中。交换一页到磁盘的操作步骤：确定要替换的页→在 `page file` 中找到空闲的存储空间→把被替换的页写到 `page file` 文件中→标识为此页在内存为不存在→修改页表项的页基地址→标识这一页是在磁盘上存在→刷新 TLB。当访问一个不在内存的页时，将引起缺页中断，缺页处理程序在不同情况下的处理如表 4.1 所示。

缺页情况	标识	相应处理
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 <code>page file</code> 中存在	从 <code>page file</code> 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

表 4.1 缺页处理表

处理进程终止。进程终止时，需释放占用的内存空间，修改 `Destroy_User_Context` 函数完成以下操作：释放进程的页、页表和页目录表对应的内存空间。

系统测试及编译运行

系统编译运行的原理

项目 0——GeekOS 系统环境调试及编译

总的来说，就是启动 `geekOS` 后调用 `StartKerneThread`，创建了一个接受键盘输入的字符并显示在屏幕上的内核进程，其细节如下：

- `Keycode WaitForKey(void)`: `project0/src/geekos/keyboard.c` 中实现了函数 `Keycode WaitForKey(void)`，它的功能是循环等待一个键盘事件，然后返回一个 16 位的 `Keycode`。此外，在 `project0/src/geekos/keyboard.h` 中定义了所有的键盘代码。

- `bool ReadKey(Keycode* keycode)`: `project0/src/geekos/keyboard.c` 中实现了函数 `bool ReadKey(Keycode* keycode)`，它的功能是轮询键盘事件，如果捕获到键盘事件，则返回 `true`，并且将按键码保存到参数 `keycode` 地址中，否则返回 `false`。
- `StartKernelThread`: `project0/src/geekos/kthread.c` 定义了 `StartKernelThread`，它的功能是建立一个内核线程。它需要一个 `ThreadStartFunc` 类型的函数指针作为参数，参数 `Start_Func` 指向的代码为进程体生成一个内核进程
- `ThreadStartFunc` : `project0/src/geekos/kthread.h` 中定义了 `Typedef void (*ThreadStartFunc)(ulongt,arg)`; 该函数指针指向一个无返回值、参数为 `ulongt` 类型的函数。与 `StartKernelThread` 配合使用，指向的代码为进程体生成一个内核进程

在 `project0` 中，`ThreadStartFunc` 就是我们自己编写的接受键盘输入字符并显示在屏幕上的函数 `project0()`。

项目 1——内核级线程设计及实现

在此项目中，GeekOS 需要从磁盘加载一个可执行文件到内存中，并且在内核线程中执行此程序。我们需要做的就是完成 `project0/src/geekos/elf.c` 中 `ParseELFExecutable` 函数，把 EXE 文件的内容填充到指定的 `Exe_format` 格式的区域。

在 `project0/include/geekos/elf.h` 中，我们可以找到 `elfHeader`、`elfHeader`、`ExeSegment` 和 `ExeFormat` 的定义。掌握 ELF 的各个结构后程序就很容易了。

在 `project0/src/geekos/elf.c` 中，我们实现 `int ParseELFExecutable(char *exeFileData, ulongt exeFileLength, struct ExeFormat *exeFormat)`。其各参数含义如下：

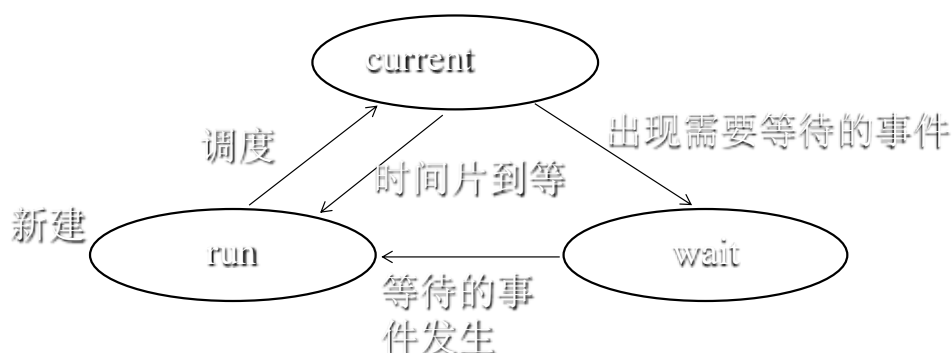
- `exeFileData`: 已装入内存的可执行文件所占用空间的起始地址
- `exeFileLength`: 可执行文件长度
- `exeFormat`: 保存分析得到的 elf 文件信息的结构体指针根据 ELF 文件格式，用户可以从 `exeFileData` 指向的内容中得到 ELF 文件头，继续分析可以得到程序头，程序代码段等信息

具体实现见代码。

geekOS 启动后，程序就会运行到 `project0/src/geekos/lprog.c` 下的 `void Spawner(unsigned long arg)`。`Spawner()` 程序中会读取并运行 `/c/a.exe`，此 `.exe` 文件由 `project0/src/user/a.c` 编译而来。执行 `/c/a.exe` 的过程中，会在屏幕输出两句话：`"Hi ! This is the first string\n"` 和 `"Hi ! This is the second string\n"`。`/c/a.exe` 运行完毕后回到 `Spawner()`，然后再 `Print` 两句：`"Hi ! This is the third (and last)"`。

string\n" 和 "If you see this you're happy\n", 到此，程序结束。

项目 2——用户级进程的动态创建和执行



内核级线程是由内核维护，所以内核可以感知到线程的存在。那么，用户级线程中的缺点可以得到解决，即同一进程的多个进程可以在多处理器上并行执行，且一个线程的阻塞不会阻塞整个进程。但是，用户级线程的优点也变成了内核级线程的缺点，即线程间切换需要进行模式转换。而用户级线程是由进程用户空间中运行的线程库来创建并管理，线程库调度线程，类似于内核调度进程，而内核是感知不到线程存在的。优点：切换用户级线程时，进程不需要为了管理线程而切换到内核模式，节省了两次状态转换（即用户模式到内核模式，内核模式到用户模式）的开销。缺点：由于内核不知

道线程的存在，内核是以进程为单位进行调度的，所以当进程中的某一个线程阻塞时，就会认为整个进程就阻塞了。并且，内核每次只把一个进程分配给一个处理器，那么一个进程内就无法使用多处理器处理技术。

项目 3——线程调度算法和信号量功能实现

如图 1.1 是进程调度处理的过程，进程定义结构体中 numTicks 变量初始化对象时为 0，此后每次时钟中断，都会加一。用以比较进程执行时间是否超过了系统规定的时间片 g_Quantum,超过表明时间片用完，使用 Make_Runnable 函数调度新的进程运行，将当前运行进程放入准备运行的进程队列 s_runQueue 中，之后调用 Get_Next_Runnable 函数找到优先级最高的进程当做当前运行的进程，从准备队列里取出并运行。

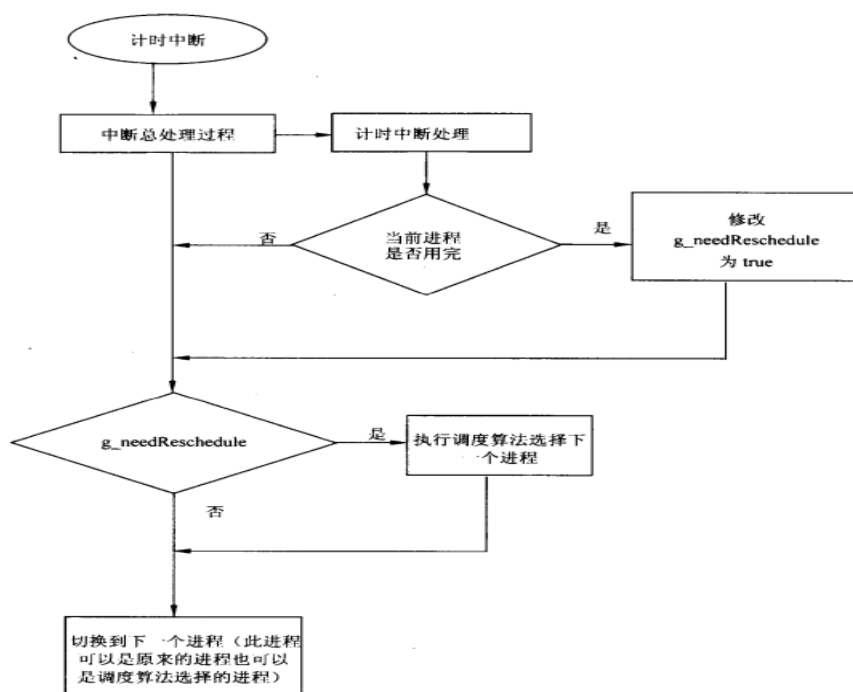
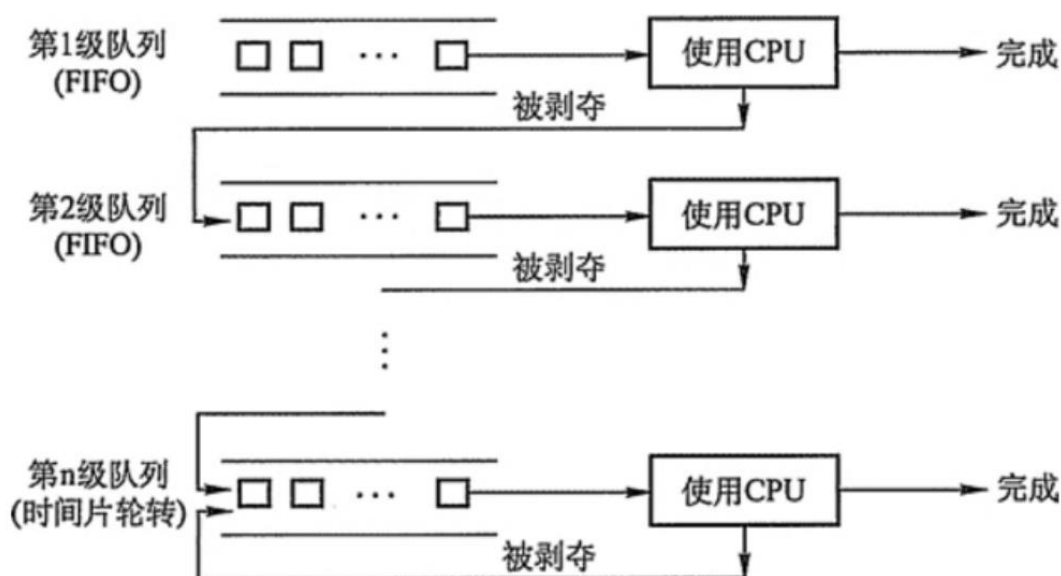


图 1.1 进程调度处理过程

如图 1.2，是多级反馈队列调度的基本流程，分为多个优先级不同的队列，新创建的进程首先进入优先级最高的准备队列，在给定的时间片内无法完成，则进入次一级队列的队尾，以此类推。对于系统中的空闲进程 Idle，应该始终放在最低优先级队列的队尾。



四级反馈队列调度策略实现的方法，首先用四个准备的运行队列替换系统的原来的一个队列，并为队列设置不同的优先级 0-3, 0 为最高优先级。修改 `s_runQueue` 的定义，将 `MAX_QUEUE_LEVEL` 设置为 4，构成结构体数组，结构体数组用于存放队列的队首指针。进程调度时，使用 `Sys_SetSchedulingPolicy` 函数选择使用 RR 还是 MLF 调度策略。使用 `Get_Next_Runnable` 函数从优先级为 0 的队列里开始寻找，具体使用 `Get_Front_Of_Thread_Queue` 函数找到应该调用的线程。

进程调度的优劣可以使用进程执行时间来衡量，即从创建到结束所花费的时间。在系统中可以通过调用 `Sys_GetTimeOfDay` 得到，分别在进程开始和结束时调用一次，获取全局变量 `g_numTicks` 的值，通过差值计算花费。

通过信号量和 PV 操作实现进程同步，其基本原理是，使用信号量标记存储使用该临界资源的进程信息，信号量只能由 PV 操作来改变。P 用来获取信号量，V 用来释放信号量，当执行 P 操作，信号量值减一，V 操作则相反，当信号量的值为 0 时进程阻塞。

具体实现通过 `Sys_P` 和 `Sys_V` 两个系统调用，当执行这两个系统调用操作信号量是，内核首先检测进程是否有操作权限，有则调用具体相应函数获得或者释放。`Sys_CreateSemaphore` 和 `Sys_DestroySemaphore` 分别用来创建和销毁信号量。

创建信号量时首先通过传入的参数获取信号量的名称，通过名称查找信号量是否在信号量队列，没有则创建信号量并加入队列，然后将进程注册加入到信号量的注册队列。当所有使用该信号量的进程都结束了，则销毁掉这个信号量。

项目 4——请求分页虚拟存储管理

为内核程序空间建立页表。要实现分页系统，第一步是修改用户的项目使用页表和段，这一步在函数 `Init_VM` 中完成，先分配一个页目录表，然后为整个存储区间的内容分配页表，从而给所有线性地址建立映射，每一个线性地址由页表来映射到物理地址。除此之外，对缺页的处理在分页存储管理中必不可少，需在函数 `Init_VM` 中加入缺页中断处理程序，通过调用 `Install_Interrupt_Handler` 函数来安装中断处理程序。

为用户进程建立页表，让用户进程拥有它们自己的线性地址空间一个用户进程的存储格式如图 3.4.1 所示。在 `uservm.c` 中修改用户进程在用户的存储区域使用分页，即对释放进程、装载可执行文件、用户缓冲区与内核缓冲区之间数据传输方式进行修改，这一步参考 `userseg.c` 完成。然后在 `Load_User_Progrm` 函数中分两步实现用户进程的分页系统，首先复制线性存储空间的低 2GB 的所有核心页目录表项到用户进程的页目录表，从而为进程分配一个页目录表；然后为用户进程的代码、数据和堆栈区间分配页表项，通过函数 `Alloc_Pageable_Page` 来分配这三个区间的页。由于用户模式进程的线性地址的基地址是 `0x8000 0000`，界限地址为 `0xFFFF FFFF`，所以为了能正常使用分页，需切换 `PDBR` 寄存器的内容，在装入 `LDT` 之后，函数 `Switch_To_Address_Space` 加入一个 `SET_PDBR` 调用作为上下文切换的一部分。

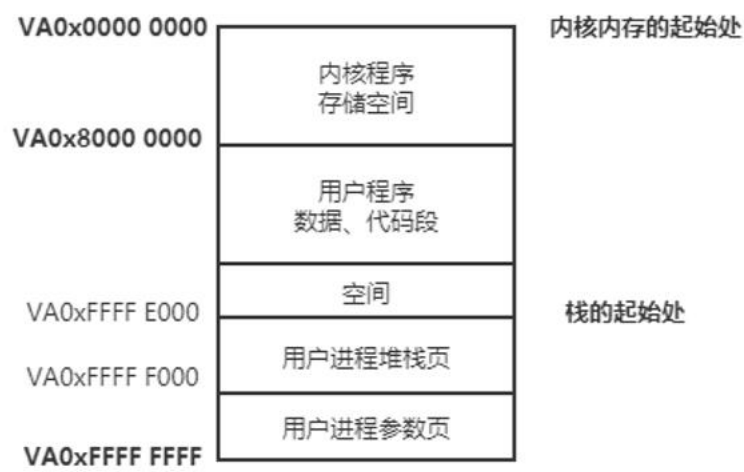


图 3.4.1 用户进程的存储格式

实现请求分页技术。使用虚拟存储技术可以把内存里的页暂时存储到磁盘上，从而使进程免受物理存储空间大小的制约，为了实现这一点操作系统将需要在磁盘设备上创建一个 `page file` 文件暂时保存从内存中替换出去的页，并实现一个类 `LRU` 算法在内存中选取一个替换页把它写到磁盘的 `page file` 文件中。交换一页到磁盘的操作步骤：确定要替换的页→在 `page file` 中找到空闲的存储空间→把被替换的页写到 `page file` 文件中→标识为此页在内存为不存在→修改页表项的页基地址→标识这一页是在磁盘上存在→刷新 `TLB`。当访问一个不在内存的页时，将引起缺页中断，缺页处理程序在不同情况下的处理如表 4.1 所示。

缺页情况	标识	相应处理
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 <code>page file</code> 中存在	从 <code>page file</code> 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

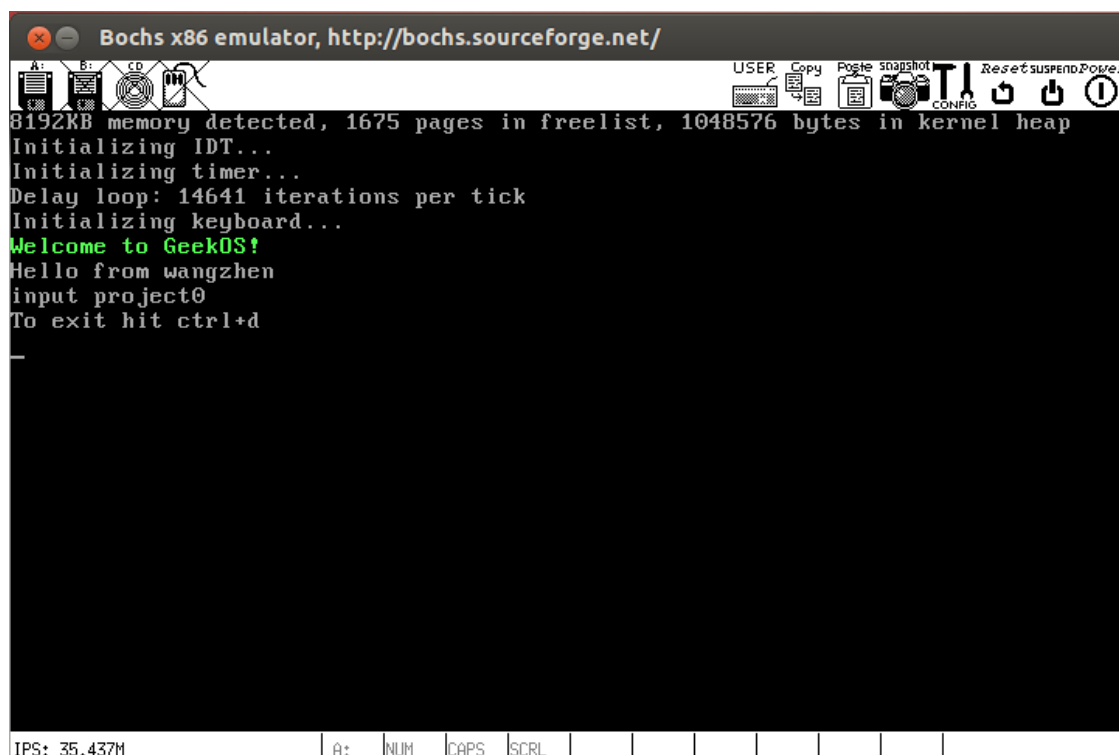
表 4.1 缺页处理表

处理进程终止。进程终止时，需释放占用的内存空间，修改 `Destroy_User_Context` 函数完成以下操作：释放进程的页、页表和页目录表对应的内存空间。

系统编译运行的结果及分析说明

项目 0——Geek0S 系统环境调试及编译

启动 geekOS，按下 input project0，屏幕上随着键盘按下而显示字符，这说明接受键盘输入的字符并显示在屏幕上的功能已实现。按下 ctrl+d 后，键盘按下，屏幕无反应，这说明进程已结束运行



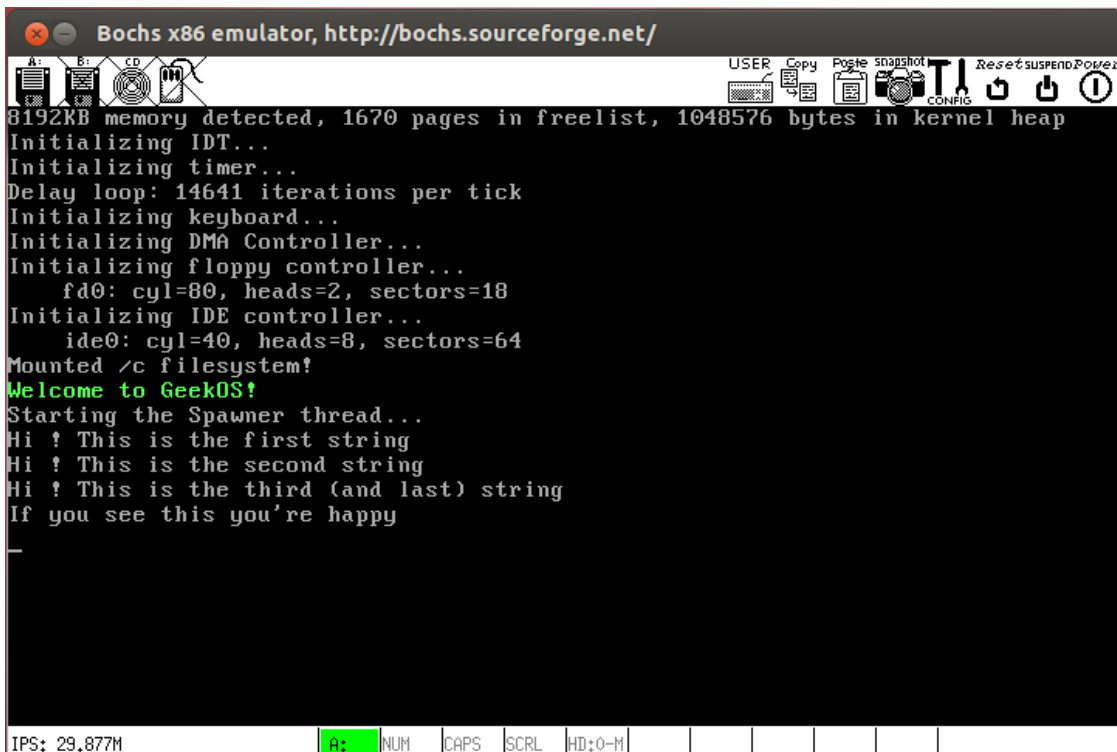
```
Bochs x86 emulator, http://bochs.sourceforge.net/
8192KB memory detected, 1675 pages in freelist, 1048576 bytes in kernel heap
Initializing IDT...
Initializing timer...
Delay loop: 14641 iterations per tick
Initializing keyboard...
Welcome to GeekOS!
Hello from wangzhen
input project0
To exit hit ctrl+d
-
IPS: 35.437M | A: | NUM | CAPS | SCRL | | | | | | | | | |
```

项目 1——内核级线程设计及实现

geekOS 启动后，程序就会运行到 project0/src/geekos/lprog.c 下的 void Spawner(unsigned long arg)。Spawner() 程序中会读取并运行 /c/a.exe，此 .exe 文件由 project0/src/user/a.c 编译而来。执行 /c/a.exe 的过程中，会在屏幕输出两句

话: "Hi ! This is the first string\n" 和 "Hi ! This is the second string\n" 。/c/a.exe 运行完毕后回到 Spawner() , 然后再 Print 两句: "Hi ! This is the third (and last) string\n" 和 "If you see this you're happy\n", 到此, 程序结束。

屏幕打印了/c/a.exe 中的 "Hi ! This is the first string\n" 和 "Hi ! This is the second string\n" , 并且 a.exe 运行完毕后回到 Spawner() , Print 两句: "Hi ! This is the third (and last) string\n" 和 "If you see this you're happy\n" , 说明内核级线程成功执行。



```
Bochs x86 emulator, http://bochs.sourceforge.net/
8192KB memory detected, 1670 pages in freelist, 1048576 bytes in kernel heap
Initializing IDT...
Initializing timer...
Delay loop: 14641 iterations per tick
Initializing keyboard...
Initializing DMA Controller...
Initializing floppy controller...
fd0: cyl=80, heads=2, sectors=18
Initializing IDE controller...
ide0: cyl=40, heads=8, sectors=64
Mounted /c filesystem!
Welcome to GeekOS!
Starting the Spawner thread...
Hi ! This is the first string
Hi ! This is the second string
Hi ! This is the third (and last) string
If you see this you're happy
-
```

项目 2——用户级进程的动态创建和执行

项目经过编译之后, 输入 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。由于 `Spawn_Init_Process(void)` 里面填写的是 shell 程序, Geekos 提供了一个简单的 shell, 保存在 PFAT 文件系统内, 所以 geekos 系统启动后, 启动 shell 程序/c/shell.exe 运行, 将/c/shell.exe 作为可执行文件传递给 spawn 函数的 program 参数, 创建第一个用户态进程, 然后由它来创建其他进程。运行后, geekos 就可以挂在这 shell, 并能运行测试文件 c.exe 和 b.exe。所以从 shell 开始执行, 输入 shell 程序中的相关命令, 即可得到不同结果。输入 pid, 系统返回的是 6, 是因为在系统初始化之初已经运行了 5 个进程, 因此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。Idle、Reaper 和 Main 三个进程, 它们由 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。

项目 3——线程调度算法和信号量功能实现

使用 make depend 和 make 编译，编译结果如图 1.4、1.5 所示：

```
stu@stu-desktop:~/project3-master/build$ make depend
perl ../scripts/generrs ../include/geekos/errno.h > libc/errno.c
gcc -M -O -Wall -Werror -g -DGEEKOS -I../include \
    ../src/geekos/idt.c ../src/geekos/int.c ../src/geekos/trap.c ../
src/geekos/irq.c ../src/geekos/io.c ../src/geekos/keyboard.c ../src/geekos/scree
n.c ../src/geekos/timer.c ../src/geekos/mem.c ../src/geekos/crc32.c ../src/geeko
s/gdt.c ../src/geekos/tss.c ../src/geekos/segment.c ../src/geekos/bget.c ../src/
geekos/malloc.c ../src/geekos/synch.c ../src/geekos/kthread.c ../src/geekos/user
.c ../src/geekos/userseg.c ../src/geekos/argblock.c ../src/geekos/syscall.c ../s
rc/geekos/dma.c ../src/geekos/floppy.c ../src/geekos/elf.c ../src/geekos/blockde
v.c ../src/geekos/ide.c ../src/geekos/vfs.c ../src/geekos/pfat.c ../src/geekos/b
itset.c ../src/geekos/main.c \
    | perl -n -e 's,^(\\S),geekos/$1;;print' \
    > depend.mak
gcc -M -O -Wall -Werror -I../include -I../include/libc \
    ../src/libc/sched.c ../src/libc/sema.c ../src/libc/compat.c ../s
rc/libc/process.c ../src/libc/conio.c libc/errno.c \
    | perl -n -e 's,^(\\S),libc/$1;;print' \
    >> depend.mak
gcc -M -O -Wall -Werror -I../include -I../include/libc \
    ../src/common/fmtout.c ../src/common/string.c ../src/common/memm
ove.c \
    | perl -n -e 's,^(\\S),common/$1;;print' \
    >> depend.mak
```

图 1.4 编译结果

```
gcc -O -Wall -Werror -I../include ../src/tools/buildFat.c -o tools/builtFat.exe
perl ../scripts/zerofile diskc.img 20480
tools/builtFat.exe diskc.img user/workload.exe user/semtest1.exe user/semtest2.exe user/p1.exe u
ser/p2.exe user/p3.exe user/schedtest.exe user/sched1.exe user/sched2.exe user/sched3.exe user/pi
g.exe user/pong.exe user/long.exe user/semtest.exe user/shell.exe user/b.exe user/c.exe
image file = diskc.img
first data blocks is 162
file workload.exe starts at block 162
file semtest1.exe starts at block 188
file semtest2.exe starts at block 212
file p1.exe starts at block 237
file p2.exe starts at block 261
file p3.exe starts at block 285
file schedtest.exe starts at block 309
file sched1.exe starts at block 333
file sched2.exe starts at block 357
file sched3.exe starts at block 380
file ping.exe starts at block 404
file pong.exe starts at block 429
file long.exe starts at block 454
file semtest.exe starts at block 479
file shell.exe starts at block 504
file b.exe starts at block 531
file c.exe starts at block 555
putting the directory at sector 161
rm libc/entry.o
stu@stu-desktop:~/project3-master/build$
```

图 1.5 编译结果

```
$ schedtest rr 1
```

图 1.6

图 1.7

信号量操作测试结果如图 1.6 所示

图 1.6

图 1.6 信号量操作结果

项目 4——请求分页虚拟存储管理

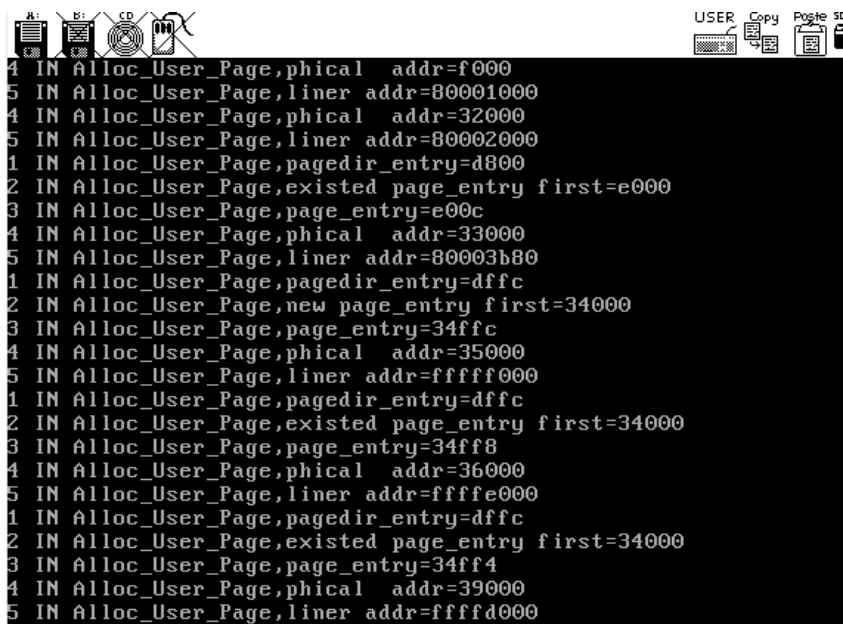


图 3.4.2 运行结果(1)

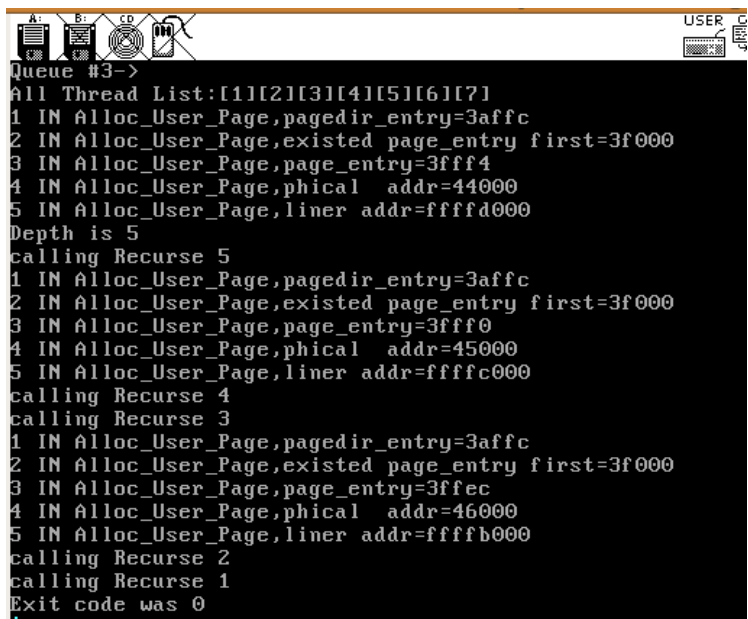


图 3.4.3 运行

结果 (2)

如图 3.4.2 所示，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。

如图 3.4.3 所示，输入命令 rec 5 后，可以得到迭代递归下的 Project4 运行截图所示结果。由结果可以看出，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 3fff4, 进程 4 显示对应物理地址为 44000, 进程 5 显示当前线性地址为 ffffd000; 最后输出此次迭代递归的搜索深度为 5, 在这个过程中一共递归调用过 5 次进程, 分别是调用递归进程 5、进程 4、进程 3、进程 2, 和进程 1。

遇到的问题及解决方法

心得体会

周志兆:

在完成了本次操作系统的课程设计之后, 我深深地体会到了开设这门课是非常有必要的。在上学期, 我们学了《操作系统教程》这门操作系统的理论课程。知道了操作系统是管理计算机系统的全部硬件资源包括软件资源及数据资源; 控制程序运行; 改善人机界面; 为其它应用软件提供支持等, 使计算机系统所有资源最大限度地发挥作用, 为用户提供方便的、有效的、友善的服务界面。同时也懂得了计算机操作系统是铺设在计算机硬件上的多层系统软件, 不仅增强了系统的功能, 而且还隐藏了对硬件操作的细节, 由它实现了对计算机硬件操作的多层次的抽象。但是, 学完理论知识, 我们对操作系统的概念只是停留在理论的层面而已, 并没有真正的体会到操作系统运行的整个流程。所以操作系统课程设计这门课程, 通过编程实验, 更加深入得理解和掌握操作系统的基本理论和功能技术, 将相对抽象的理论应用于实践, 提高分析问题和解决问题的能力, 提高编写和开发系统程序的能力。

在我看来, 理论与实际相结合是很重要的, 只有理论知识是远远不够的, 只有把所学的理论知识与实践结合起来, 从理论中得出结论, 才能真正为社会服务, 从而提高自己的实际动手能力和独立思考的能力。这一次的课程设计的收获还是很大的。发现问题, 查找资料, 解决问题的能立得到了很大的提升。学会了很多学习的方法。我觉得这是以后最实用的方法, 受益匪浅。其次独立思考、动手操作的能力也得到了加强。通过自己去动手实践, 加深了对操作系统的理论知识的认知。还有就是团队协作的能立也得到了很好的锻炼。本次的课程设计, 是以小组的形式开展的, 在课程的工程中, 我们小组内遇到问题会主动进行沟通交流, 相互讨论, 最后得到一个解决的方案, 这种能力在未来的学习和工作中也是非常重要的。

袁嘉鸿:

项目 4 主要涉及到操作系统中存储管理这部分的内容, 特别是要先掌握分页存储管理、多级页表、分页系统的地址转换机制、请求分页虚拟存储管理这些知识点, 但由于对这些知识已经生疏, 导致一开始对项目 4 毫无头绪, 在重新把操作系统存储管理的内容复习了一遍, 并把老师给的计算

机操作系统实践教程上关于项目 4 的内容看了一遍后，才对项目 4 的请求分页虚拟存储管理的具体实现技术有了一个清晰的认识，本次课设给我的体会就是扎实的理论基础必不可少，并且通过这次课设，我对操作系统中请求分页虚拟存储管理等知识有了更为深刻的认识。

参考文献

- [1] 李志欣, 魏海洋, 黄飞成, 等. 结合视觉特征和场景语义的图像描述生成[J]. 计算机学报, 2020, 43(9): 1624-1640.
- [2] 黄廷辉, 王宇英. 计算机操作系统实践教程[M]. 北京: 清华大学出版社, 2007.
- [3] 周飞燕, 金林鹏, 董军. 卷积神经网络研究综述[J]. 计算机学报, 2017, 40(06): 1229-1251.

参考文献格式供参考