《计算机操作系统》
实验指导书

湖南工业大学计算机学院

二０一六 年九月

前言

计算机操作系统是计算机科学与技术、软件工程、网络工程等专业的主要专业基础课程，其实践性、应用性很强。实践教学环节是必不可少的一个重要环节。计算机操作系统的实验目的是加深对理论教学内容的理解和掌握，使学生较系统地掌握操作系统的基本原理，加深对操作系统基本方法的理解，加深对课堂知识的理解，为学生综合运用所学知识，在 Linux 环境下调用一些常用的函数编写功能较简单的程序来实现操作系统的基本方法、并在实践应用方面打下一定基础。要求学生在实验指导教师的帮助下自行完成各个操作环节，并能实现且达到举一反三的目的，完成一个实验解决一类问题。要求学生能够全面、深入理解和熟练掌握所学内容，并能够用其分析、设计和解答类似问题；对此能够较好地理解和掌握，并且能够进行简单分析和判断；能够熟练使用 Linux 用户界面；掌握操作系统中进程的概念和控制方法；了解进程的并发，进程之间的通信方式，了解虚拟存储管理的基本思想。同时培养学生进行分析问题、解决问题的能力；培养学生完成实验分析、实验方法、实验操作与测试、实验过程的观察、理解和归纳能力。

为了收到良好的实验效果，编写了这本实验指导书。在指导书中，每一个实验均按照该课程实验大纲的要求编写，力求紧扣理论知识点、突出设计方法、明确设计思路，通过多种形式完成实验任务，最终引导学生有目的、有方向地完成实验任务，得出实验结果。任课教师在实验前对实验任务进行一定的分析和讲解，要求学生按照每一个实验的具体要求提前完成准备工作，如：查找资料、设计程序、完成程序、写出预习报告等，做到有准备地上机。进行实验时，指导教师应检查学生的预习情况，并对调试过程给予积极指导。实验完毕后，学生应根据实验数据及结果，完成实验报告，由学习委员统一收齐后交指导教师审阅评定。

实验成绩考核：

实验成绩占计算机操作系统课程总评成绩的 20%。指导教师每次实验对学生进行出勤考核，对实验效果作记录，并及时批改实验报告，综合评定每一次的实验成绩，在学期终了以平均成绩作为该生的实验成绩。有以下情形之一者，实验成绩为不及格：

1. 迟到、早退、无故缺勤总共 3 次及以上者；
2. 未按时完成实验达 3 次及以上者；
3. 缺交实验报告 2 次及以上者。

目　录

第一部分 实验环境及所用系统函数介绍

DOS 操作系统是单道操作系统，无法进行多道程序设计，Windows 环境下的 Visual C++虽然也可用于多道程序设计，但是并不是完全的多道，因为它也可用于 DOS 编程。所以我们的实验选用 Linux 操作系统。Linux 操作系统是一个类 UINX 操作系统，是一个纯多道并发的网络操作系统。

1.1 Linux 操作系统简介

Linux 是一个多用户操作系统，是 UNIX 的一个克隆版本(界面相同但内部实现不同)，同时它是一个自由软件，是免费的、源代码开放的，这是它与 UNIX 的不同之处。现在，Linux 凭借优秀的设计，不凡的性能，加上 IBM、Intel、CA、CORE、Oracle 等国际知名企业的大力支持，市场份额逐步扩大，已成为与 Windows 和 UNIX 并存的三大主流操作系统之一。

1.2 常用文件及目录操作命令

Linux 系统信息存放在文件里，文件与普通的公务文件类似。每个文件都有自己的名字、内容、存放地址及其它一些管理信息，如文件的用户、文件的大小等。文件可以是一封信、一个通讯录，或者是程序的源语句、程序的数据，甚至可以包括可执行的程序和其它非正文内容。 Linux 文件系统具有良好的结构，系统提供了很多文件处理程序。这里主要介绍常用的文件处理命令。

1.2.1 ls 命令

1.功能

ls 命令用于显示目录内容，类似 DOS 下的 dir 命令，它的使用权限是所有用户。

2.格式

ls [options][filename]

3.options 主要参数

－a, －－all：列出所有项目，包括以“.” 字符开始的项目。

－A, －－almost－all：列出除了“ . ”及 “.. ”以外的任何项目。

－－author：印出每个文件作者。

－b, －－escape：以八进制溢出序列表示不可打印的字符。

－－block－size=大小：块以指定<大小>的字节为单位。

－B, －－ignore－backups：不列出任何以 ~ 字符结束的项目。

－f：不进行排序，－aU 参数生效，－lst 参数失效。

－F, －－classify：加上文件类型的指示符号 (*/=@| 其中一个)。

－g：与－l 类似，但不列出文件所有者

－G, －－no－group：以长列表形式列出，不显示用户组名称。

－i, －－inode：列出每个文件的 inode 号。

－I, －－ignore=样式：不印出任何符合 Shell 万用字符<样式>的项目。

－k：即－－block－size=1K。

－l：使用较长格式列出信息。

－L, －－dereference：当显示符号链接的文件信息时，显示符号链接所指示的对象，而并非符号链接本身的信息。

－m：所有项目以逗号分隔，并填满整行行宽。

－n, －－numeric－uid－gid：类似－l，但列出 UID 及 GID 号。

－N, －－literal：列出未经处理的项目名称，例如不特别处理控制字符。

－p, －－indicator-style=slash：在目录后加/指示符－Q, －－quote－name：将项目名称括上双引号。

－r, －－reverse：依相反次序排列。

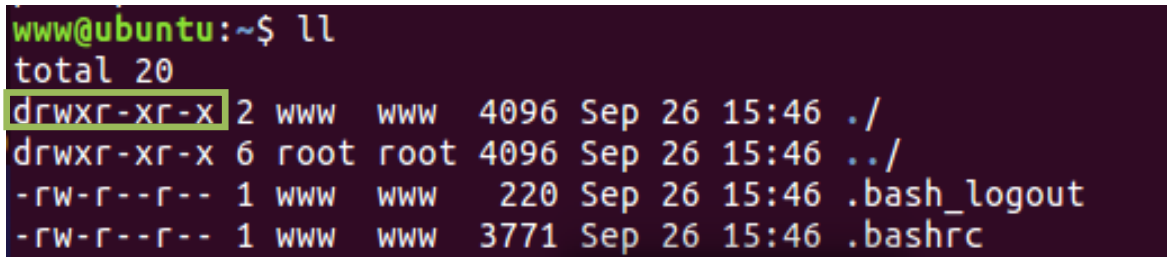
－R, －－recursive：同时列出所有子目录层。

－s, －－size： 列出每个文件所占块数

4.应用举例

ls 命令是 Linux 系统使用频率最多的命令，它的参数也是 Linux 命令中最多的。使用 ls 命令时会有几种不同的颜色，其中蓝色表示是目录，绿色表示是可执 行文件，红色表示是压缩文件，浅蓝色表示是链接文件，加粗的黑色表示符号链接，灰色表示是其它格式文件。

如下图所示，文件属性列表中，第一列文件属性开头是由 10 个字符构成的字符串。其中第一个字符表示文件类型，它可以是下述类型之一：－（普通文件）、d（目录）、l（符号链接）、 b（块设备文件）、c（字符设备文件）。后面的 9 个字符表示文件的访问权限，分为 3 组，每组 3 位。第一组表示文件属主的权限，第二组表示同组用户的权限， 第三组表示其他用户的权限。每一组的三个字符分别表示对文件的读（r）、写（w）和执行权限（x）。对于目录，表示进入权限。



1.2.2 cp 命令

1. 功能

复制文件或目录

2. 说明

cp 指令用于复制文件或目录，如同时指定两个以上的文件或目录，且最后的目的地是一个已经存在的目录，则它会把前面指定的所有文件或目录复制到此目录中。若同时指定多个文件或目录，而最后的目的地并非一个已存在的目录，则会出现错误信息。

3. 参数:

-a 或 --archive 此参数的效果和同时指定"-dR –preserve=all"参数相同

-b 或 --backup 删除、覆盖目的文件先备份，备份的文件或目录亦建立为符号链接，并指向源文件或目录链接的源文件或目录。假如没有加上这个参数，在复制过程中若遇到符号链接，则会直接复制源文件或目录

-f 或 --force 强行复制文件或目录， 不论目的文件或目录是否已经存在

-i 或 --interactive 覆盖文件之前先询问用户

-l 或 --link 对源文件建立硬链接，而非复制文件

-p 或 --preserve 保留源文件或目录的属性，包括所有者、所属组、权限与时间

-R, -r 或 --recursive 递归处理，将指定目录下的文件及子目录一并处理

-s 或 --symbolic-link 对源文件建立符号链接，而非复制文件

-S <备份字尾字符串> 或 --suffix=<备份字尾字符串> 用"-b"参数备份目的文件后，备份文件的字尾会被加上一个备份字符串。默认的备份字尾符串是符号"~"

-u 或 --update 使用这项参数之后，只会在源文件的修改时间(Modification Time)较目的文件更新时，或是名称相互对应的目的文件并不存在，才复制文件

-v 或 --verbose 显示执行过程

-x 或 --one-file-system 复制的文件或目录存放的文件系统必须与 cp 指令执行时所处的文件系统相同，否则不复制，亦不处理位于其他分区的文件

--help 显示在线帮助

--sparse=<使用时机> 设置保存稀疏文件的时机

--version 显示版本

4. 示例:

.复制文件，只有源文件较目的文件的修改时间新时，才复制文件

cp -u -v file1 file2

1.2.3 mv 命令

1.功能

mv 命令用来为文件或目录改名，或者将文件由一个目录移入另一个目录中，它的使用权限是所有用户。该命令如同 DOS 命令中的 ren 和 move 的组合。

2.格式

mv[options] 源文件或目录 目标文件或目录

3.[options]主要参数

－i：交互方式操作。如果 mv 操作将导致对已存在的目标文件的覆盖，此时系统询问是否重写，要求用户回答“y”或“n”，这样可以避免误覆盖文件。

－f：禁止交互操作。mv 操作要覆盖某个已有的目标文件时不给任何指示，指定此参数后 i 参数将不再起作用。

4.应用实例

（1）将/usr/cbu 中的所有文件移到当前目录（用“.”表示）中：

$ mv /usr/cbu/ * .

（2）将文件 cjh.txt 重命名为 wjz.txt：

$ mv cjh.txt wjz.txt

1.2.4 rm 命令

1 功能

删除一个目录中的一个或多个文件或目录，它也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，只是删除了链接，原有文件均保持不变。rm 是一个危险的命令，使用的时候要特别当心，尤其对于新手，否则整个系统就会毁在这个命令（比如在/（根目录）下执行 rm * -rf)。所以，我们在执行 rm 之前最好先确认一下在哪个目录，到底要删除什么东西，操作时保持高度清醒的头脑。

2．命令格式：

rm[选项]文件…

3．命令参数：

-f, --force　　忽略不存在的文件，从不给出提示。

-i, --interactive 进行交互式删除

-r, -R, --recursive　　指示 rm 将参数中列出的全部目录和子目录均递归地删除。

-v,--verbose　详细显示进行的步骤

-help　　显示此帮助信息并退出

-version　输出版本信息并退出

4．命令实例：

实例一：删除文件 file，系统会先询问是否删除。

命令：

rm 文件名

输出：

[root@localhost test1]# ll

总计 4

-rw-r--r-- 1 root root 56 10-26 14:31 log.log

root@localhost test1]# rm -i log.log

rm：是否删除 一般文件 “log.log”? y

root@localhost test1]# ll

总计 0[root@localhost test1]#

说明：输入 rm log.log 命令后，系统会询问是否删除，输入 y 后就会删除文件，不想删除则数据 n。

1.2.5 mkdir 命令

1.作用

mkdir 命令的作用是建立名称为 dirname 的子目录，与 MS DOS 下的 md 命令类似，它的使用权限是所有用户。

2.格式

mkdir [options] 目录名

3.[options]主要参数

－m, －－mode=模式：设定权限<模式>，与 chmod 类似。

－p, －－parents：需要时创建上层目录；如果目录早已存在，则不当作错误。

－v, －－verbose：每次创建新目录都显示信息。

－version：显示版本信息并退出。

1.2.6 rmdir 命令

1.功能

删除空目录，一个目录被删除之前必须是空的。(注意，rm - r dir 命令可代替 rmdir，但是有很大危险性。）删除某目录时也必须具有对父目录的写权限。

2．命令格式：

rmdir [选项]... 目录...

3．命令参数：

- p 递归删除目录 dirname，当子目录删除后其父目录为空时，也一同被删除。如果整个路径被删除或者由于某种原因保留部分路径，则系统在标准输出上显示相应的信息。

-v, --verbose 显示指令执行过程

4．命令实例

实例一：rmdir 不能删除非空目录

命令：rmdir doc

1.2.7 cd 命令

1. 命令功能：

切换当前目录至 dirName

2. 命令格式：

cd [目录名]

3. 常用范例

例：进入系统根目录

命令：cd /

输出：[root@localhost ~]# cd /

说明：进入系统根目录,上面命令执行完后拿 ls 命令看一下，当前目录已经到系统根目录了

命令：cd .. 或者 cd .. //

说明：进入系统根目录可以使用“ cd .. ”一直退，就可以到达根目录

1.2.8 file 命令

1. 功能：根据文件内容判断文件类型，使用权限是所有用户。

2. 格式

file [options] 文件名

3.[options]主要参数

-v：在标准输出后显示版本信息，并且退出。

-z：探测压缩过的文件类型。

-L：允许符合连接。

-f name：从文件 namefile 中读取要分析的文件名列表。

4.简单说明

使用 file 命令可以知道某个文件究竟是二进制（ELF 格式）的可执行文件, 还是 Shell Script 文件，或者是其它的什么格式。file 能识别的文件类型有目录、Shell 脚本、英文文本、二进制可执行文件、C 语言源文件、文本文件、DOS 的可执行文件。

1.2.9 pwd 命令

1. 功能

查看”当前工作目录“的完整路径。每当在终端进行操作时，都会有一个当前工作目录。 在不太确定当前位置时，就会使用 pwd 来判定当前目录在文件系统内的确切位置。

2．命令格式：

pwd [选项]

3．常用参数：

一般情况下不带任何参数

如果目录是链接时：

格式：pwd -P　显示出实际路径，而非使用连接（link）路径。

4．常用实例：

实例：用 pwd 命令查看默认工作目录的完整路径

命令：

pwd

输出：

[root@localhost ~]# pwd

/root

[root@localhost ~]#

1.2.10 cat 命令

cat 命令的用途是连接文件或标准输入并打印。这个命令常用来显示文件内容，或者将几个文件连接起来显示，或者从标准输入读取内容并显示，它常与重定向符号配合使用。

1．功能：

cat 主要有三大功能：

1）一次显示整个文件:cat filename

2）从键盘创建一个文件:cat > filename 只能创建新文件,不能编辑已有文件.

3）将几个文件合并为一个文件:cat file1 file2 > file

2．命令格式：

cat [选项] [文件]...

3．命令参数：

-A, --show-all 等价于 -vET

-b, --number-nonblank 对非空输出行编号

-e 等价于 -vE

-E, --show-ends 在每行结束处显示 $

-n, --number 对输出的所有行编号,由 1 开始对所有输出的行数编号

-s, --squeeze-blank 有连续两行以上的空白行，就代换为一行的空白行

-t 与 -vT 等价

-T, --show-tabs 将 TAB 字符显示为 ^I

-u (被忽略)

-v, --show-nonprinting 使用 ^ 和 M- 引用，除了 LFD 和 TAB 之外

4．使用实例：

把 log2012.log 的文件内容加上行号后输入 log2013.log 这个文件里

命令：cat -n log2012.log log2013.log

1.2.11 find 命令

1. 功能

find 命令的作用是在目录中搜索文件，它的使用权限是所有用户。

2. 格式

find [path][options][expression]

path 指定目录路径，系统从这里开始沿着目录树向下查找文件。它是一个路径列表，相互用空格分离，如果不写 path，那么默认为当前目录。

3. 主要参数

[options]参数：

－depth：使用深度级别的查找过程方式，在某层指定目录中优先查找文件内容。

－maxdepth levels：表示至多查找到开始目录的第 level 层子目录。level 是一个非负数，如果 level 是 0 的话表示仅在当前目录中查找。

－mindepth levels：表示至少查找到开始目录的第 level 层子目录。

－mount：不在其它文件系统（如 Msdos、Vfat 等）的目录和文件中查找。

－version：打印版本。

[expression]是匹配表达式，是 find 命令接受的表达式，find 命令的所有操作都是针对表达式的。它的参数非常多，这里只介绍一些常用的参数。

—name：支持通配符*和?。

－atime n：搜索在过去 n 天读取过的文件。

－ctime n：搜索在过去 n 天修改过的文件。

－group grpoupname：搜索所有组为 grpoupname 的文件。

－user 用户名：搜索所有文件属主为用户名（ID 或名称）的文件。

－size n：搜索文件大小是 n 个 block 的文件。

－print：输出搜索结果，并且打印。

4. 应用技巧

find 命令查找文件的几种方法：

（1）根据文件名查找

例如，我们想要查找一个文件名是 lilo.conf 的文件，可以使用如下命令：

find / －name lilo.conf

find 命令后的“/”表示搜索整个硬盘。

（2）快速查找文件

根据文件名查找文件会遇到一个实际问题，就是要花费相当长的一段时间，特别是大型 Linux 文件系统和大容量硬盘文件放在很深的子目录中时。如果我们知道了这个文件存放在某个目录中，那么只要在这个目录中往下寻找就能节省很多时间。比如 smb.conf 文件，从它的文件后缀“.conf”可以判断这是一个配置文件，那么它应该在/etc 目录内，此时可以使用下面命令：

find /etc －name smb.conf

这样，使用“快速查找文件”方式可以缩短时间。

（3）根据部分文件名查找方法

有时知道某个文件包含有 abvd 这 4 个字，那么要查找系统中所有包含有这 4 个字符的文件可以输入下面命令：

find / －name '*abvd*'

输入这个命令以后，Linux 系统会将在/目录中查找所有的包含有 abvd 这 4 个字符的文件（其中*是通配符），比如 abvdrmyz 等符合条件的文件都能显示出来。

(4) 使用混合查找方式查找文件

find 命令可以使用混合查找的方法，例如，在/etc 目录中查找大于 500000 字节，并且在 24 小时内修改的某个文件，则可以使用-and (与)把两个查找参数链接起来组合成一个混合的查找方式。

find /etc -size +500000c -and -mtime +1

1.3 man 帮助命令

1. 功能

man 命令用来提供在线帮助，使用权限是所有用户。在 Linux 系统中存储着一部联机使用的手册，以供用户在终端上查找。使用 man 命令可以调阅其中的帮助信息，非常方便和实用。

2. 格式

man 命令名称

man [-acdfhkKtwW] [-m system] [-p string] [-C config_file] [-M path] [-P pager] [-S section_list] [section] name ...

3. 参数

-C config_file：指定设定文件 man.conf，缺省值是/etc/man.conf。

-M path：指定了联机手册的搜寻路径, 如果没有指定则使用环境变数 MANPATH 的设定；如果没有使用 MANPATH， 则会使用/usr/lib/man.conf 内的设定；如果 MANPATH 是空字串，则表示使用缺省值。

-P pager：指定使用何种 pager.man 会优先使用此选项设定，然后是依环境变数 MANPAGER 设定，然后是环境变数 PAGER；man 缺省使用/usr/bin/less -is。

-S section_list man：所搜寻的章节列表(以冒号分隔)，此选项会覆盖环境变数 MANSECT 的设定。

-a man：缺省情况是在显示第一个找到的手册之后，就会停止搜寻，使用此选项会强迫 man 继续显示所有符合 name 的联机手册。

-c：即使有最新的 cat page，也继续对联机手册重新作排版，本选项在屏幕的行列数改变时或已排版的联机手册损坏时特别有意义。

-d：不要真的显示联机手册，只显示除错讯息。

-D：同时显示联机手册与除错讯息。

-h：显示求助讯息然后结束程式 。

-K：对所有的联机手册搜寻所指定的字串。请注意，本功能回应速度可能很慢，如果指定 section（区域）会对速度有帮助。

-m system：依所指定的 system 名称而指定另一组的联机手册。

man：是 manual（手册）的缩写。在输入命令有困难时，可以立刻得到这个文档。例如, 如果使用 ps 命令时遇到困难，可以输入 man ps 得到帮助信息，此时会显示出 ps 的手册页（man page）。由于手册页 man page 是用 less 程序来看的(可以方便地使屏幕上翻和下翻), 所以在 man page 里可以使用 less 的所有选项。less 中比较重要的功能键有:

[q] 退出；

[Enter] 一行行地下翻；

[Space] 一页页地下翻；

[b] 上翻一页；

[/] 后跟一个字符串和[Enter]来查找字符串；

[n] 发现上一次查找的下一个匹配。

4. 应用实例

Linux 命令中有一些基础的、重要的命令，例如 ps、find、cat 和 ls 等。下面来举一个综合应用的例子，由此可以看出 man 的地位在 Linux 中可谓至关重 要。但是，man 所显示的信息却不是普通的文本，如果直接将这些文字重定向到一个文本文件，就会发现在 man 中高亮显示的文字就变成了两个，而且有不计其数的制表符，使打印、编辑都变得非常不便。不过，使用下面这样一条语句就能得到 ps 命令打印。

man ps | col -b | lpr

这条命令同时运用了输出重定向和管道两种技巧，作用是将 ps 命令的帮助信息可以直接打印出来。更多的 Man 文件可以查看 Linux Man

1.4 Linux 编程工具的使用

1.4.1 编辑器 vi 的使用

1) vi 的简单应用

vi 是 linux 环境下赫赫有名的文本编辑工具之一。

进入 vi 编辑器的方法：vi filename.c 注：文件名必须带有扩展名.c，如 filename.c 否则无法通过编译；

进入 vi 后要按：按“i”键从命令方式切换到输入方式；

从输入方式切换到命令方式的方法：按“Esc”键

保存文件 :w

保存文件并退出 :wq

不保存文件退出 :q!

注：以上操作必须在命令方式下进行。

2) 其他常用 vi 操作命令

j,k,h,*l*:上下左右

0： 行首

$: 行尾

ctrl+f :后翻页

ctrl+b:前翻页

G : 文件尾

数字 G: 数字所指定行

i,I : 插入命令，i 在当前光标处插入， I 行首插入

a,A: 追加命令，a 在当前光标后追加，A 在行末追加

o,O: 打开命令，o 在当前行下打开一行，O 在当前行上插入一行

x: 删除光标处字符

dd: 删除当前行

d0: 删除光标前半行

d$: 删除光标后半行

r,R : 替换命令，r 替换当前光标处字符，R 从光标处开始替换

/string： 查找字符串

n ： 继续查找

N ： 反向继续查找

% ：查找对应括号

u ：取消上次操作

注：以上操作必须在命令方式下进行。

1.4.2 编译器 gcc 的使用

gcc 是 linux 下的一种 c 程序编译工具，使用方法如下（有提示符#的情况）：

编译： gcc –o filename1 filename.c

其中: filename.c 是源文件名，filename1 是目标文件名，o 代表 object

执行：./filenamel

示例：键入一简单的程序：

vi ab.c 按回车键输入以下代码段，

```
#include<stdio.h>
#include<unisted.h>
main()
{printf(“aaaa”);
}
```

按 ESC 键，再按“**:WQ**”(W 是保存，Q 是退出)

出现提示符：[root @GGG-LINUX root]#

键入：gcc –o ab ab.c 进行编译。

执行：./ab

1.4.3 调试工具 gdb 的使用

Linux 中调试工具很多，此处只介绍一个强大的命令行调试工具 gdb。使用 GDB 调试 c++代码是 unix 环境 c++编程的基本技能。对于长期在 windows 平台下面从事 c++编程的程序员来说，在 linux 环境下面进行 gdb 调试确实有点难度，但是 gdb 本身是一个功能强大的代码调试工具，只要多使用，熟悉了基本的命令使用方法和 windows 平台下 IDE 的调试相关功能有个很好的影射，那么在 linux 下面进行 c++开发能力又将前进一大步。

1. 启动 gdb

   在使用 gdb 调试 C/C++程序之前，必须先使用 gcc –g 命令生成带有调试信息的可执行程序。

   如：gcc –o ab –g ab.c

   gdb ab　　##启动 gdb

2. 获取帮助

   任何时候都可以使用 help 命令查看帮助信息。

   如：(gdb) help

3. 查看源代码

   list 命令用于查看程序源代码，一次列出 10 行代码。gdb 会自动在源代码前加行号。按回车表示执行上一天命令。List 命令指定行号，列出改行附件的代码

   如：list

   或 list 10

   gdb 提供了 search 命令搜索特定内容。search 命令只显示第一个符合条件的行，再次按回车找到下一个符合条件的行。search 只能向文件尾搜索。reverse-search 向文件头搜索。

   如：search int summary　　##查找”int summary”

   reverse-search int summary

4. 设置断点

   break 命令(缩写为 b)用于设置断点，这个命令接受行号或函数名为参数。info break 命令查看已设置的断点信息。clear 命令清除当前所在行的断点。

   如：break 10

   break main

5. 运行程序与单步执行

   run 命令（缩写为 r）运行程序至断点。next 命令（缩写为 n）单步执行，或指定数字（如 next2 表示连续执行两行）。continue 命令（缩写为 c）继续运行程序，直至遇到下一个断点。

   step 命令（缩写为 s）单步执行代码，遇到函数调用时进入函数内部。

6. 监视变量

   print 命令（缩写为 p）显示指定变量的值。

如：print c　　##显示 c 的值

watch 命令用于设置观察点，接收变量名或表达式作为参数，一旦参数值发生变化就停下来。

如：watch sum

7. 临时修改变量

gdb 允许用户在程序运行时通过 set var 命令改变变量的值。

如：set var i=5

print i

8. 查看堆栈情况

bt 命令可以查看当前运行时栈的情况。

9. 退出 gdb

调试完毕，使用 quit 命令（缩写为 q）退出 gdb 程序。

1.4.4 主要系统调用函数

1. 系统调用 fork()

功能：创建一个新的进程.

头文件：#include <unistd.h>

说明:本系统调用产生一个新的进程, 叫子进程, 是调用进程的一个复制品. 调用进程叫父进程, 子进程继承了父进程的几乎所有的属性。

(1) 该子进程继承了父进程的程序空间，复制了父进程的数据段和栈段。也就是说不管是父进程还是子进程，在占有处理机后，都从 fork（）调用的返回点开始运行;

(2) 调用成功则对父进程返回子进程标识号 pid；

(3) 调用成功对子进程返回 0, 这也是最方便的区分父子进程的方法。

(4) 若调用失败则返回-1 给父进程, 子进程不生成。

注意：如果 fork()值>0,>0 的数即是子进程号。但这时是父进程占有处理机。

2. 系统调用 wait（&status）：

功能：父进程一旦调用了 wait 就立即阻塞自己，由 wait 自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸的子进程，wait 就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait 就会一直阻塞在这里，直到有一个出现为止。

当父进程忘了用 wait()函数等待已终止的子进程时,子进程就会进入一种无父进程的状态,此时子进程就是僵尸进程。wait()要与 fork()配套出现,如果在使用 fork()之前调用 wait(),wait()的返回值则为-1,正常情况下 wait()的返回值为子进程的 PID.

如果先终止父进程,子进程将继续正常进行，只是它将由 init 进程(PID 1)继承,当子进程终止时,init 进程捕获这个状态。

参数 status 用来保存被收集进程退出时的一些状态，它是一个指向 int 类型的指针。但如果我们对这个子进程是如何死掉毫不在意，只想把这个僵尸进程消灭掉，（事实上绝大多数情况下，我们都会这样想），我们就可以设定这个参数为 NULL，就像下面这样：

pid = wait(NULL);

如果成功，wait 会返回被收集的子进程的进程 ID，如果调用进程没有子进程，调用就会失败，此时 wait 返回-1，同时 errno 被置为 ECHILD。

如果参数 status 的值不是 NULL，wait 就会把子进程退出时的状态取出并存入其中， 这是一个整数值（int），指出了子进程是正常退出还是被非正常结束的，以及正常结束时的返回值，或被哪一个信号结束的等信息。

(1)当有多个子进程时，任一个子进程结束即将控制返回调用者，并将子进程调用 exit （status）时的 status 值送到&status 指针所指单元中。

(2)在控制返回调用者时，同时将所等到的子进程 pid 作为 wait（）系统调用函数的返回值。

(3)waitpid（pid，…）：等待 pid 所指定的进程结束。

(4)返回值：等待到一个子进程返回时，返回值为该子进程号；否则返回值为-1。

说明: 允许调用进程（即父进程）取得子进程的状态信息，调用进程将会挂起直到其一个子进程终止。

3. 系统调用 exit()

功能: 终止进程.

语法: #include <stdlib.h>

void exit(status)

int status

功能：调用进程被该系统调用终止。该系统调用发出后，操作系统将从系统中删除调用 exit 的进程，并将 status 值传给等待它结束的父进程。

返回值: 无

4. 系统调用 kill()

功能: 向一个或一组进程发送一个信号。

语法:#include <signal.h>

int kill(pid, sig);

pid_t pid; int sig;

说明:向一个或一组进程发送一个信号,该信号由参数 sig 指定，为系统给出的信号表中的一个。

Sig 是 signal 的缩写。

返回值: 调用成功则返回 0，否则返回-1.

kill -STOP [pid]： 发送 SIGSTOP (17,19,23)停止一个进程，而并不消灭这个进程。

kill -CONT [pid]： 发送 SIGCONT (19,18,25)重新开始一个停止的进程。

kill -KILL [pid]： 发送 SIGKILL (9)强迫进程立即停止，并且不实施清理操作。

kill -9 -1： 终止拥有的全部进程。

SIGKILL 和 SIGSTOP 信号不能被捕捉、封锁或者忽略。

5. 系统调用 lockf()

功能：应用、检测或删除打开文件的一个 POSIX 锁

语法：#include <unistd.h>

int lockf(int fd, int cmd, off_t len);

说明：

应用、检测或删除打开文件某部分的一个 POSIX 锁，文件通过 fd 指明，文件的描述符的请求操作通过 cmd 指明。

#define F_ULOCK 0 解锁一个区域

#define F_LOCK 1 上锁一个区域

#define F_TLOCK 2 检测并上锁一个区域

#define F_TEST 3 检测一个区域是否上锁

文件上锁区域的请求起始于隐含的偏移并包好 len 字节，假如 len 为负，从 pos…pos+len-1，这里 pos 为当前文件位置，假如 len 为零，则位置从当前文件位置延伸延伸到无限长，包括当前和以后的文件最后的位置。在所有情况下，位置可延伸到以前当前的文件的最后位置。

在 Linux 中，这称为 fcntl(2)的接口（一般会指明 lockf 和 fcntl 的关系）。

6. 系统调用 pipe()

是用来建立管道的。

语法: #include <unistd.h>

int pipe(int fd[2]); 这里 fd[1]为写入端，fd[0]为读出端。

功能：从管道里写或从管道里读。

第二部分 实验内容

实验一 熟悉 Linux 基本命令及编程环境

一、 实验类型

本实验为验证性实验。

二、 实验目的与任务

1）熟悉使用 Linux 字符界面，窗口系统的常用命令；

2）熟悉运用 Linux 常用的编程工具；

3）熟悉运用 Linux 的在线求助系统。

三、 预习要求

1）熟悉一种操作系统的使用和安装，如 windows 操作系统的安装，DOS 系统的使用

2）了解进程的概念及进程的状态

3）熟悉 c 语言程序设计

4）熟悉 c 语言程序编译过程

四、 实验基本原理

进入 Linux 操作系统后，控制终端的命令行输入方式下输入各种命令，并显示各种命令操作后的输出结果，操作包括文件操作命令的使用，vi 命令的使用以及 gcc 编译器的使用，详细的各种命令及使用方式见第一部分的介绍。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

1）使用常用的操作命令 ls, cp, rm, mkdir, man 等。

①显示当前目录下的文件

-#ls

②创建 1 个空目录

-#mkdir test

③进入目录

-#cd test

④创建子目录目录

-#mkdir test1

⑤在目录 test1 中建立一个文件 k.c

-#cd test1

-#vi k.c

⑥显示 test1 目录中的文件

-#ls

⑦显示文件内容

-#cat k.c

⑧判断文件 k.c 类型

-#file k.c

⑨删除目录 test1

-#rm k.c

-#cd..

-#rmdir test1

⑩用 man 命令查找 grep 命令含义

-#man grep

2）使用 vi 和 gcc 编辑一个 C 语言程序并运行（编辑，调试，运行)。

(1）参考程序 1：将一个数组逆序输出。用第一个与最后一个交换。

```
#include "stdio.h"
#define N 5
main()
{
  int a[N]={9,6,5,4,1},i,temp;
  printf("\n original array:\n");
  for(i=0;i<N;i++)
    printf("%4d",a);
  for(i=0;i<N/2;i++)
  {
    temp=a[i];
    a[i]=a[N-i-1];
    a[N-i-1]=temp;
  }
  printf("\n sorted array:\n");
```

```
    for(i=0;i<N;i++)
    printf("%4d",a);}
```

（2）参考程序 2：求 100 之内的素数

```
#include "stdio.h"
#include "math.h"
#define N 101
main()
{
 int i,j,line,a[N];
  for(i=2;i<N;i++) a[i]=i;
    for(i=2;i<sqrt(N);i++)
      for(j=i+1;j<N;j++)
      {
        if(a[i]!=0&&a[j]!=0)
          if(a[j]%a[i]==0)
            a[j]=0;
      }
  printf("\n");
  for(i=2,line=0;i<N;i++)
  {
    if(a[i]!=0)
    {
      printf("%5d",a[i]);
      line++;
    }
    if(line==10)
    {
      printf("\n");
      line=0;
    }
  }
}
```

（3）参考程序 3：对 10 个数进行排序。1.程序分析：可以利用选择法，即从后 9 个比较过程中，选择一个最小的与第一个元素交换，下次类推，即用第二个元素与后 8 个进行比较，并进行交换。

```
#include "stdio.h"
#define N 10
main()
{
 int i,j,min,tem;
  /*input data*/
  int a[N]={9,6,5,4,1,8,15,3,10,13};
  for(i=0;i<N;i++)
  {
    printf("  a[i]=%5d\n",a[i]);
  }

  printf("\n");
  /*sort ten num*/
  for(i=0;i<N-1;i++)
  {
    for(j=i+1;j<N;j++)
      if(a[i]>a[j])
  {
    tem=a[i];
    a[i]=a[j];
    a[j]=tem;
  }
  }
  /*output data*/
  printf("After sorted \n");
  for(i=0;i<N;i++)
  printf("%5d",a[i]);
}
```

七、 实验步骤

1）进入 Linux 操作系统

开机后，选择进入 Linux 操作系统方式，输入用户名和密码，即可进入 Linux 操作系统。

2）进入控制终端的命令行输入方式

在 Linux 操作系统环境下，点击“小红帽”菜单，选择“系统”下拉菜单，选择“终端”选项，进入命令行输入方式。

3）输入命令后按回车键

在命令行输入方式下，输入各种文件操作命令，并按回车查看显示结果。

八、 注意事项

1）gcc 编译器不能编译不带扩展名的 c 语言程序。

2）注意编译和运行程序的基本过程。

3）注意熟练使用 man 命令来查看某条命令的含义及使用方式。

九、 实验报告要求

要求列出多条命令的使用和相应结果，需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。

实验二 进程管理

一、 实验类型

本实验为设计性实验。

二、 实验目的与任务

1）掌握进程的概念，明确进程的含义

2）掌握进程创建方法，认识并了解并发执行的实质

3）熟悉进程的睡眠、同步、撤消等进程控制方法

4）分析进程竞争资源的现象，学习解决进程互斥的方法

三、 预习要求

1）进程的概念

2）进程控制的概念及内容

3）进程的并发执行

4）熟悉进程同步互斥的概念

5）用到的 Linux 函数有：fork()，exec( )，wait()，exit()，lockf()等。

四、 实验基本原理

使用 fork()系统调用创建一个子进程，如果 fork( )调用成功，它向父进程返回子进程的 PID，并向子进程返回 0，即 fork( )被调用了一次，但返回了两次。此时 OS 在内存中建立一个新进程，所建的新进程是调用 fork( )父进程（parent process）的副本，称为子进程（child process）。子进程继承了父进程的许多特性，并具有与父进程完全相同的用户级上下文，父进程与子进程并发执行。fork( )只是将父进程的用户级上下文拷贝到新进程中，而 exec( )系列可以将一个可执行的二进制文件覆盖在新进程的用户级上下文的存储空间上，以更改新进程的用户级上下文。exec( )系列中的系统调用都完成相同的功能，它们把一个新程序装入内存，来改变调用进程的执行代码，从而形成新进程。如果 exec( )调用成功，调用进程将被覆盖，然后从新程序的入口开始执行，这样就产生了一个新进程，新进程的进程标识符 id 与调用进程相同。使用 lockf()系统调用对临界区进行加锁操作，实现对共享资源的互斥使用。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

1）进程的创建

（1）编写程序，使用系统调用 fork( )创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示'a'，子进程分别显示字符'b'和字符'c'。试观察记录屏幕上的显示结果，并分析原因。

参考程序如下：

```
#include <stdio.h>
main( )
{
 int p1,p2;
 while((p1=fork( ))= = -1);              /*创建子进程 p1*/
 if (p1= =0)    putchar('b');
 else
  {
   while((p2=fork( ))= = -1);     /*创建子进程 p2*/
   if(p2= =0)    putchar('c');
   else    putchar('a');
   }
 }
```

运行结果：bca(或 bac)

分析：从进程并发执行来看，输出 bac，acb 等情况都有可能。

原因：fork()创建进程所需的时间要多于输出一个字符的时间，因此在主程序创建进程的同时，进程 2 就输出了”b”，而进程 2 和主程序的输出次序是随机的，所以出现上述结果。

（2）修改程序，每一个进程循环显示一句话。子进程显示'daughter …'及'son ……'，父进程显示 'parent ……'，观察结果，分析原因。

参考程序如下：

```
#include <stdio.h>
main( )
{
 int p1,p2,i;
 while((p1=fork( ))= = -1);                /*创建子进程 p1*/
 if (p1= =0)
    for(i=0;i<10;i++)
       printf("daughter  %d\n",i);
```

```
 else
    {
      while((p2=fork( ))= = -1);     /*创建子进程 p2*/
      if(p2= =0)
         for(i=0;i<10;i++)
         printf("son    %d\n",i);
      else
         for(i=0;i<10;i++)
       printf("parent    %d\n",i);
      }
}
```

运行结果：略

分析：由于函数 printf()输出和字符串之间不会被中断，因此字符串内部的字符顺序输出不变。但是由于进程并发执行时的调度顺序和父进程的抢占处理机问题，输出字符串的顺序和先后随着执行的不同而发生变化。

2）进程的互斥

（1）修改程序，用 lockf( )来给每一个进程加锁，以实现进程之间的互斥

进程加锁后的参考程序如下：

```
#include <stdio.h>
#include <unistd.h>
main( )
{
      int p1,p2,i;
      int n=500;
      while((p1=fork( ))= = -1);            /*创建子进程 p1*/
      if (p1= =0)
      {
       lockf(1,1,0); /*加锁，这里第一个参数为 stdout（标准输出设备的描述符）*/
       for(i=0;i<n;i++)
       printf("daughter %d\n",i);
       lockf(1,0,0);                              /*解锁*/
      }
else
    {
      while((p2=fork( ))= =-1);   /*创建子进程 p2*/
```

```
        if (p2= =0)
        {
         lockf(1,1,0);              /*加锁*/
         for(i=0;i<n;i++)
         printf("son %d\n",i);
         lockf(1,0,0);                    /*解锁*/
         }
         else
             {
               lockf(1,1,0);              /*加锁*/
               for(i=0;i<n;i++)
               printf(" parent %d\n",i);
               lockf(1,0,0);              /*解锁*/
             }
       }
  }
```

运行结果：大致与未上锁的输出结果相同，也是随着执行时间不同，输出结果的顺序有所不同。

分析：上述程序执行时，不同进程之间不存在共享临界资源（其中打印机的互斥性已由操作系统保证）问题，所以加锁与不加锁效果相同。

（2）分析下列程序结果（用 cat　to_be_locked.txt 查看输出结果）

```
#include<stdio.h>
#include<unistd.h>
main()
{
    int p1,p2,i;
    int n=500;
    int *fp;
    fp = fopen("to_be_locked.txt","w+");
    if(fp==NULL)
     {
        printf("Fail to create file");
        exit(-1);
     }
    while((p1=fork( ))== -1);          /*创建子进程 p1*/
```

```
        if (p1==0)
        {
            lockf(*fp,1,0);                    /*加锁*/
            for(i=0;i<n;i++)
            fprintf(fp,"daughter %d\n",i);
            lockf(*fp,0,0);                    /*解锁*/
         }
       else
        {
            while((p2=fork( ))==-1);   /*创建子进程 p2*/
          if (p2==0)
            {
              lockf(*fp,1,0);              /*加锁*/
              for(i=0;i<n;i++)
              fprintf(fp,"son %d\n",i);
              lockf(*fp,0,0);              /*解锁*/
             }
         else
           {
              wait(NULL);
              lockf(*fp,1,0);              /*加锁*/
              for(i=0;i<n;i++)
              fprintf(fp,"parent %d\n",i);
              lockf(*fp,0,0);              /*解锁*/
           }
        }
      fclose(fp);
}
```

七、 实验步骤

1）进入 vi 编辑器

2）在编译器中输入所要运行的程序代码

3）退出编辑器，返回命令行输入方式，使用 gcc 编译器编译程序，获得能运行的目标程序。

4）运行目标程序，查看运行结果。

八、 注意事项

1）如果使用 gcc 编译程序有错的话，需要重新修改程序，直到无错为止。

2）注意系统是如何创建进程的？

3）查看结果是否是交替输出，如果修改输出的次数是否会出现交替现象？

4）相关函数的介绍见第一部分的介绍。

九、 实验报告要求

需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。对结果的分析主要讨论结果为什么会交替出现？并发进程是如何执行的？

实验三 进程调度

一、 实验类型

本实验为综合性实验

二、 实验目的与任务

在采用多道程序设计的系统中，往往有若干个进程同时处于就绪状态。当就绪进程个数大于处理机数时，就必须依照某种策略来决定那些进程优先占用处理机。本实验模拟在单处理机情况下的处理机调度，帮助学生加深了解处理机调度的工作。

三、 预习要求

1）熟悉进程控制块和进程组织方式

2）熟悉进程调度的概念

3）熟悉时间片轮转调度算法等

4）熟悉 c 语言编程，指针及结构体等知识

5）数据结构中的链表的建立及基本操作

四、 实验基本原理

进程控制块通过链表队列的方式组织起来，系统中存在运行队列和就绪队列（为简单起见，不设阻塞队列），进程的调度就是进程控制块在运行队列和就绪队列之间的切换。当需要调度时，从就绪队列中挑选一个进程占用处理机，即从就绪队列中删除一个进程，插入到运行队列中，当占用处理机的进程运行的时间片完成后，放弃处理机，即在运行队列中的进程控制块等待一段时间（时间片）后，从该队列上删除，如果该进程运行完毕，则删除该进程（节点）；否则，则插入到就绪队列中。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

设计一个时间片轮转调度算法实现处理机调度的程序，具体内容如下

1）实验中使用的数据结构

(1) PCB 进程控制块

内容包括参数①进程名 name；②要求运行时间 runtime；③优先数 prior；④状态

state；⑤已运行时间 runedtime。

(2) 为简单起见，只设运行队列，就绪链表两种数据结构，进程的调度在这两个队列中切换，如图 3.1 所示

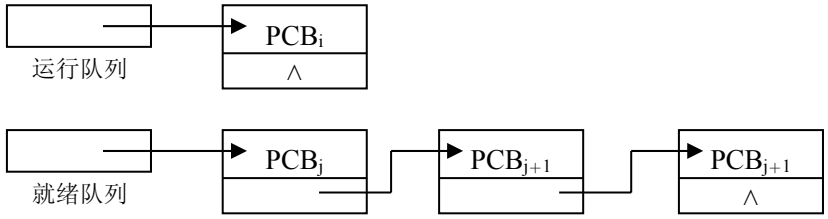


图 3.1PCB 链表

2）每个进程运行时间随机产生，为 1~20 之间的整数。

3）时间片的大小由实验者自己定义，可为 3 或 5

4）可参考的程序流程图如图 3.2

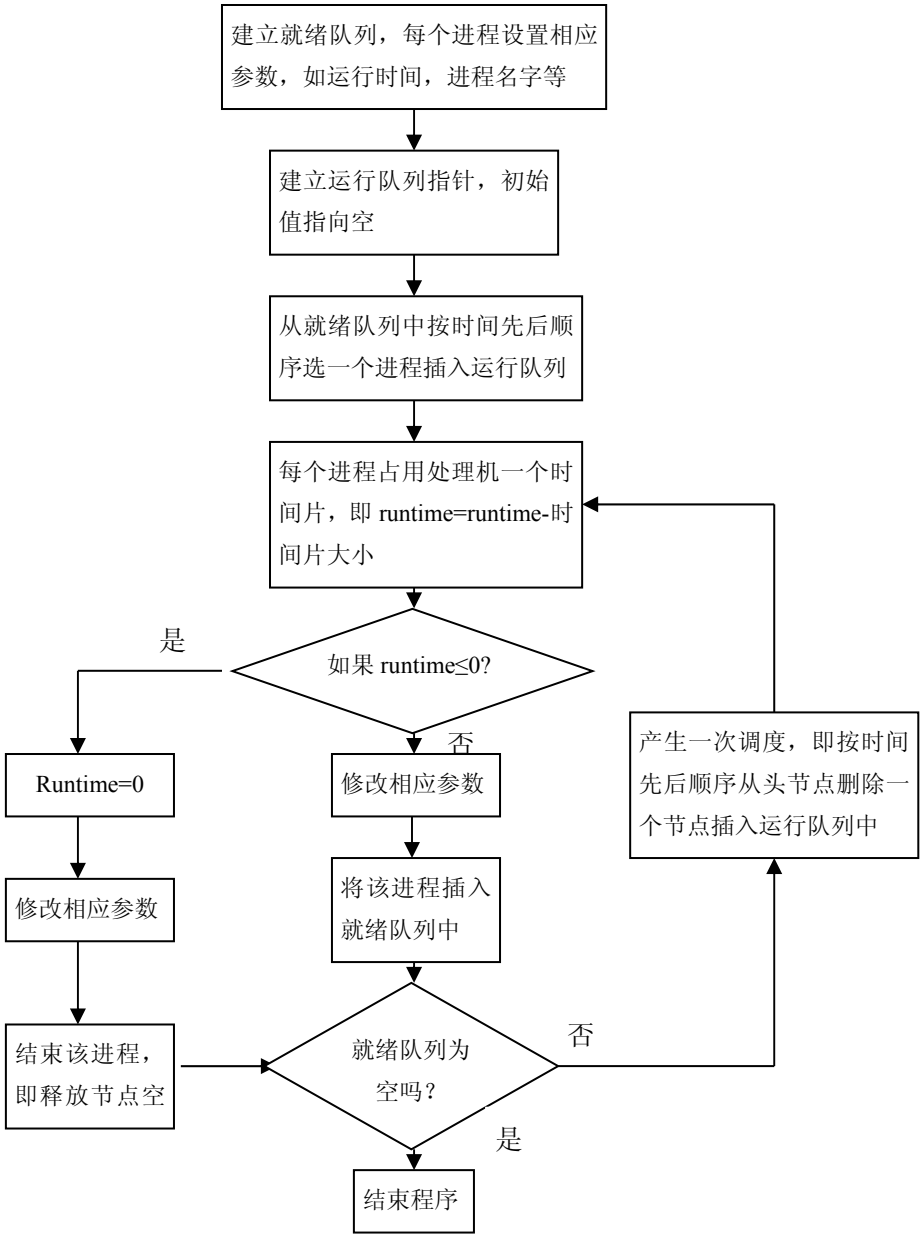


图 3.2 模拟进程调度的流程图

5）参考程序

```
#include "stdio.h"
#include "stdlib.h"

typedef struct PCB
{
        int name;
        int runtime;
        int runedtime;
        int state;
        int killtime;
   struct PCB *next;
}PCB;
#define NUM 10

void main()
{
      int timeslice=3;
      PCB *runqueue;
      PCB *top,*tail,*temp;
      int i;
      srand(10);
    for(i=0;i<NUM;i++)
      {
            temp=new PCB;
            temp->name=i;
            temp->runtime=rand()%20;
            temp->runedtime=0;
            temp->next=NULL;
            temp->killtime=0;
           if(i==0) {top=temp; tail=temp;}
           else{
                tail->next=temp;
                tail=temp;
           }
           printf("process name %d, runtime=%d, runedtime=%d,killtime=%d\n"
           ,tail->name,tail->runtime,tail->runedtime,tail->killtime);
      }

      while(top!=NULL)
      {
        runqueue=top;
        top=top->next;
        runqueue->next=NULL;
        runqueue->runtime= runqueue->runtime-timeslice;
        if(runqueue->runtime<=0)
```

```
        {
          runqueue->killtime=runqueue->runtime+timeslice;
          runqueue->runedtime=runqueue->runedtime+runqueue->killtime;
          runqueue->runtime=0;
          printf("process name %d, runtime=%d, runedtime=%d,killtime=%d\n"
          ,runqueue->name,runqueue->runtime,runqueue->runedtime,runqueue->killtime);
        }
        else{
          runqueue->killtime=timeslice;
          runqueue->runedtime=runqueue->runedtime+runqueue->killtime;
          printf("process name %d, runtime=%d, runedtime=%d,killtime=%d\n"
          ,runqueue->name,runqueue->runtime,runqueue->runedtime,runqueue->killtime);
          tail->next=runqueue;
          tail=tail->next;
        }
    }
}
```

6）运行结果，包括各个进程的运行顺序，每次占用处理机的运行时间，可以参考下列输出如图 4.3。

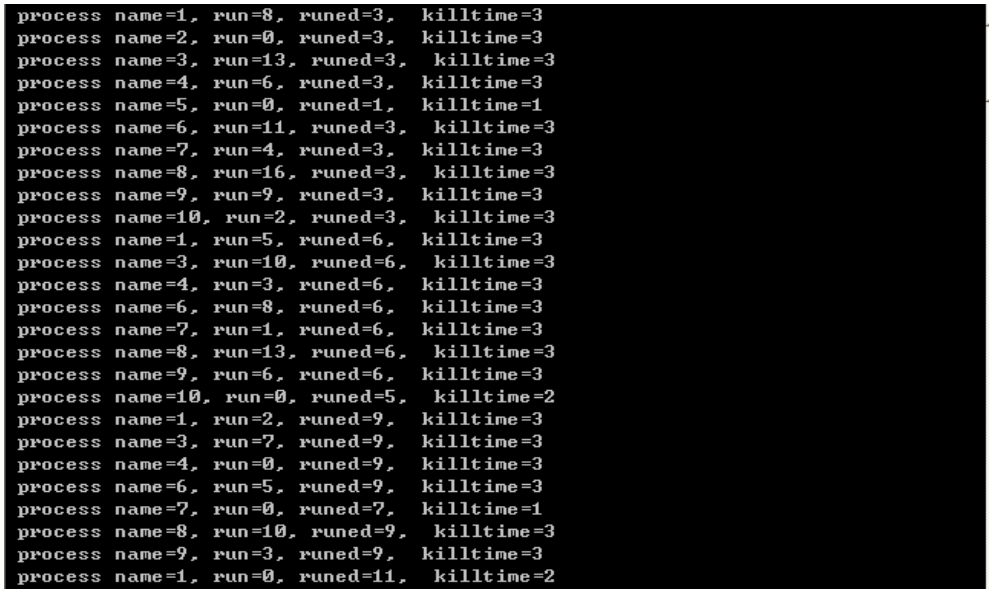


图 4.3 输出结果示意图

七、 实验步骤

1）进入 vi 编辑器

2）在编译器中输入所要运行的程序代码

3）退出编辑器，返回命令行输入方式，使用 gcc 编译器编译程序，获得能运行的目标程序。

4）运行目标程序，查看运行结果。

八、 注意事项

1）修改时间片大小，查看对实验结果的影响。

2）随机数的产生由 rand()函数实现，rand()的输出随机数范围在 0~2^{15} 之间，需要转换到 0~20 范围。

3）注意链表节点的插入，删除方法。

九、 实验报告要求

需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。对结果的分析主要讨论时间片大小对程序执行的影响？

实验四 进程间通信

一、 实验类型

本实验为设计性实验

二、 实验目的与任务

了解 Linux 系统中的进程管道通信的基本原理，并能编写简单的管道通信的程序。

三、 预习要求

1）熟悉进程通信的概念

2）熟悉管道通信的基本原理

3）用到的系统调用有 pipe()

四、 实验基本原理

实验利用 Linux 操作系统提供的系统调用 pipe()来实现两个进程之间的通信。输入进程从管道的一端输入数据，另一个进程从管道的另一端读数据，通过这种方式来实现进程间的通信。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

编制一段程序，实现进程管道通信。使用系统调用 pipe()建立一条管道。两个子进程 P1 和 P2 分别向管道各写一句话：

Child process 1 is sending message!

Child process 2 is sending message!

而父进程则从管道中读出来自两个子进程的信息，显示在屏幕上。

参考程序程序如下

```
#include<unistd.h>
#include<signal.h>
#include<stdio.h>
int pid1,pid2;
main(){
int fd[2];
```

```
    char OutPipe[100],InPipe[100];
    pipe(fd);
    while((pid1=fork())==-1);
    if(pid1==0){
    lockf(fd[1],1,0);
    sprintf(OutPipe,"\n Child process 1 is sending message!\n");
    write(fd[1],OutPipe,50);
    sleep(5);
    lockf(fd[1],0,0);
    exit(0);
    }
    else{
    while((pid2=fork())==-1);
    if(pid2==0){
    lockf(fd[1],1,0);
    sprintf(OutPipe,"\n Child process 2 is sending message!\n");
    write(fd[1],OutPipe,50);
    sleep(5);
    lockf(fd[1],0,0);
    exit(0);
    }
    else{
    wait(0);
    read(fd[0],InPipe,50);
    printf("%s\n",InPipe);
    wait(0);
    read(fd[0],InPipe,50);
    printf("%s\n",InPipe);
    exit(0);
    }
    }
}
```

七、 实验步骤

1）进入 vi 编辑器

2）在编译器中输入所要运行的程序代码

3）退出编辑器，返回命令行输入方式，使用 gcc 编译器编译程序，获得能运行的目标程序。

4）运行目标程序，查看运行结果。

八、 注意事项

1）管道按怎样的方式传送消息？是否能够双向传送消息？

2）在管道通信方式中，如何来实现进程同步与互斥的？

3）pipe()函数的使用可参看第一部分的介绍。

九、 实验报告要求

需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。对结果的分析主要讨论管道通信方式的特点，实验结果是否体现该特点。

实验五 存储管理实验

一、 实验类型

本实验为综合性实验。

二、 实验目的与任务

1）理解动态异长存储分区资源管理

2）掌握所需数据结构和管理程序

3）了解各种存储分配算法的优点和缺点。

4）编程实现动态不等长存储管理的模拟程序。

三、 预习要求

1）进程控制的概念及内容

2）熟悉存储管理的概念

3）了解动态分区管理的思想，熟悉分配算法和回收算法

4）熟悉 c 语言编程，指针及结构体等知识

5）数据结构中的链表的建立及基本操作

四、 实验基本原理

使用一个一维数组来模拟内存储空间，建立内存块来记录内存分配使用情况，通过随机产生进程及其所需要的内存来模拟真实的进程。通过给进程分配内存及回收来实现对动态不等长存储管理方法。

五、 实验仪器与设备（或工具软件）

实验设备：计算机一台，软件环境要求：安装 Red Hat Linux 操作系统和 gcc 编译器。

六、 实验内容

1）实验中使用的数据结构

(1) 内存块表，包括参数①进程名 name；②起始地址 address；③长度 length；④标志 flag，表示该块是否被分配。

(2) 为简单起见，只设内存分配记录链表数据结构，用来记录内存分配与空闲情况。

2）实验中假设有若干个进程，如 5 个，每个需要空间随机产生，为 0~20 之间的整

数，进程名字实验者自己定义，可以是一个整数。

3）其他一些参数，如内存空间的大小实验者自己定义，建议为 100；

4）为了得到清晰的实验结果，建议先给每个进程分配存储空间，后进行回收。

5）程序的流程图如图 5.1, 图 5.2, 图 5.3

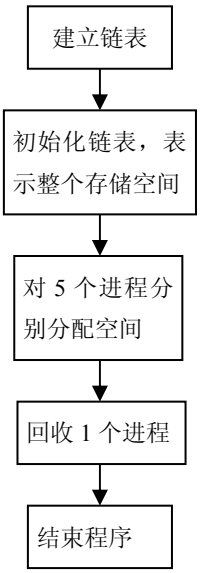


图 5.1 总的流程图

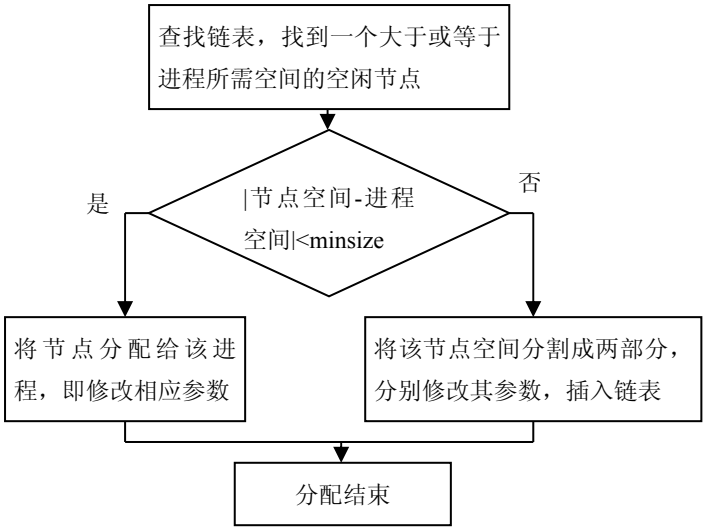


图 5.2 分配流程图

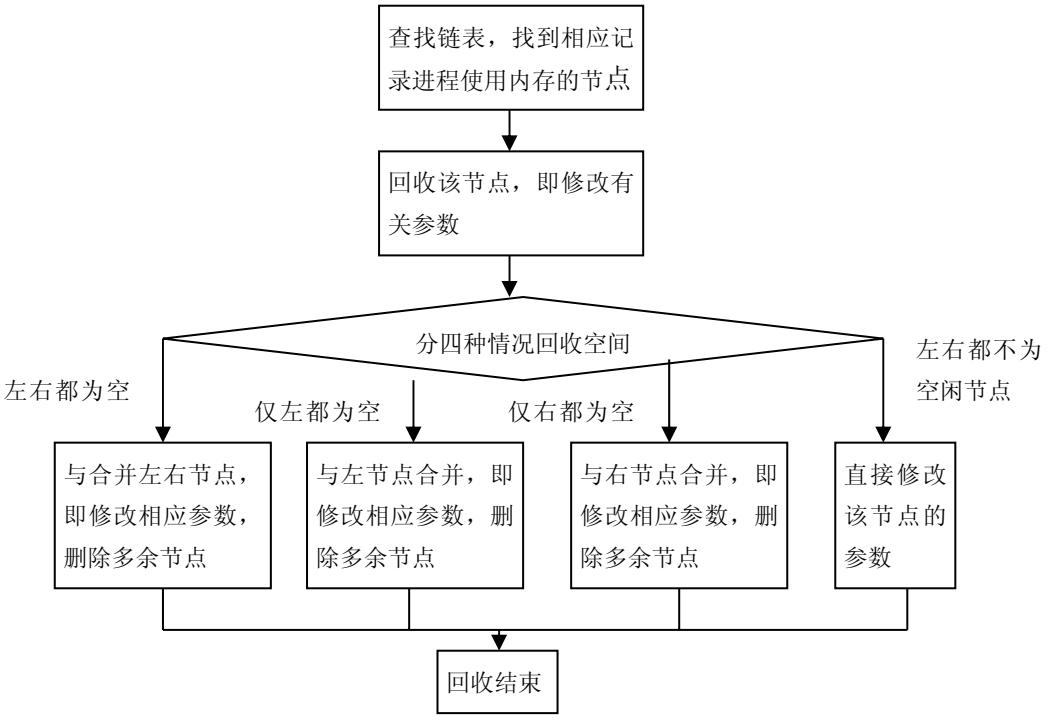


图 5.3 回收流程图

6）参考程序

```
#include "math.h"
#include "stdafx.h"
#include "stdio.h"
```

```
#include "stdlib.h"
 typedef struct MEMORY_BLOCK
{
   int name;
   int address;
   int length;
   int flag;
   MEMORY_BLOCK *next;
}MEMORY_BLOCK;
int allocation(MEMORY_BLOCK *Header,int name,int time);
int reclaim(int processname, MEMORY_BLOCK *Header);
int main()
{
 #define NUM 10
 int time,i;
 MEMORY_BLOCK *Header,*t;
 Header=new MEMORY_BLOCK;   //初始化存储空间
 Header->name=-1;
 Header->address=0;
 Header->length=100;
 Header->flag=0;
 Header->next=NULL;
srand(100);
 for(i=0;i<NUM;i++)
 {
  time=rand()%20;      // time=time%20;
  allocation(Header,i,time);
 }
 t=Header;
 while(t!=NULL)
 {
  printf("process name %d, address=%d, length=%d,flag=%d\n"
       ,t->name,t->address,t->length,t->flag);
  t=t->next;
 }
      int processname=3;//回收
      printf("回收  process name %d\n",processname);
      reclaim(processname,Header);
      t=Header;
 while(t!=0)
 {
  printf("process name %d, address=%d, length=%d,flag=%d\n"
       ,t->name,t->address,t->length,t->flag);
  t=t->next;
 }
    processname=4;
```

```
    printf("回收  process name %d\n",processname);
    reclaim(processname,Header);
   t=Header;
  while(t!=0)
  {
   printf("process name %d, address=%d, length=%d,flag=%d\n"
       ,t->name,t->address,t->length,t->flag);
   t=t->next;
  }
   return 1;
}
int reclaim(int processname, MEMORY_BLOCK *Header)
{
   MEMORY_BLOCK *temp,*t,*tt;
  t=Header;
  temp=t;
  while(t->name!=processname)
  {
   temp=t;
   t=t->next;
  }
  if(t==0) {printf("no process");return 0;}
   else
   { if(t->next!=NULL)
    if(temp->flag==0&&t->next->flag==0)//左右为空
    {   temp->name=-1;
        temp->length=temp->length+t->length+t->next->length;
    tt=t->next;
    temp->next=tt->next;
    delete tt;
    delete t;
    }
       else if(temp->flag==0) //左为空
   {
        temp->name=-1;
        temp->length=temp->length+t->length;
    temp->next=t->next;
    delete t;
   }
   else if(t->next->flag==0) //右为空
   {
    t->name=-1;
        t->length=t->length+t->next->length;
    t->flag=0;
    tt=t->next;
    t->next=tt->next;
```

```
    delete tt;
   }
   else {
    t->name=-1;
    t->flag=0;
   }
  else
       {
       if(temp->flag==0) //左为空
   {
        temp->name=-1;
        temp->length=temp->length+t->length;
    temp=t->next;
    delete t;
   }
   else {
    t->name=-1;
    t->flag=0;
   }
       }
   }
 return 1;
}

int allocation(MEMORY_BLOCK *Header,int name,int time)
{
 MEMORY_BLOCK *temp,*t,*tt;
 int thresh=2;
 t=Header;
  while(t!=0)
  {
 if(t->length>time&&t->flag==0) break;
     t=t->next;
  }
  if(t==0) { printf("no memory :%d\n",name); return 0;}
  else{

        if(t->length-time>thresh) //分割
        {
                  temp=new MEMORY_BLOCK;
                temp->name=-1;
                temp->flag=0;
            temp->length=t->length-time;
            temp->address=t->address+time;
                t->name=name;
                t->flag=1;
```

```
                t->length=time;
                    temp->next=t->next;
                t->next=temp;
            }
            else   //直接分配
            {
                    t->name=name;
                    t->flag=1;
            }
     }
    return 1;
  }
```

7）编写测试程序，对存储分配表进行初始化，存储分配情况和回收一个进程的存储空间后的结果在屏幕上显示出来，显示的结果如图 5.4 所示。

```
初始内存分配情况:  共5个进程
process name: 1, address: 0, length:1, flag: 1
process name: 2, address: 1, length:5, flag: 1
process name: 3, address: 6, length:8, flag: 1
process name: 4, address: 14, length:11, flag: 1
process name: 5, address: 25, length:14, flag: 1
process name: 0, address: 39, length:61, flag: 0

回收后内存分配情况:  进程   1
process name: 0, address: 0, length:2, flag: 0
process name: 2, address: 1, length:5, flag: 1
process name: 3, address: 6, length:8, flag: 1
process name: 4, address: 14, length:11, flag: 1
process name: 5, address: 25, length:14, flag: 1
process name: 0, address: 39, length:61, flag: 0
Press any key to continue
```

图 5.4 模拟输出的示意图

七、 实验步骤

1）进入 vi 编辑器

2）在编译器中输入所要运行的程序代码

3）退出编辑器，返回命令行输入方式，使用 gcc 编译器编译程序，获得能运行的目标程序。

4）运行目标程序，查看运行结果。

八、 注意事项

1）随机数的产生由 rand()函数实现，rand()的输出随机数范围在 0~215 之间，需要转换到 0~20 范围。

2）节点的删除和插入方式。

3）回收内存空间时，分四种情况讨论是否需要合并。

九、 实验报告要求

需要列出运行了的程序清单及相应结果，并对结果进行分析和讨论。对结果的分析主要讨论首次适应存储分配算法的优缺点, 实验结果是如何体现的？