

**操作系统与安全**

**课程设计报告**

**学院名称 信息科学与技术学院**

**专业名称**

**学生姓名**

**学生学号**

**指导教师**

**实验地点 7203**

**2020年 12 月**

**说明**

1. 封面按照给定的模板对应填写相关内容；
2. 要求整个实验报告标上页码，生成一个目录；
3. 两个课程设计要按模板中所规定的内容顺序展开；
4. 最后一页学生签名的时间统一为2020年12月30日；
5. 格式要求：
6. 正文用宋体小四号，1.5倍行距，页边距采取默认形式（上下2.54cm，左右2.54cm，页眉1.5cm，页脚1.75cm）。字符间距为默认值（缩放100%，间距：标准）；页码用小五号字底端居中；
7. 程序代码书写要规范（5号、单倍行距、段前段后行距为0，尽量添加注释）；
8. 版面布局排版一定要界面友好；
9. 其他要求：

**目录：**自动生成，小四号1.5倍行距；

（提交的正式报告中此说明页要删除）

设计项目一：作业调度算法的模拟实现

一、实现目的

1）理解BIOS中断调用、系统调用以及C语言标准库函数的联系和区别

2）理解Linux API和系统调用的区别

3）熟悉Linux下进程控制相关的系统调用，并熟悉使用相关函数完成进程控制的操作

4）学习写makefile文件

二、实验准备

1）了解Linux编译工具和调试工具的使用方法

2）查阅相关资料，掌握阅读和编写makefile文件的能力

3）查阅资料，了解进程控制的相关理论知识

三、实验基本知识及原理

1.系统调用

操作系统的主要功能是为应用程序的运行创建良好的环境。为了达到这个目的，内核提供了一系列具备预订功能的多内核函数，通过一组称为系统调用（system call）的接口呈现给用户。系统调用将应用程序的请求传给内核，调用相应的内核函数完成所需的处理，然后将处理结果返回给应用程序，如果没有系统调用和内核函数，用户将不能编写大型应用程序。

Linux提供系统调用，使用户进程能够调用内核函数。这些系统调用允许用户操纵进程、文件和其他系统资源，从用户级切换到内核级。也就是说，系统调用的执行会引起特权级的切换，是一种受约束的、为切换到保护核心的“函数调用”。普通函数调用不会引起特权级的转换，一般不受约束。

2.BIOS中断调用

BIOS中断服务程序实质上是微机系统中软件与硬件之间的一个可编程接口，主要用于程序软件功能与微机硬件之间的连接。BIOS中断服务“封装”了许多系统底层的细节，使得一些用户程序也能够使用BIOS功能。

3.C语言标准库

C语言标准库是利用系统调用来实现的，它依赖于系统的系统调用封装起来，而对开发者透明。系统调用的实现在内核完成，而C语言标准库则在用户态实现，标准库函数完全运行在用户空间。

4.API和系统调用的区别

API（Application Programming Interface，应用编程接口）是程序员在用户空间下可以直接使用的函数接口，如常用的read()，malloc()，free()函数等。这些函数用来获得一个给定的服务。系统调用是通过软中断向内核发出一个明确的请求。API和系统调用并没有严格的对应关系：

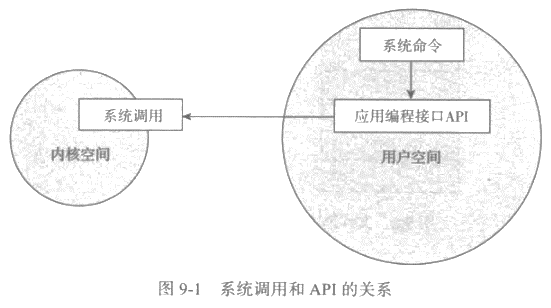
1）API有可能和系统调用的形式是一样的。比如API的read()函数就和read()系统调用的形式是一致的。

2）几个不同的API的内部实现可能是调用同一个系统调用。例如，Linux的libc库实现了内存分配和释放函数malloc()，calloc()和free()，这几个函数的实现都调用了brk()系统调用。

3）一个API的功能可能并不需要系统调用，如abs()。

4）一个API的功能实现需要多个系统调用。

系统调用与API的关系可以用下图9-1来表示。



5，makefile文件和make命令

一个工程中的源文件可能不计其数，按类型、功能、模块分别放在若干目录中，makefile定义了一系列的规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至进行更复杂的功能操作。makefile就像一个shell脚本一样，也可以执行操作系统的命令。makefile文件需要按照某种语法进行编写，文件中需要说明如何编译各个源文件并链接生成可执行文件，以及定义文件间的依赖关系。

make是一个命令工具，即解释makefile中指令的命令工具，一般来说，大多数IDE都有这个命令，如Delph的make，VC的nmake，GNU的make。

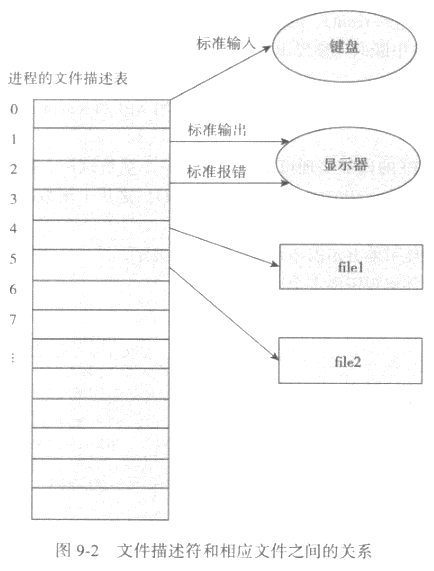
makefile带来的好处就是“自动化编译”，程序一旦写好，只需要一个make命令，整个工程就可以自动编译，极大提高了软件开发的效率。

6.文件描述符

内核利用文件描述符来访问文件。文件描述符是非负整数。打开或新建文件时，内核会返回一个文件描述符。读写文件也需要使用文件描述符来指定待读写的文件。

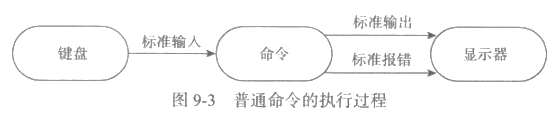
习惯上，标准输入的文件描述符是0，标准输出的文件描述符是1，标准错误的文件描述符是2，注意，这种使用方式不是UNIX内核的特性，但是因为一些shell和应用程序都采用这种方式，所以如果内核不遵循这种习惯的话，很多应用程序将不能使用。

标准文件和文件描述符的关系可用图9—2来表示。

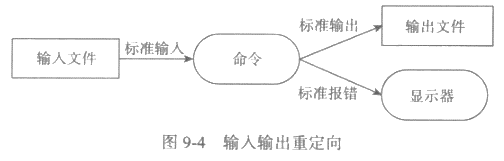


7.输入输出重定向

通常来讲，输入默认为键盘输入，输出默认为输出到屏幕，一条命令的执行语义如下图9-3所示。



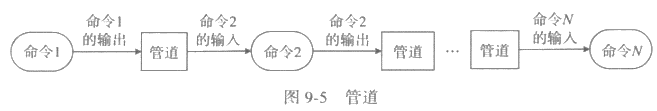
输入输出重定向就是改变输入输出的方向，如将标准输入和输出都改为文件，如下图9-4所示。



8.进程通信机制——管道

管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立两个管道；管道只能用于具有“亲缘”关系的进程（父子进程或者兄弟进程之间）；管道单独构成一种独立的文件系统：管道对于管道两端的进程而言就是一个文件，但它不是普通的文件，不属于某种文件系统，而是自立门户，并且只存在于内存中。

一个进程向管道中写的内容被管道另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾，并且每次都由缓冲区的头部读出数据，如下图9—5所示。



四、实验说明

本实验在Linux环境下运行

1.重要函数说明

（1）调用fork函数创建子进程

**头文件：**

#include<unistd.h>

#include<sys/types.h>

**函数原型：**

pid\_t fork(void); //pid\_t是一个宏定义，其实质是int

**返回值：**若成功调用一次则返回两个值，子进程返回0，父进程返回子进程id；出错返回-1。

Linux下的fork函数与Windows下的\_spawnl函数的作用都是创建子进程，它们的区别是：fork将进程代码复制一份并执行，\_spawnl从头开始执行。

（2）pipe函数介绍

**头文件：**

#include<unistd.h>

**函数原型：**

int pipe(int fildes[2]);

**返回值：**成功返回0，失败返回-1.

功能：参数fildes用来描述管道的两端，管道被创建时，两端的任务也是确定的。fildes[0]是管道的读出端，fildes[1]是管道的写入端。

（3）dup2函数介绍

**头文件：**

#include<unistd.h>

**函数原型：**

int dup2(int oldfd, int targetfd);

dup2函数允许调用者规定一个有效描述符oldrd和一个目标描述符targetfd， dup2函数成功返回时，目标描述符（dup2函数的第二个参数）将变成源描述符（dup2函数的第一个参数）的复制品，换句话说，两个文件描述符现在都指向同一个文件，并且是函数第一个参数指向的文件。

**功能：**将oldfd文件描述符复制到targefd，使oldfd和targefd指向同一文件。

2.makefile文件

make命令执行时，需要一个makefile文件，以告诉make命令如何编译和链接。makefile的规则如下：

target … : prerequisites …

command

…

…

target是目标文件：prerequisites是要生成target所需要的文件或目标；command是使用prerequisites生成target的命令，即make需要执行的命令。这是一个文件的依赖关系，target包括的一个或多个目标文件依赖于prerequisites中的文件，其生成规则定义在command中。例如，执行一个最简单的程序：

//helloworld.c

#include<stdio.h>

{

printf("Hel1o world\n");

return 0;

}

在Linux命令行下直接使用gcc命令可以方便地将该程序编译为可执行文件。在此，为了讲解makefile的使用，将使用makefile的方式进行编译。

写好的makefile文件如下：

#makefile

al1:helloworld

hel1oworld: hel1oworld.o

gcc -o helloworld helloworld.o

hel1oworld.o: hel1oworld.c

gcc —c hel1oworld.c

clean：

rm helloworld.o

all表示最终结果是什么，在这个makefile中即为生成的可执行文件helloworld。

helloworld表示它是由helloworld.o文件而来，由helloworld.o生成helloworld的命令即为：gcc -o helloworld helloworld。

接下来的语句指明了helloworld.。的出处：它是依赖于helloworld.c，通过如下命令得到的：

gcc -c helloworld.c

最后，因为我们生成的中间文件helloworld.o对于用户使用是没有作用的，所以要将其删除，完成该项工作的语句是：

clean：

rm helloworld.o

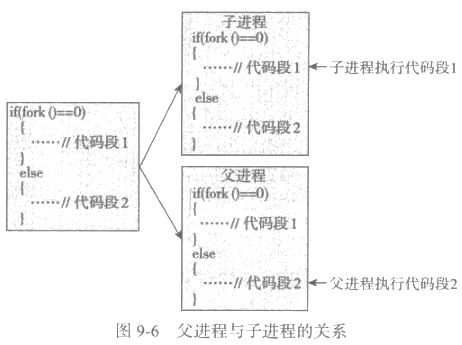
这样，makefile文件就写好了，此时，切换到makefile所在的目录执行make命令。make命令会找到makefile文件，以all语句的内容为最终目标，通过依赖关系依次往下执行，这是因为clean与之前的语句并不存在依赖关系，所以clean的内容无法执行。如果用户希望执行该语句，可以使用make clean语句来实现，或者将makefile的第一句改为"all：helloworldclean"。

3.源码说明

在参考代码中的程序包括signal.c， fork.c. pipe.c三个程序文件，下面对其进行说明。

（1）程序代码fork.c

该程序中使用fork系统调用创建了一个子进程来执行相关命令，其中，fork创建子进程的部分示意图如图9-6所示。

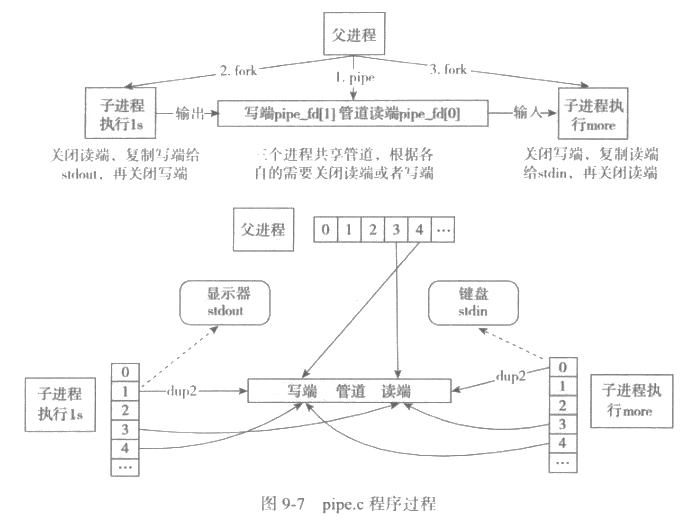


子进程是父进程的一个副本，它会获得父进程所有资源的副本，即子进程拥有和父进程相同代码段的内存块。fork函数被调用一次但返回两次。子进程复制了父进程的堆栈段，所以两个进程在执行过程中都停留在fork函数中等待返回。事实上，fork函数在父进程和子进程各返回一次。在子进程中返回0值，而父进程返回子进程的id。

调用fork之后，数据、堆栈有两份，代码仍为一份，但是该代码段为两个进程的共享代码段，都从fork函数中返回，箭头表示各自的执行处。当父、子进程有一个想要修改代码段时，两个进程真正分裂。

(2)程序代码pipe.c

该程序中，系统调用pipe和dup2函数来模拟实现shell命令1s -1 /etc/ | more，其示意图如图9-7所示。



管道是在内存中实现的，从10重定向的观点来看，等同于以下命令的组合：

1s -1 /etc/ > temp //（输出重定向到temp）

more < temp //(或者more temp)

rm temp //(删除临时文件)

该命令组合需要3条命令和1个临时文件，且含有磁盘1/0操作，执行效率比使用管道低。

五、实验内容

1）登录Linux系统

2）在home目录下建立以自己学号为文件名的文件

3）复制实验提供的源代码至自己建立的文件

4）阅读关于系统调用fork、exec、wait、exit、pipe等函数

5）编写makefile，用make编译源代码中的fork.c和pipe.c

6）运行步骤5生成的可执行文件，观察结果及进程

7）查阅资料，掌握信号和进程调度，阅读signal.c代码，编译并运行，另开终端，用ps和kill命令终结进程

六、完整代码

七、实验结果（附上运行结果的截图）

八、实验总结分析

设计项目二：页面调度算法的模拟实现

（以下红色字部份正式报告中要删除）

（空一行）

一、实现目的

二、实验准备

三、实验基本知识及原理

四、实验说明

五、实验内容（要求画出实验流程图）

六、完整代码（提交算法实现的程序程序，尽量给些注释）

七、实验结果（附上运行结果的截图）

八、实验总结分析

（请参照下面教材P98页，实验指导3的相关内容）



|  |  |
| --- | --- |
| **学生实验 心得** | （必须填写，总结通过课程设计实验后，对相关知识的心得和收获）  学生（签名）：  年 月 日 |
| **指导**  **教师**  **评语** | 成绩评定：  指导教师（签名）：  年 月 日 |

2.实验心得及教师评语