广东工业大学

计算机科学与技术学院

《Linux技术》

实验报告



|  |  |
| --- | --- |
| 课程名称： | Linux技术 |
| 实验名称： | shell编程、常用开发工具 |
| 指导教师： | 赵靖亮 |
| 学生姓名： | 王宁 |
| 班级学号： | 4-3120005084 |
| 实验日期： | 2023-05-22 |
| 实验时间： | 4学时 |

实验报告撰写要求

实验操作是教学过程中理论联系实际的重要环节，而实验报告的撰写又是知识系统化的吸收和升华过程，因此，实验报告应该体现完整性、规范性、正确性、有效性。现将实验报告撰写的有关内容说明如下：

1、 实验报告模板为电子版。

2、 下载统一的实验报告模板，学生自行完成撰写和打印。报告的首页包含本次实验的一般信息：

*  班级学号：例如：2-5 表示第二班学号为5。
*  实验日期：例如：2023-05-09 表示本次实验日期。
*  实验时间：例如：4学时 表示本次实验所用的时间。

实验报告正文部分，从五个方面（目的、内容、步骤等）反映本次实验的要点、要求以及完成过程等情况。模板已为实验报告正文设定统一格式，一、二已填好，学生只需填三、四、五。三和四计分，五不计分。

3、 实验过程中可灵活参考教材、PPT和网络教程以解决问题。报告应独立完成，若出现雷同报告，均按零分处理。

4、 实验报告正文部分具体要求如下：

一、实验目的

本次实验所涉及并要求掌握的知识点。

二、实验内容

实验内容、原理分析及具体实验要求。

三、主要实验步骤

根据具体实验，记录、整理相应操作、命令、运行结果等，包括截图和文字说明。

若显示内容过长，可仅截取关键部分图。

具体项目已给出，学生在项目下面添加实验记录。

四、相关知识测试

完成思考题目。

五、实验心得体会

若在实验过程中出现故障和问题，应详细记录并进行故障分析，说明故障排除的过程及方法。

一、实验目的

*本次实验所涉及并要求掌握的知识点。*

**1. shell编程**

1.1 了解shell的特点和主要种类。

1.2掌握shell脚本的建立和执行方式。

1.3掌握 bash 的基本语法。

1.4学会编写 shell脚本。

**2.常用开发工具**

2.1掌握 C 语言编译系统 gcc 的基本用法。

2.2掌握 gdb 调试工具的基本用法。

2.3理解 make 程序维护工具的功能，学会编制 makefile 文件。

二、实验内容

**1. shell编程**

1.1 shell脚本的建立和执行。

1.2历史命令和别名定义。

1.3 shell变量和位置参数、环境变量。

1.4 bash 的特殊字符。

1.5 一般控制结构。

1.6 算术运算及 bash函数。

**2. 常用开发工具**

2.1利用 gcc 命令编译 C语言程序，使用不同选项，观察并分析显示结果。

2.2 用gdb 命令调试一个编译后的 C语言程序。

2.3编写一个由多个文件构成的 C语言程序，编制 makefile，然后运行 make 工具进行维护。

三、主要实验步骤

**1. shell编程**

1.1 利用vi建立一个脚本文件，其中包括 date、cal、pwd、ls 等常用命令；然后以不同方式执行该脚本。

图形用户界面, 文本

描述已自动生成

# 方式1

bash ex1

# 方式2

bash <ex1

# 方式3

chmod a+x ex1 && ./ex1

# 方式4：将当前目录配置到环境再执行

chmod a+x ex1

PATH+$PATH:.

ex1

文本

描述已自动生成文本

描述已自动生成

文本

描述已自动生成

1.2 运行 history 命令，配置历史命令环境。

[简书-设置linux系统history](https://www.jianshu.com/p/7a14b98e750d)

# 将内容追加到.bash\_profile文件并重新登录或 source .bash\_profile

vi ~/.bash\_profile

HISTFILE="/home/wn-3120005084/桌面/test/lab2/.my\_history"

# 控制历史命令记录的总行数

HISTSIZE=3000

HISTFILESIZE=3000

# 查看历史记录文件地址

echo $HISTFILE

# 将HISTFILE更新为默认的.bash\_history

vi ~/.bashrc

HISTFILE="$HOME/.bash\_history" # 在~/.bashrc文件末尾追加

source ~/bashrc

手机屏幕截图

描述已自动生成

1.3 体会 bash 的命令补齐功能。

键入hist按下1次Tab键，系统补全命令history。

键入mk之后连续按2次Tab键，系统会列出符合条件的所有命令以供参考。

文本

描述已自动生成

1.4 用alias 定义别名，然后执行。

文本

描述已自动生成

1.5 分析 shell 脚本 ex1.5的功能，然后执行。

根据传递给脚本的参数数量，构建一个字符串，将传递给脚本的参数按倒序拼接成一个命令，并将该命令执行。

如：bash ex1.5 arg1 arg2 arg3会构建出eval “echo arg3 arg2 arg1”执行倒序输出参数。

屏幕上有字

描述已自动生成

1.6 编写一个shell脚本，求斐波那契数列的前10项及总和，然后执行。

图形用户界面, 文本

描述已自动生成



**2. 常用开发工具**

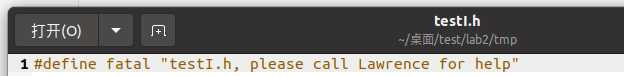
2.1 结合课件 8-常用开发工具 的6.1.3节，对hello.c进行编辑和编译。

分别测试如下编译选项：-I，-D，-E，-c，-o，并说明这些选项的含义。

|  |  |
| --- | --- |
| **选项格式** | **功能** |
| -I dir | 指定搜索头文件的路径dir。先在指定的路径中搜索要包含的头文件，若找不到，则在标准路径（/usr/include,/usr/lib及当前工作目录）上搜索 |
| -D name  -D name=definition | 定义一个宏name，其值为1  定义一个宏name，并指定其值为definition。其作用等价于在源文件中使用宏定义指令：#define name definition。但-D选项比宏定义指令的优先级高，它可以覆盖源文件中的定义 |
| -E | 只对指定的源文件进行预处理，不进行编译，生成的结果送到标准输出 |
| -c | 只生成目标文件，不进行链接。用于对源文件的分别编译 |
| -o file | 将输出写到file指定的文件中，等价于把file作为cpp的第二个非选项参数（即预处理后的输出文件） |

图形用户界面, 文本, 应用程序

描述已自动生成



1. 利用-I选项指定搜索头文件的路径：

gcc -I tmp hello.c

文本

描述已自动生成

1. 利用-D选项声明DOPTION为”testing -D”，利用-E选项只看预处理结果：

gcc -I tmp -D DOPTION='"testing -D"' -E hello.c

文本

描述已自动生成

1. 注释掉包含行，利用-c选项编译hello.c文件生成hello.o文件，：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

gcc -D DOPTION='"testing -D"' -c hello.c

图形用户界面, 文本, 应用程序

描述已自动生成

1. 利用-o选项指定输出写到指定文件：

gcc -D DOPTION='"testing -D"' hello.c -o hello.out

chmod a+x hello.out && ./hello.out

文本

描述已自动生成

2.2 对 badprog.c 进行编译，并简单分析该程序的功能。

结合课件的 6.2.6节，使用gdb对程序进行调试：

1. 显示代码的1~5行

gcc -g badprog.c -o badprog.out

gdb ./badprog.out

list 1,5

文本

描述已自动生成

1. 分别用参数 –b 和 –f 运行程序

// 退出gdb

q

./badprog.out -b

文本

描述已自动生成

./badprog.out -f



1. 在第12行设置断点，打印 argc 的数值，然后继续运行

gdb ./badprog.out

// 在第12行设置断点

break 12

// 运行程序

run

// 打印argc的数值

print argc

// 继续运行

continue

文本

描述已自动生成

1. 打印p[0]的内存地址

// 在第12行设置断点

break 12

// 运行程序

run

// 打印p[0]的内存地址

print &p[0]

// 继续运行

continue

文本

描述已自动生成

1. 分析代码中的错误之处
2. 当argv[1]等于"-f "时，有一个额外的空格导致条件判断失败。

break 18

// 运行程序并传递”-f”参数

run -f

// 打印argv[1]的值

print argv[1]

continue

文本

描述已自动生成

发现argv[1]的值是"-f"，没有额外的空格。因此，条件判断失败。

修复代码，将条件判断修改为：strcmp(argv[1], "-f") == 0。

1. 在使用p[50]='a'时，访问了超过p分配的内存范围，导致内存越界。

break 16

// 运行程序并传递”-b”参数

run -b

print p

print sizeof(p)

continue

文本

描述已自动生成

在使用p[50]='a'时，访问了超过p分配的内存范围，导致内存越界。

修复代码，确保不会访问超过分配内存范围的位置。将p[50]=’a’;修改为p[0]='a';

1. free(p);

图形用户界面, 文本, 应用程序, Word

描述已自动生成

**最终修改后的代码：**

文本

描述已自动生成

**运行结果：**

文本

描述已自动生成

**功能：**

通过命令行参数控制对分配的内存进行操作。

2.3 设某个正在开发的程序由以下内容组成：

1. 4个C语言源文件：a.c，b.c，c.c和d.c。设b.c和d.c都使用了defs.h中的声明。
2. 汇编语言源文件assmb.s被某个C语言源文件调用。
3. 使用了在/home/user/lib/libm.so中的一组例程。
4. 设最后生成的可执行文件名为prog。

试画出依赖关系图，并编写相应的makefile文件（不可使用隐式规则）。

**依赖关系图：**



**Makefile文件：**

图形用户界面, 应用程序

描述已自动生成

四、相关知识测试

1.1 说明Linux系统的体系结构分为哪几层？

1. 应用层
2. **用户空间**
3. C库和系统库
4. **内核空间**
5. **硬件层**

1.2 说明Linux系统核心的组成情况？

1. 内核
2. 设备驱动程序
3. 系统调用接口
4. 文件系统
5. 网络协议栈

内核作为操作系统的核心提供了基本的系统功能和服务，而设备驱动程序、系统调用接口、文件系统和网络协议栈等则扩展了内核的功能。

表格

描述已自动生成

1.3 Linux系统的进程有哪两种模式？各有什么特点？

Linux系统的进程有两种模式：用户态、内核态。

通过用户态和内核态的划分，Linux系统能够实现对进程的权限和资源访问进行有效的控制和保护，提高了系统的稳定性和安全性。

用户态和内核态之间的切换需要通过特定的机制，如系统调用、中断和异常等。

特点：

1. 用户态

用户态进程运行在较低的特权级别，受限于操作系统的保护机制，无法直接访问和操作核心功能和敏感资源；

用户态进程的执行速度较快，但受到访问限制。

1. 内核态

内核态进程运行在较高的特权级别，具有更高的权限和访问权，可以执行特权指令和访问核心功能和敏感资源；

内核态进程的执行速度较慢，但拥有更广泛的系统权限；

内核态进程对系统的稳定性和安全性具有重要作用。

1.4 Linux系统如何执行进程调度？

进程调度机制主要设计调度方式、调度时机、调度策略、调度算法。

1. 调度方式：抢占式优先级

进程在用户态下，当时间片用完或等待I/O，核心可暂停其运行，而让其它进程运行；

进程在内核态下，不受以上限制；

核心为每个进程计算一个优先权，分配时间片；

进程运行过程中，时间片和优先权随时间递减；

当所有进程的优先权都变为0时，重新计算优先权。

1. 调度策略

Linux系统支持多种调度策略，如先来先服务（FIFO）、最短作业优先（SJF）、时间片轮转等。

可以根据需求选择不同的调度策略，或者根据实际情况进行调度策略的定制和扩展。

1. 调度时机

核心进行进程调度的时机有五种：

当前进程调用系统调用nanosleep()或pause()，使自己进入睡眠状态，主动让出CPU使用权；

进程终止，永久放弃对CPU的使用；

时钟中断处理程序发现，当前进程连续运行时间过长；

当唤醒一个睡眠进程时，发现该进程比当前进程更有资格运行；

一个进程通过系统调用(如nice命令)，降低了自身的优先权从而引起立即调度。

1. 调度算法

在设计上应比较简单，以便减少频繁调度时的系统开销；

先查找就绪队列的进程，从中选择优先级最高且在内存的一个进程，执行；

如果队列中有实时进程，则优先运行；

如果最需要运行的进程不是当前进程，则挂起当前进程，保存它的现场(包括程序计数器、CPU寄存器)；

然后，为优先级最高的进程恢复现场。

1.5 什么是索引节点？它与文件有何关系？

1. 索引节点

索引节点用于存储文件的元数据信息，包含了文件的各种属性和控制信息。每个文件、目录都对应一个唯一的索引节点，而文件名则是通过索引节点与文件进行关联。

1. 索引节点与文件的关系：通过索引节点，操作系统可以管理文件的访问和操作，实现文件的读写、修改和删除等操作。

索引节点与文件的关系可以通过文件系统中的目录进行映射。目录是一种特殊的文件，它包含了文件名和对应的索引节点号的映射关系。当用户创建或打开一个文件时，操作系统会根据文件名查找对应的索引节点，并读取相关的元数据信息。通过索引节点，操作系统可以获取文件的各种属性和控制信息，进而进行文件的读取、写入、修改和删除等操作；

索引节点的存在可以使多个文件名指向同一个索引节点，从而实现文件的硬链接。硬链接是指通过创建新的文件名与同一个索引节点关联，使得多个文件名都可以访问同一个文件内容。不同的文件名只是不同的路径，而文件内容和索引节点是共享的。当其中一个文件名进行修改或删除时，其他的文件名仍然可以访问和操作文件内容。

1.6 为什么要设立虚拟文件系统（VFS）？

设立虚拟文件系统（VFS）可以提供统一的文件系统接口、支持多种文件系统类型、实现文件系统的层次结构、支持文件共享和并发访问控制，以及增强应用程序的可移植性。

VFS的存在使得操作系统能够更加灵活、可扩展和易用，屏蔽了底层文件系统的差异性，提供了更好的文件系统抽象和管理能力。

1.7 Linux为何采用三级页表？该机制如何工作？

通过采用三级页表的机制，Linux可以管理大容量的内存空间，因为三级页表的层次结构可以扩展地址空间。同时，三级页表也提供了高效的内存管理能力，因为多级页表的层次结构可以减少整个页表的大小，节省内存。此外，三级页表的惰性加载策略可以有效地利用内存资源，并提高内存访问效率。

三级页表的工作原理：

1. 物理内存划分

物理内存被划分成固定大小的页框，每个页框大小通常为4KB或2MB。

1. 页表层次结构

Linux的三级页表由三个层次组成，分别是页全局目录表（PGD）、页中间目录表（PMD）和页表（PT）。每个层次都是一个表，用于存储页表项。

1. 虚拟地址翻译

当应用程序访问虚拟地址时，CPU会通过页表进行翻译以获取对应的物理地址。

1. 页表项结构

每个页表项记录了虚拟页到物理页框的映射关系。页表项包含了物理页框的地址、访问权限标志等信息。

1. 惰性加载

Linux采用惰性加载的策略来节省内存。当应用程序访问虚拟页时，如果对应的物理页框不在内存中，操作系统会触发缺页异常。此时，操作系统会根据页表中的映射关系将物理页框加载到内存中，并更新页表项。

1.8 管道文件如何实现两个进程间的通信？

使用管道进行进程间通信的基本步骤：

1. 创建管道：调用pipe()函数创建一个管道，该函数会返回两个文件描述符，通常将其中一个描述符用于写入数据，另一个描述符用于读取数据。
2. 创建子进程：使用系统调用（如fork()）创建一个子进程，子进程会继承父进程的文件描述符。
3. 进程通信：在父进程和子进程中使用管道的文件描述符进行通信。父进程可以通过写入管道的文件描述符将数据发送给子进程，而子进程可以通过读取管道的文件描述符来接收数据。
4. 关闭文件描述符：在通信完成后，需要关闭未使用的文件描述符以释放资源。通常，父进程会关闭读取管道的文件描述符，而子进程会关闭写入管道的文件描述符。

表格

描述已自动生成

图示

描述已自动生成

1.9 Linux系统怎样处理系统调用？

当CPU执行到用户程序中的系统调用时，处理机的状态变为核心态，从而进入操作系统内部，执行有关代码，实现操作系统对外服务。

1. 用户空间发起系统调用；
2. 用户空间到内核空间的切换；
3. 内核执行系统调用：一旦切换到内核空间，操作系统的内核开始执行相应的系统调用处理程序。内核根据系统调用号确定要执行的具体操作；
4. 参数验证和复制；
5. 执行系统调用操作；
6. 返回结果到用户空间：一旦系统调用执行完成，内核将结果返回给用户空间。通常，结果会被复制到用户空间的缓冲区中，以便应用程序可以访问它；
7. 返回用户空间：最后，CPU从内核空间切换回用户空间，恢复应用程序的执行

五、实验心得体会

在本次实验中学习使用了GDB、Shell脚本编程、利用Makefile自动化构建项目，了解了Linux进程调度、进程通信、内核等知识。

其中GDB是一个功能强大的调试工具，可以帮助定位和解决程序中的错误和异常，了解GDB的基本用法和调试技巧对于快速定位和修复程序中的问题至关重要；

Shell脚本编程方便自动化执行命令，批量处理文件、任务等，可以简化复杂的任务和流程；

Makefile是用于自动化构建和管理软件项目的脚本文件，使用Makefile可以定义源文件的依赖关系、编译规则和链接选项，从而实现可靠的构建过程。学习Makefile编写可以提高代码的可维护性和可重复性，减少手动操作和错误。

平常写项目实践时一定要通过不断练习使用加深对这些工具的理解。