

**课 程 设 计**

**课程设计名称：** 操作系统课程设计

**专 业 班 级 ：** 物联网1602

**学 生 姓 名 ：** 王滨伟

**学 号 ：** 201616070213

**指 导 教 师 ：** 廖庆荣

**课程设计时间：** 2018年4月23日

**操作系统 课程设计任务书**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学生姓名** | 王滨伟 | **专业班级** | | **物联网1602** | **学院名称** | | **信息科学与工程学院** |
| **题 目** | 操作系统整合性shell用户接口和历史纪录实践 | | | | | | |
| **课题性质** | 操作系统整合实践 | | **课题来源** | | | 教师指定 | |
| **指导教师** | 廖庆荣 | | **同组姓名** | | | 无 | |
| **主要内容** | shell在UNIX或Linux操作系统中是一种很重要的使用者接口，可以透过命令行的方式输入指令。这个项目希望可以实现shell的操作方式，使用者可以在命令行的后方（>）输入指令，也可以加上（&）符号在背景执行，表示父进程（parent process）和子进程（child process）并行（concurrently）执行。例如编程结果可以执行如下：  osh>cat simple-shell.c  或是背景执行  osh>cat simple-shell.c &  例如下列五个指令也能在osh>底下运行：  exit：结束程序执行  history：显示历史纪录  ls：显示目录  mkdir：创建目录  rmdir：删除目录  并且可以将所有的执行记录下来，保存程历史纪录。这跟现行所有操作系统保存系统的log档案是一样的道理。 | | | | | | |
| **任务要求** | 当完成这个项目之后，运行程序可以显示提示字符osh>，接着可以下达命令，例如下列所示history、!!、!N（N为数字，代表第几个曾经执行过的指令）等。  1. history：显示最近的十笔历史纪录，如果历史纪录超过十笔，则会显示最近的十笔，但是号码仍会显示出来。例如有三十笔纪录，会显示第21-30笔。如果没有历史纪录，则会显示没有历史纪录的讯息。  2. !!：執行最近運行過的指令，如果没有历史執行纪录，则会显示没有指令可以執行的讯息。  3. !3：執行第三笔历史纪录，如果历史纪录没有第三笔，则会显示错误讯息没有这一笔历史纪录可供再執行一次。  4. 程序中使用fork()可以来启动进程。  5. 如果下列15项在osh中运行不出来，则我们必须重新撰写shell指令，使其正确运行。  6. 这些shell指令主要是在UNIX或Linux 操作系统上运行，如果你使用 Windows的环境运行，除非Windows操作系统完整支持，否则可能会发生程序出错或是shell指令无法运行。  以下这15项请各位同学填上指令功能说明请直接输入在程序中的批注说明。  (1). ps：  (2). ps -ael：  (3). ps aux：  (4). ls -l：  (5). ls -l|more：  (6). top：  (7). cal：  (8). who：  (9). date：  (10). pwd：  (11). mv：  (12). cp：  (13). file：  (14). cat：  (15). rm： | | | | | | |
| **参考文献** | 1. 汤小丹、梁红兵、哲凤屏、汤子瀛（2014）。计算机操作系统 第四版，西安电子科技大学出版社。 2. 梁红兵、汤小丹、汤子瀛（2014）。计算机操作系统 第四版 学习指导与题解，西安电子科技大学出版社。 3. Abraham Silberschatz, Greg Gagne, Peter B. Galvin (2013). Operating system concepts. John Wiley & Sons. | | | | | | |
| **审查意见** | **指导教师签字：**  **教研室主任签字： 年 月 日** | | | | | | |

说明：本表由指导教师填写，由教研室主任审核后下达给选题学生，装订在设计（论文）首页

**操作系统整合性shell用户接口和历史纪录实践**

# 摘要：

shell在UNIX或Linux操作系统中是一种很重要的使用者接口，可以透过命令行的方式输入指令。这个项目希望可以实现shell的操作方式，并且可以将所有的执行记录下来，保存程历史纪录，这跟现行所有操作系统保存系统的log档案是一样的道理。在这个实践过程，能熟悉Linux的指令操作，同时也能以指令操作了解进程的产生、进程的管理，以及系统的调用等功能，充分了解操作系统的指令运行。

# 一、设计目的

1. 能孰悉Linux的shell指令
2. 能实践整合性的操作系统shell用户接口指令设计
3. 能实现操作系统的历史纪录
4. 能了解和实现进程如何产生以及运作
5. 能实现进程产生
6. 能实现操作系统功能调用

# 二、设计要求与内容

shell在UNIX或Linux操作系统中是一种很重要的使用者接口，可以透过命令行的方式输入指令。这个项目希望可以实现shell的操作方式，使用者可以在命令行的后方（>）输入指令，也可以加上（&）符号在背景执行，表示父进程（parent process）和子进程（child process）并行（concurrently）执行。例如编程结果可以执行如下：

osh>cat simple-shell.c

或是背景执行

osh>cat simple-shell.c &

　　例如下列五个指令也能在osh>底下运行：

exit：结束程序执行

history：显示历史纪录

ls：显示目录

mkdir：创建目录

rmdir：删除目录

并且可以将所有的执行记录下来，保存程历史纪录。这跟现行所有操作系统保存系统的log档案是一样的道理。

当完成这个项目之后，运行程序可以显示提示字符osh>，接着可以下达命令，例如下列所示history、!!、!N（N为数字，代表第几个曾经执行过的指令）等。

1. history：显示最近的十笔历史纪录，如果历史纪录超过十笔，则会显示最近的十笔，但是号码仍会显示出来。例如有三十笔纪录，会显示第21-30笔。如果没有历史纪录，则会显示没有历史纪录的讯息。

2. !!：執行最近運行過的指令，如果没有历史執行纪录，则会显示没有指令可以執行的讯息。

3. !3：執行第三笔历史纪录，如果历史纪录没有第三笔，则会显示错误讯息没有这一笔历史纪录可供再執行一次。

4. 程序中使用fork()可以来启动进程。

5. 如果下列15项在osh中运行不出来，则我们必须重新撰写shell指令，使其正确运行。

6. 这些shell指令主要是在UNIX或Linux 操作系统上运行，如果你使用 Windows的环境运行，除非Windows操作系统完整支持，否则可能会发生程序出错或是shell指令无法运行。

以下这15项请各位同学填上指令功能说明请直接输入在程序中的批注说明。

(1).ps:process status,进程状态，基本功能进程查看命令

(2).ps -ael:显示所有终端机下执行的程序，除了现阶段作业领导者之外。列出程序时，显示每个程序所使用的环境变量。采用详细的格式来显示程序状态。

(3).ps aux:显示所有终端机下执行的程序，包括其他用户的程序。以用户为主的格式来显示程序的状态。显示所有程序，不以终端机来区分。

(4).ls -l: 显示指定工作目录下的内容。将文件型态、权限、拥有者、文件大小等详细资讯详细列出。

(5).ls -l|more: ‘|’管道符，ls的内容为more的输入

(6).top: 用来监控linux的系统状况，显示cpu、内存的使用，动态展现各进程的字段排序

(7).cal: 用来显示日历

(8).who:简单显示当前登录系统用户的信息，可以轻松的获取当前登录系统的用户列表，包含使用终端登录。

(9).date:可以用来显示或设定系统的日期与时间

(10).pwd:是Print Working Directory的缩写，其功能是显示当前所在工作目录的全路径。主要用在当不确定当前所在位置时，通过pwd来查看当前目录的绝对路径

(11).mv:用来对文件或目录重新命名，或者将文件从一个目录移到另一个目录中。source表示源文件或目录，target表示目标文件或目录。如果将一个文件移到一个已经存在的目标文件中，则目标文件的内容将被覆盖

(12).cp:命令用来将一个或多个源文件或者目录复制到指定的目的文件或目录。它可以将单个源文件复制成一个指定文件名的具体的文件或一个已经存在的目录下。cp命令还支持同时复制多个文件，当一次复制多个文件时，目标文件参数必须是一个已经存在的目录，否则将出现错误。

(13).file:用来探测给定文件的类型。file命令对文件的检查分为文件系统、魔法幻数检查和语言检查3个过程。

(14).cat:是linux下的一个文本输出命令，通常是用于观看某个文件的内容的

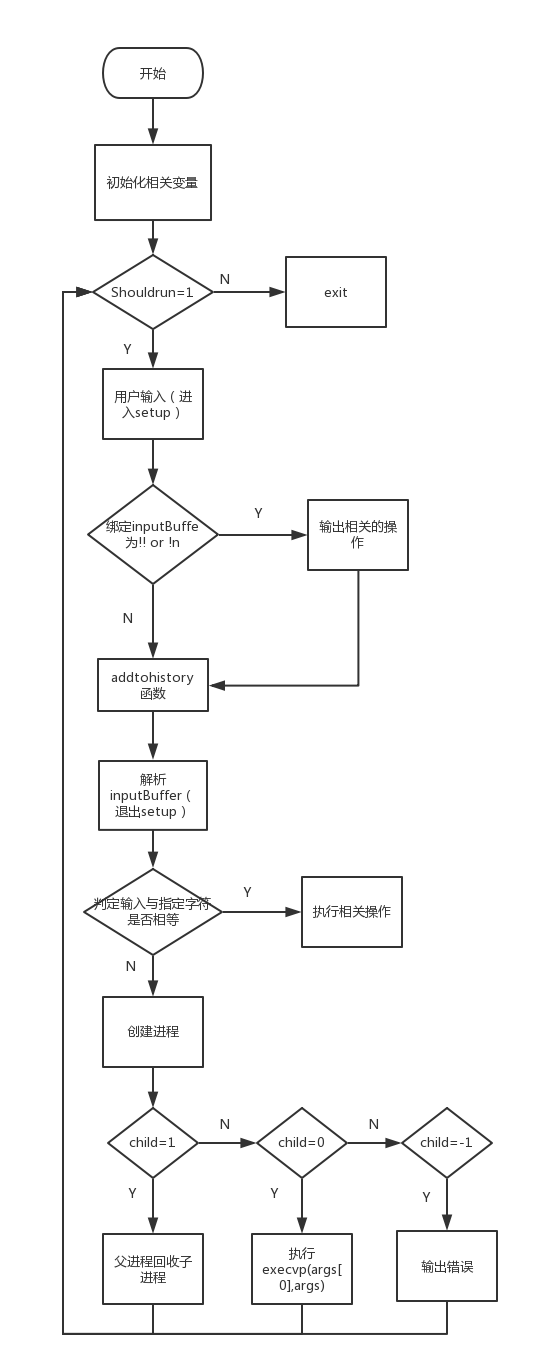
(15).rm:可以删除文件和文件夹，但是文件夹不能为空。

# 设计原理

利用setup()函数记录输入指令，并进行解析到argc上

利用execvp()函数，会从PATH 环境变量所指的目录中查找符合参数file 的文件名, 找到后便执行该文件, 然后将第二个参数argv 传给该欲执行的文件。

四、流程圖

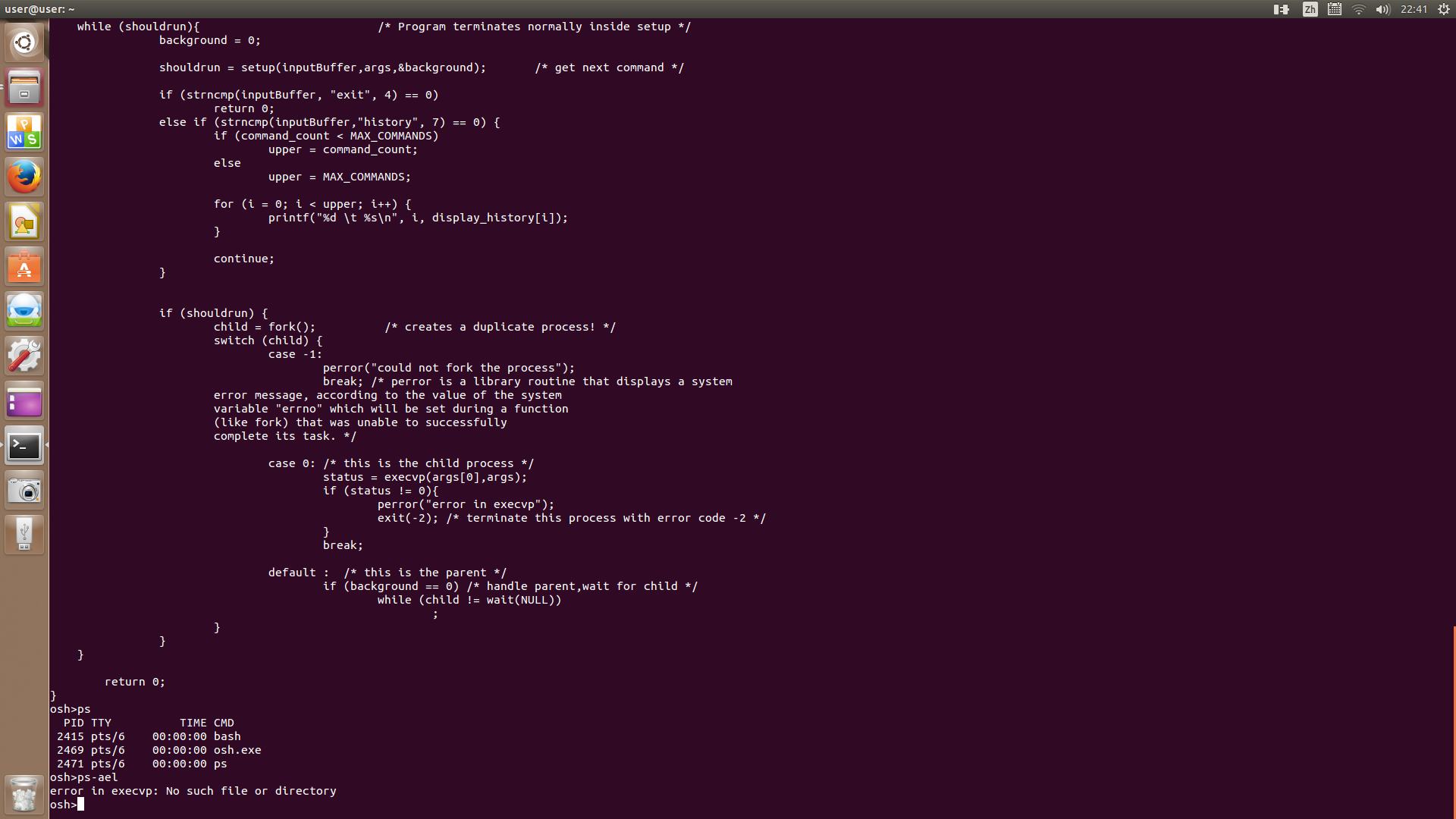


# 测试结果與說明

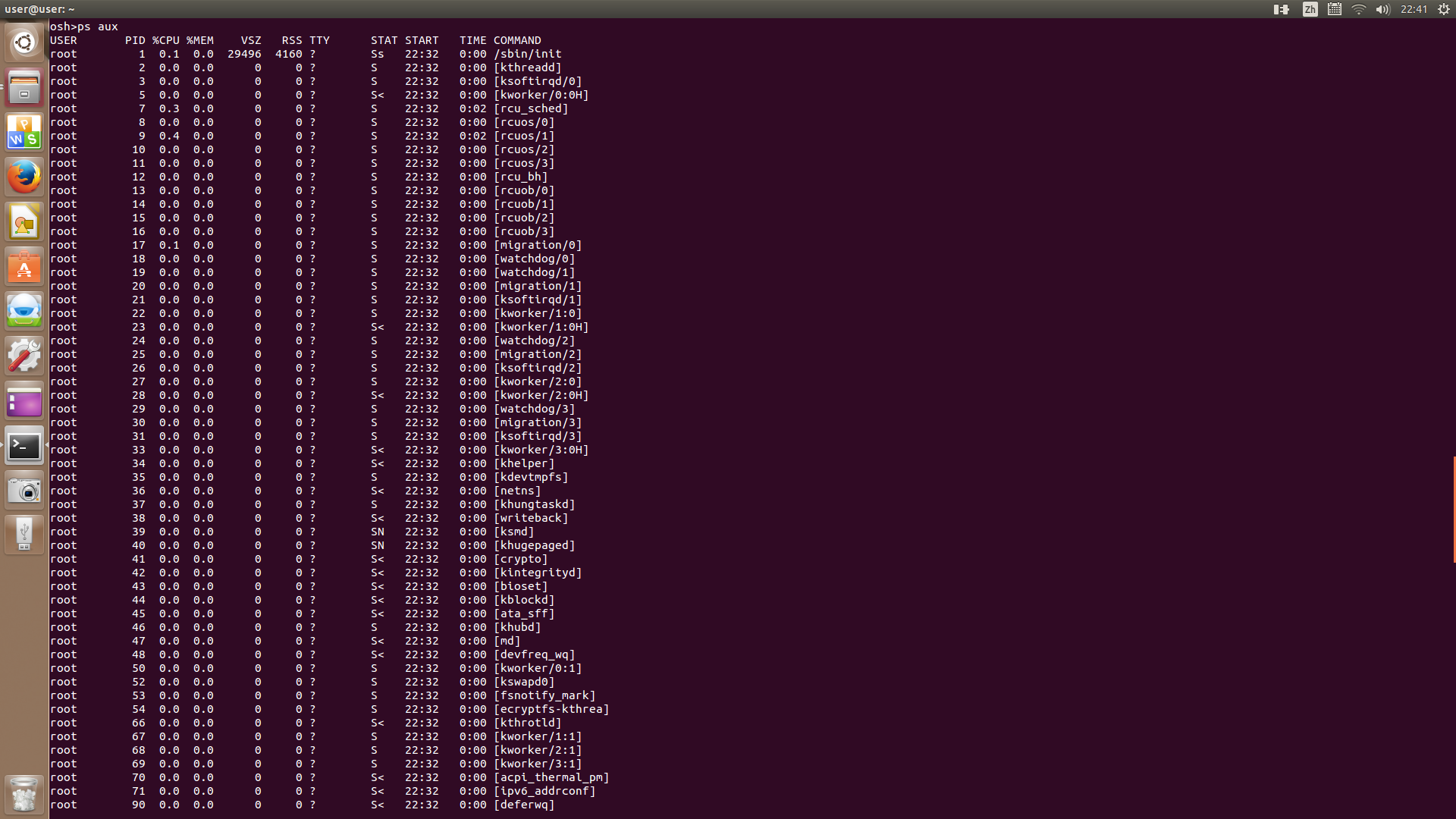
**ps命令：**

****

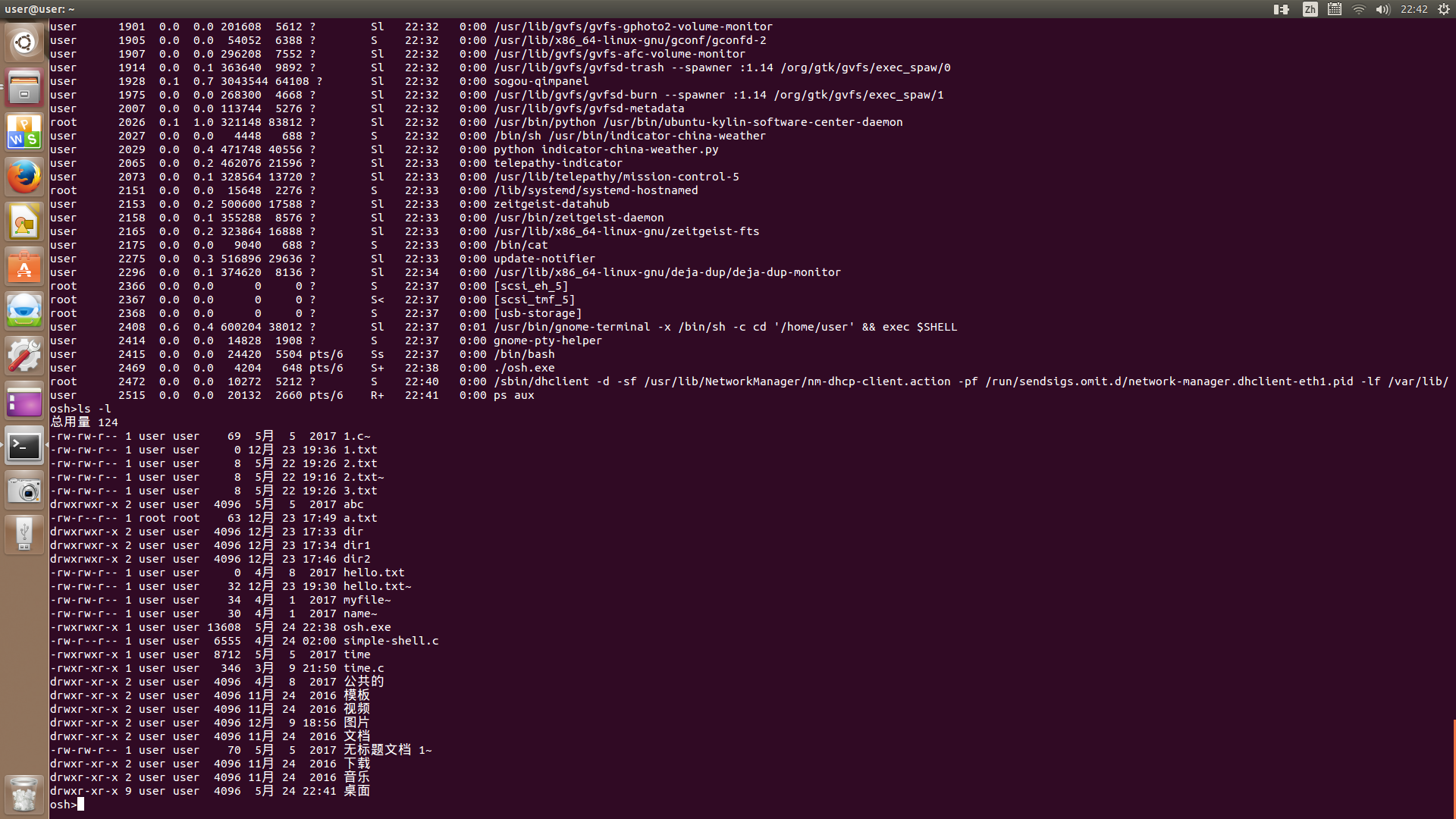
**ps –ael命令：**



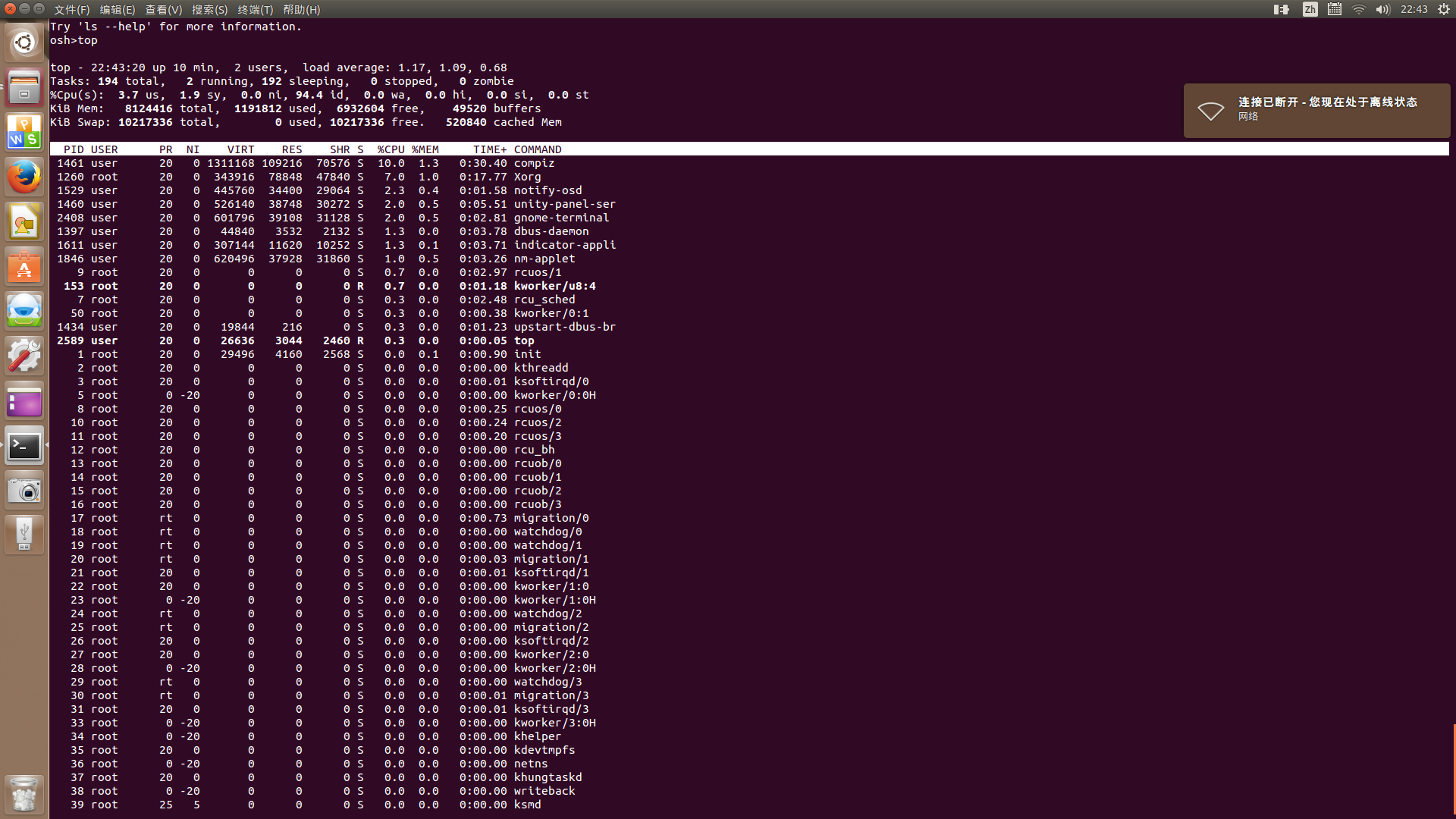
ps aux 命令：



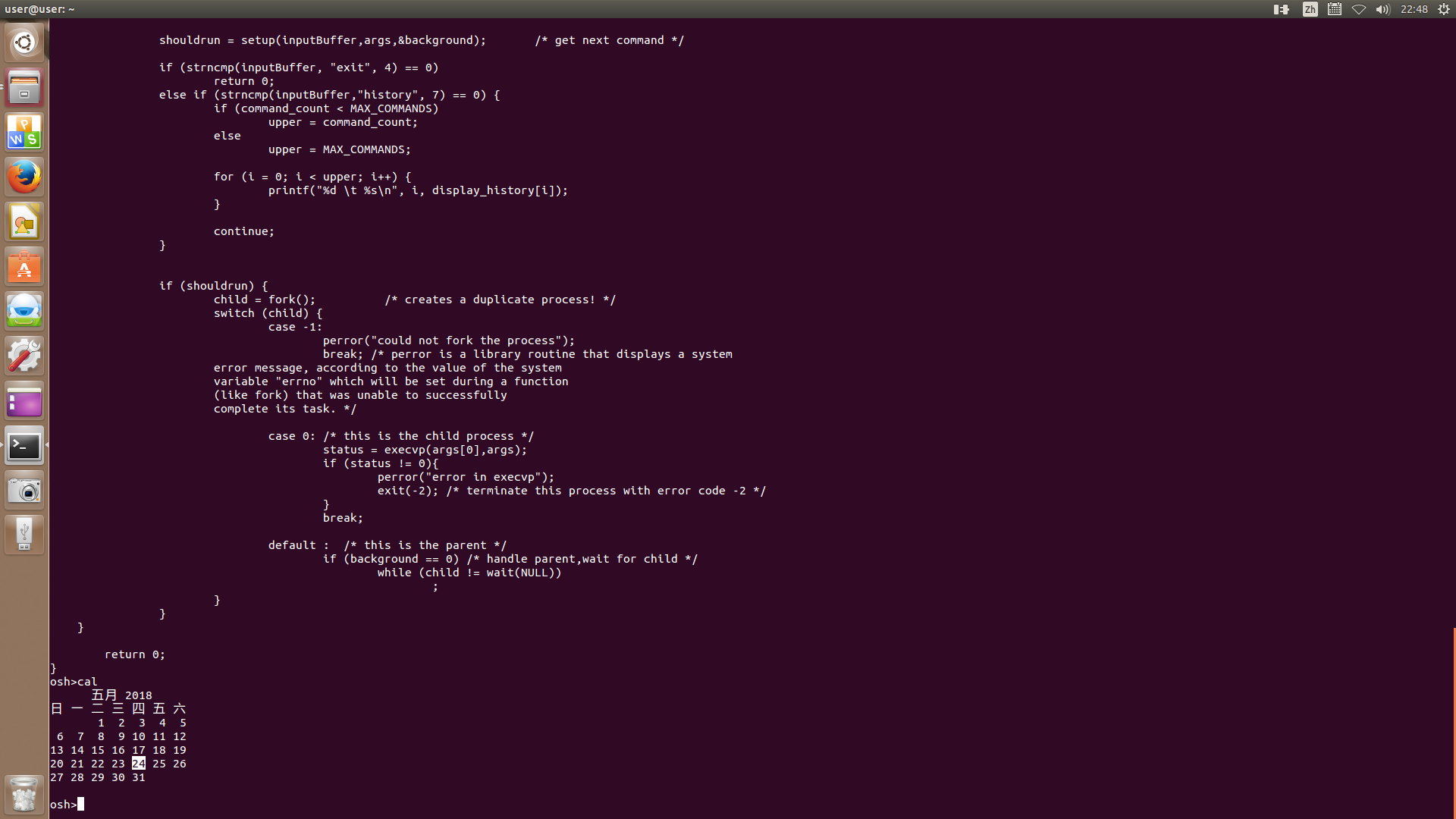
**ls –l 命令：**

****

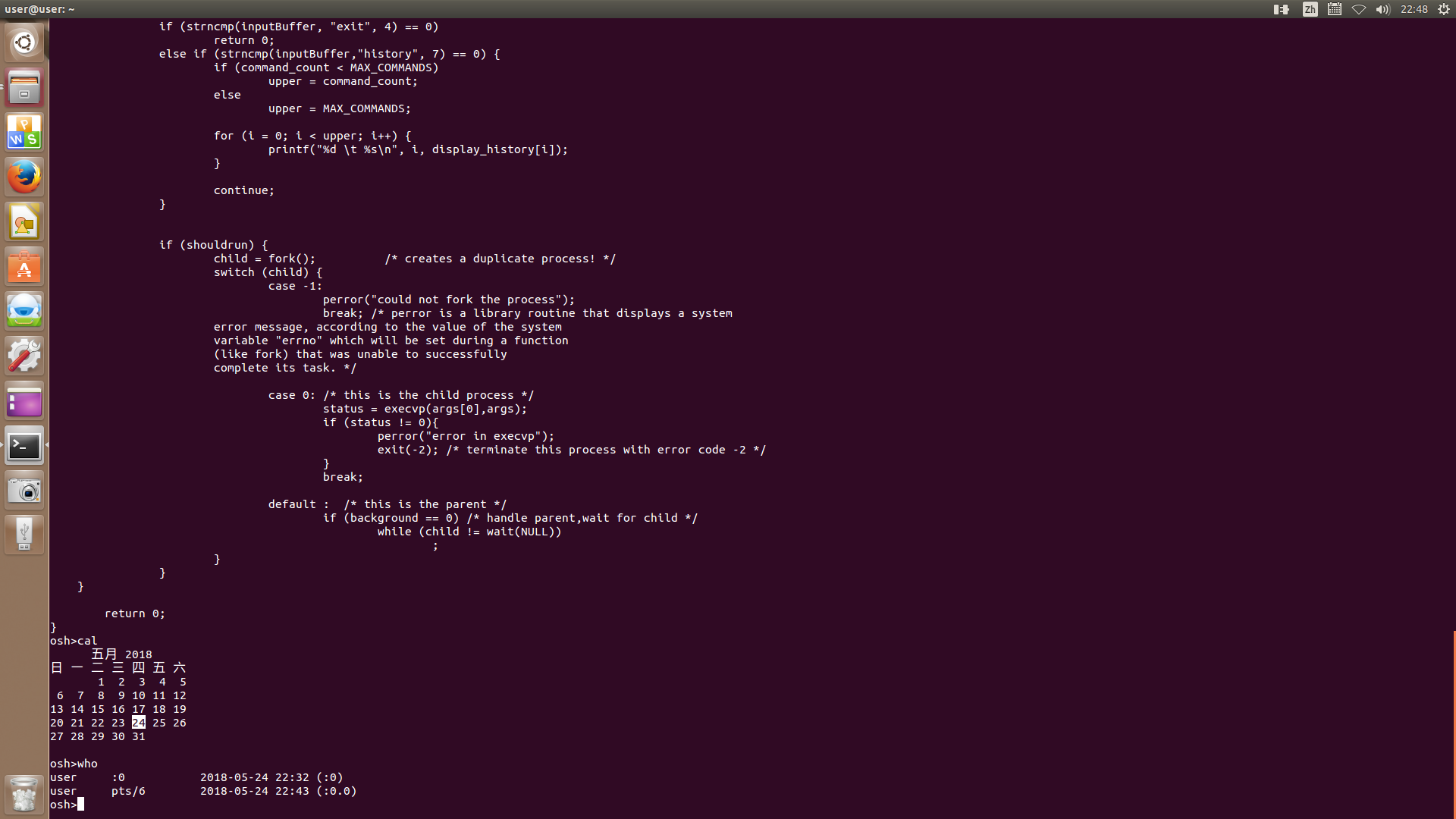
**top命令：**

****

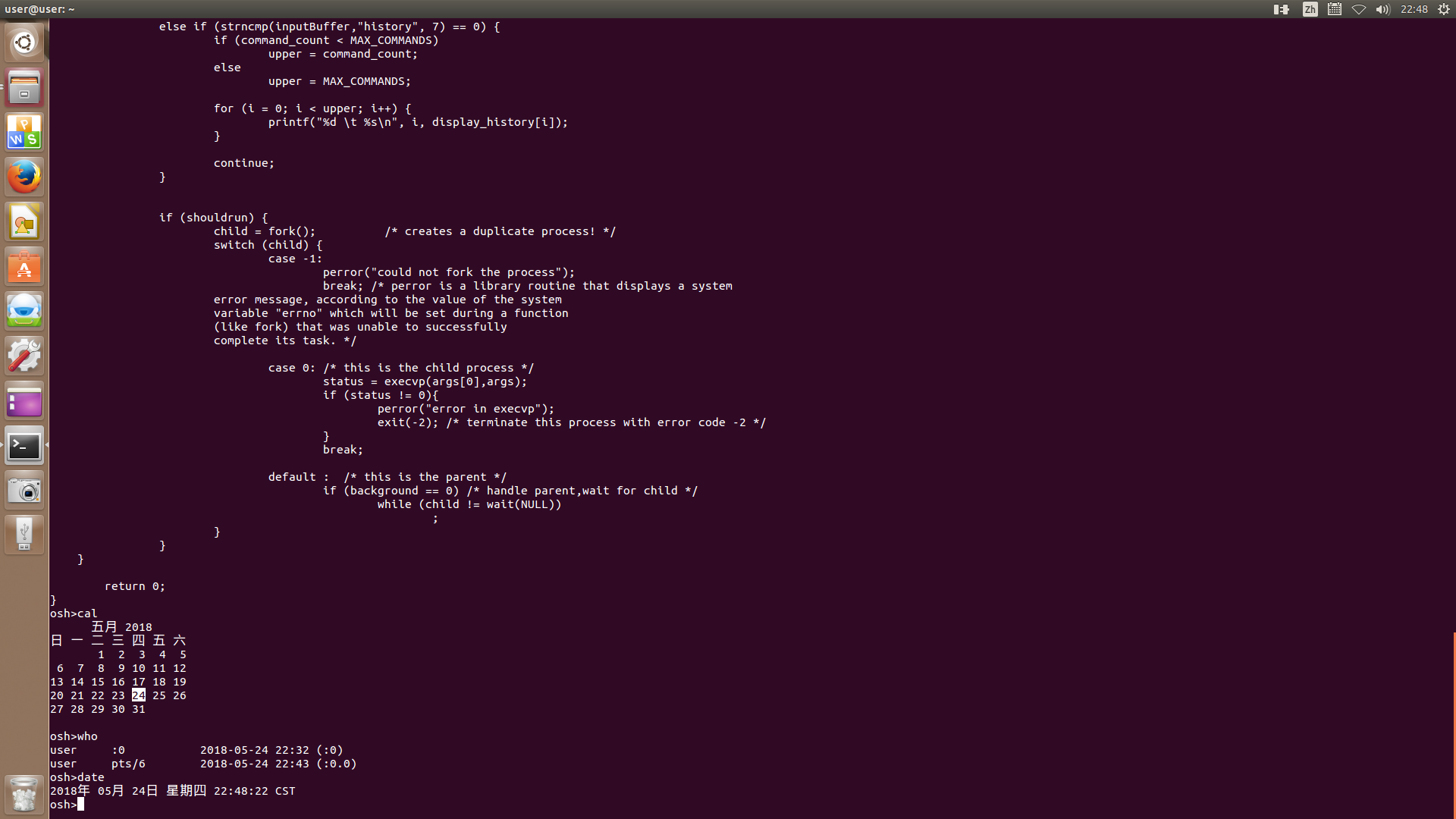
**cal 命令：**



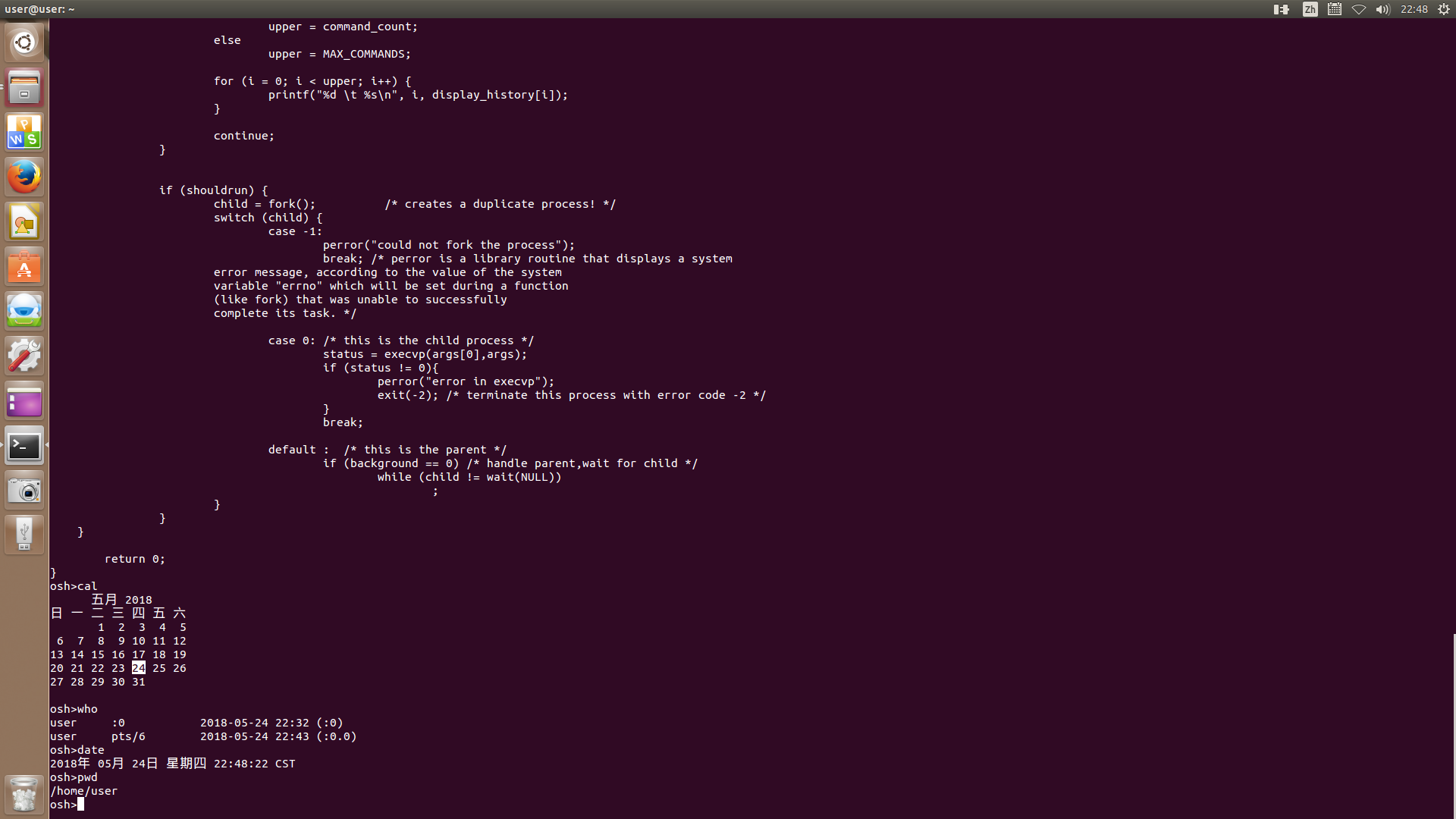
**who 命令：**



**date 命令：**



**pwd 命令：**

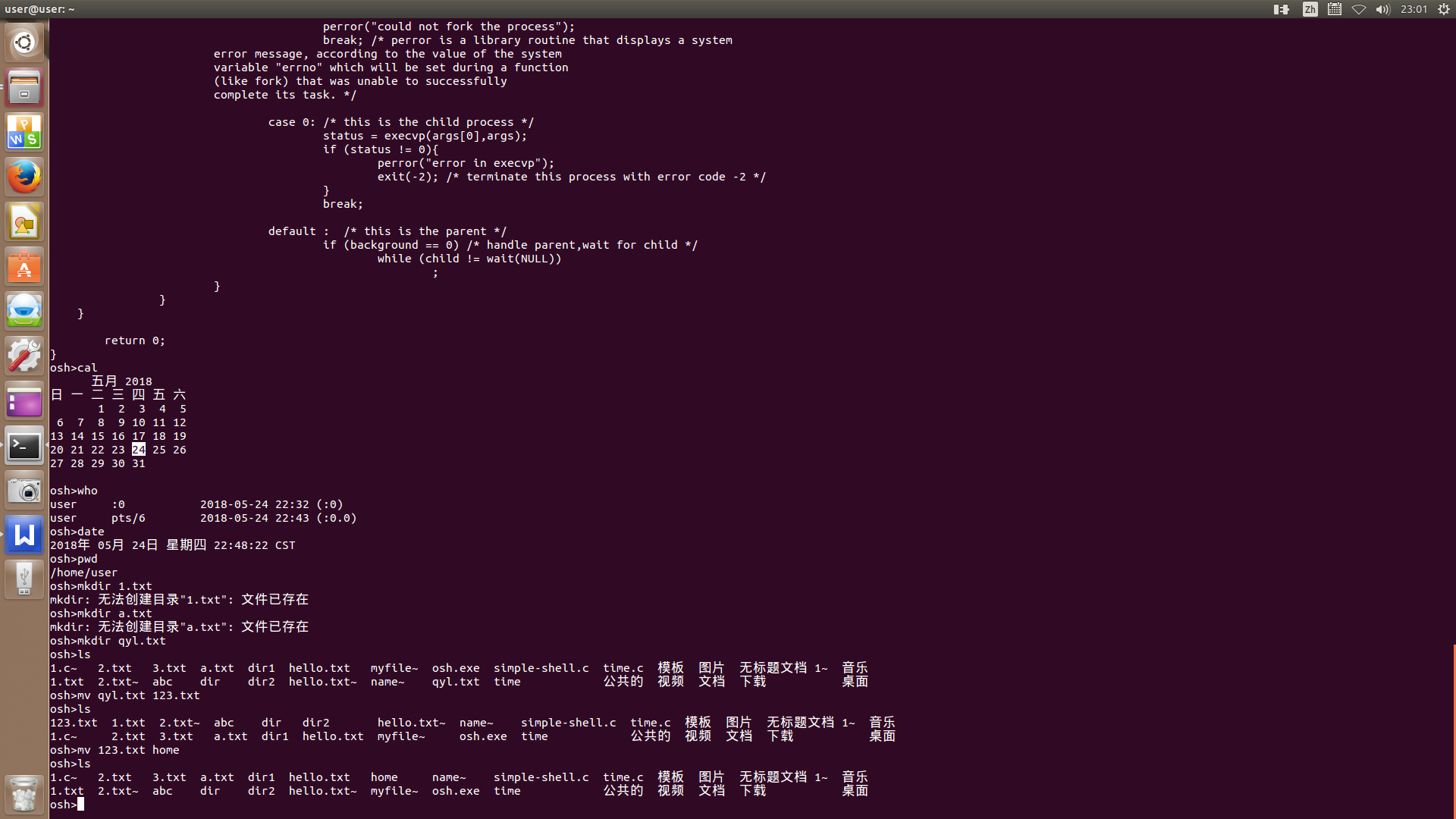


**mv命令：**

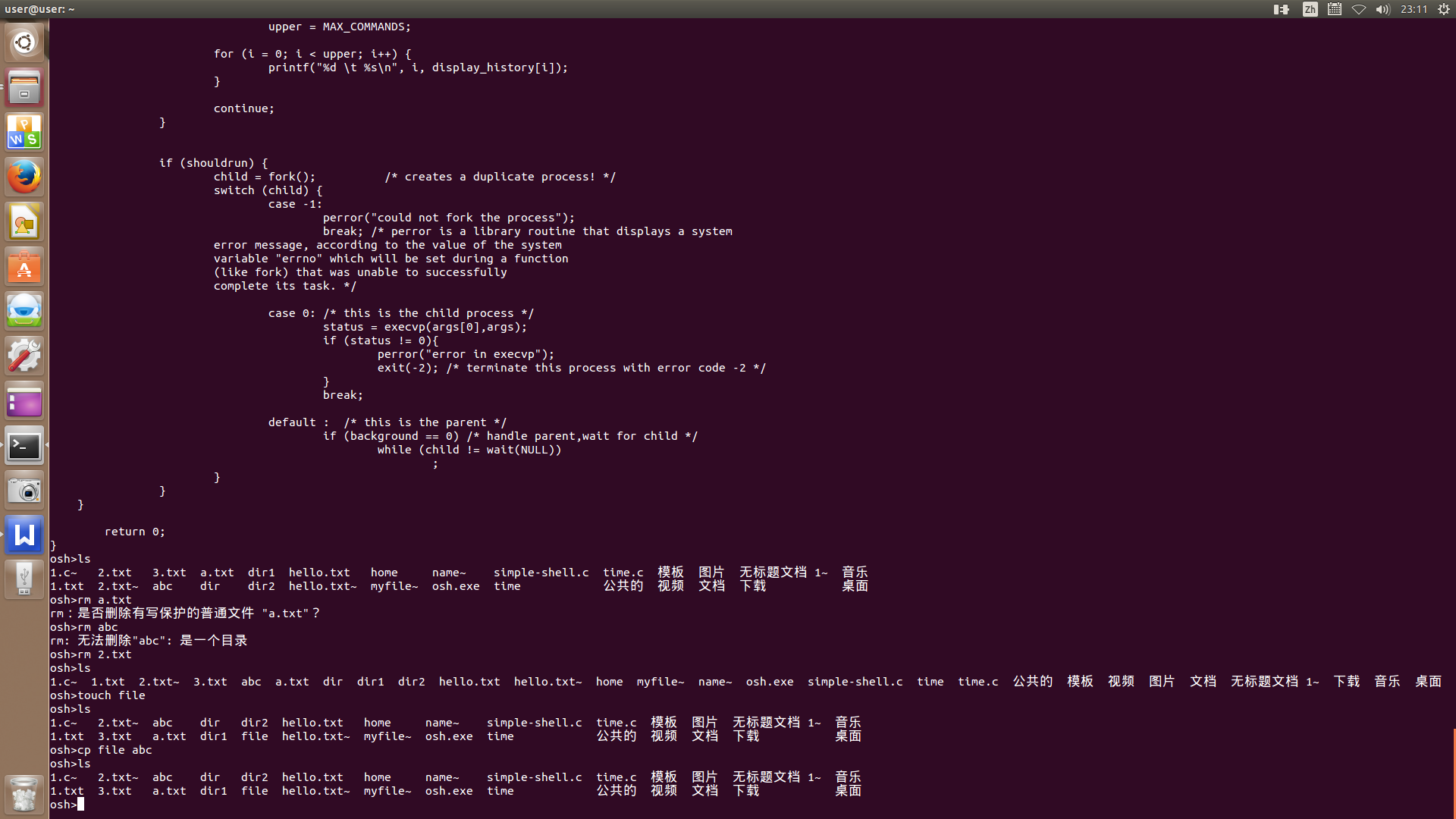
1.改名



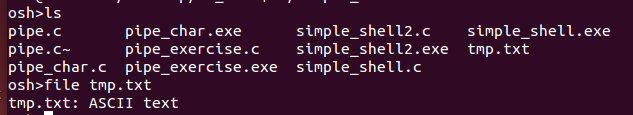
2.移动文件

****

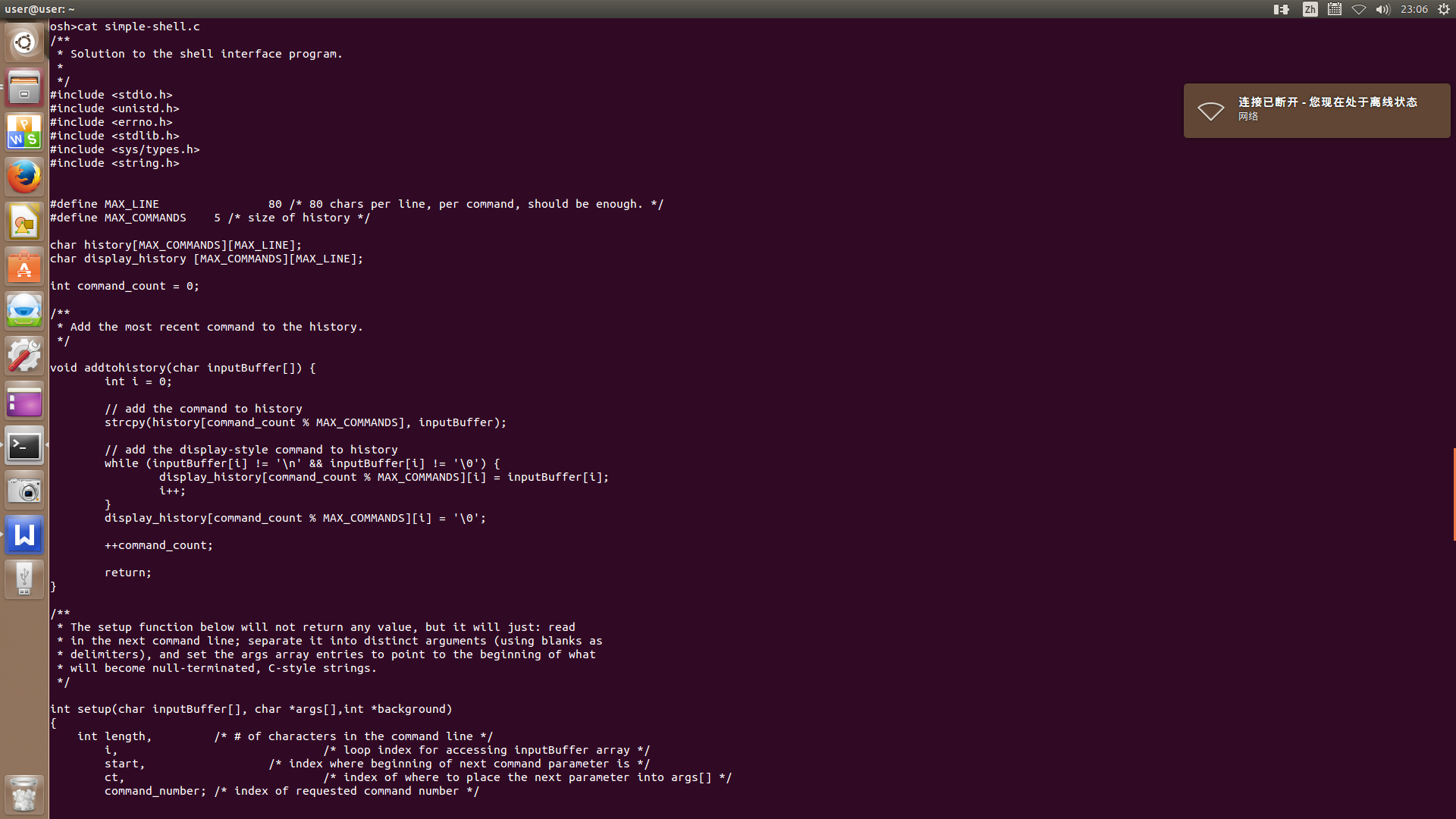
**cp 命令：**



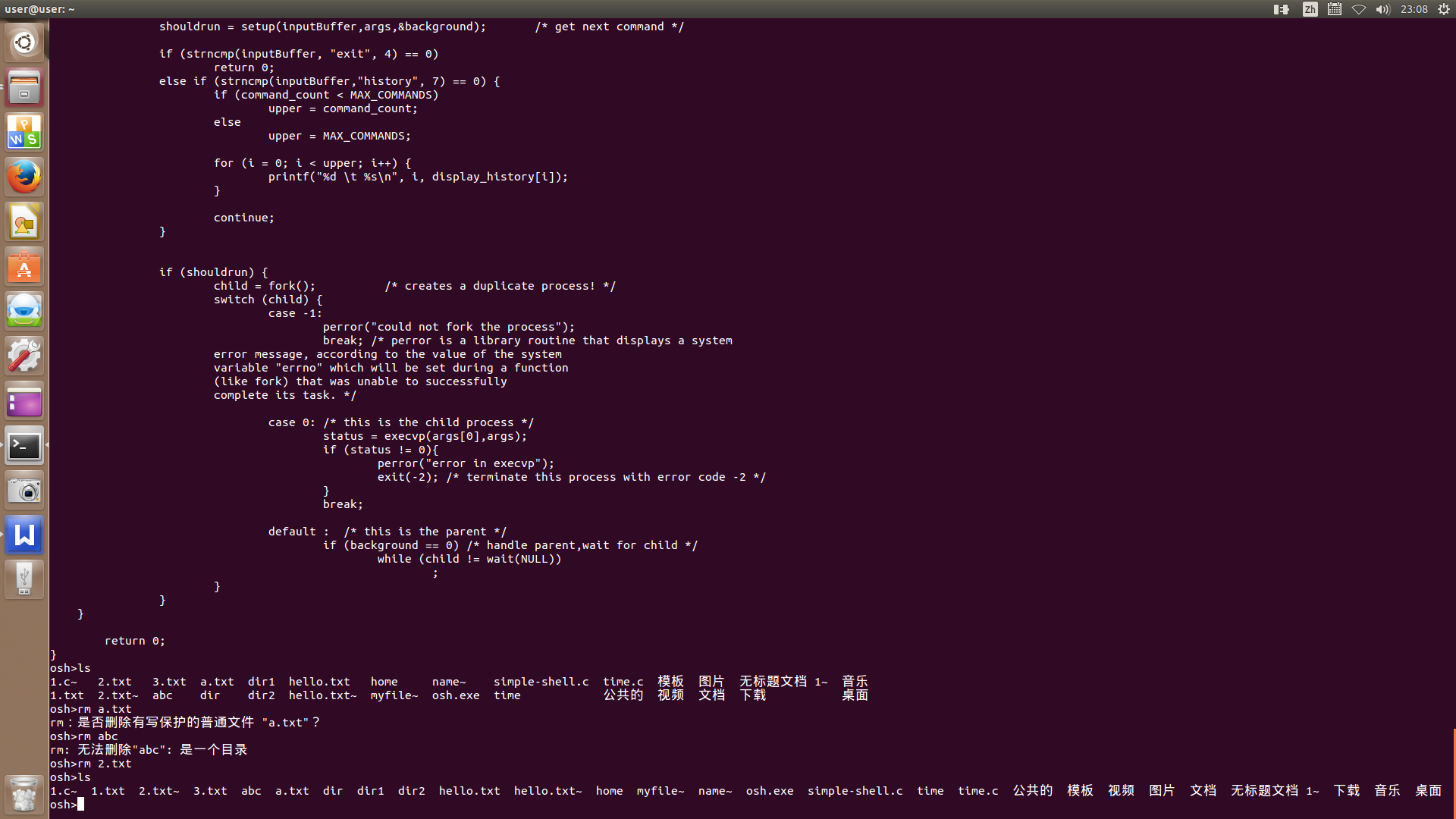
**file命令：**



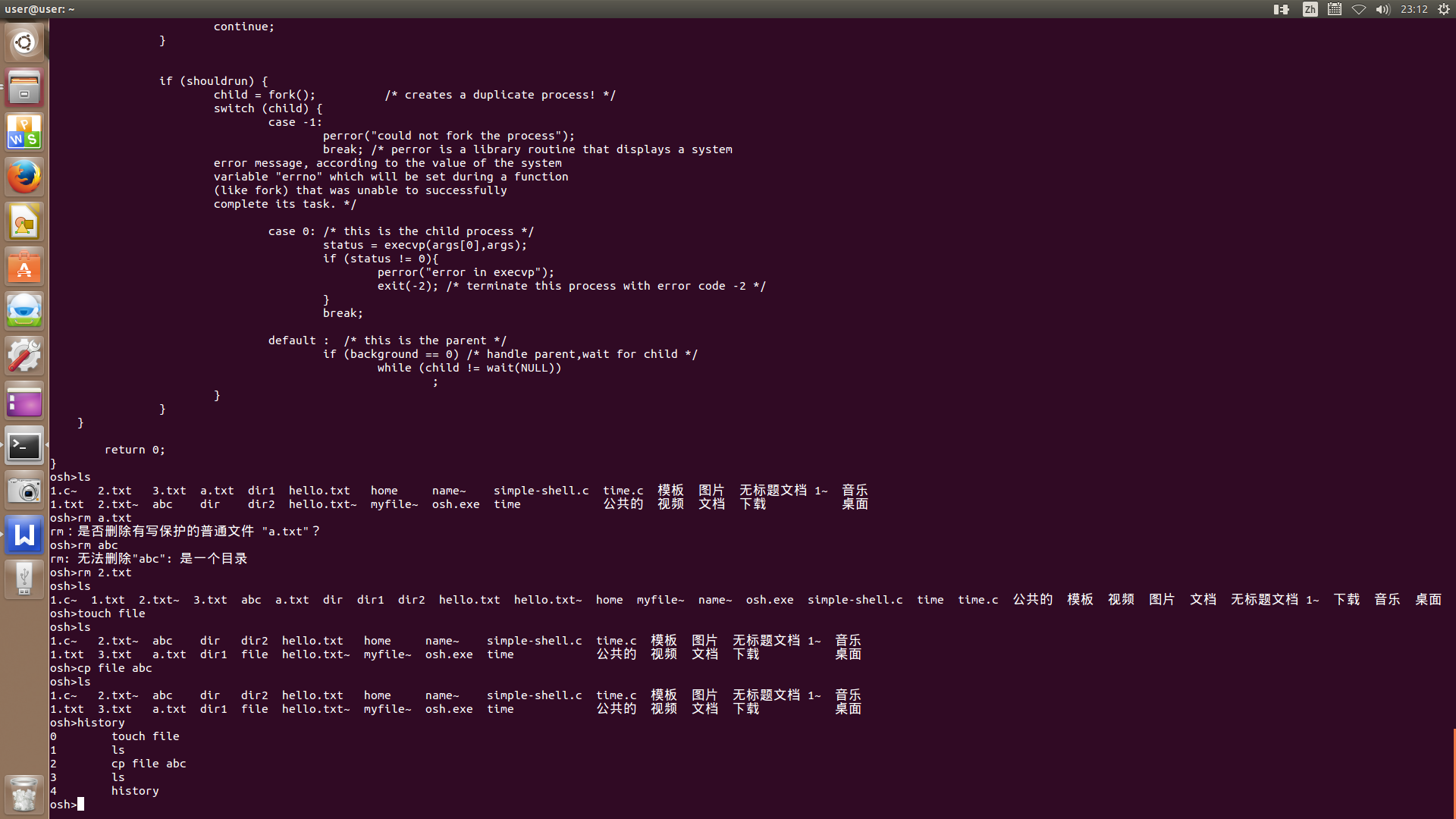
**Cat命令：**



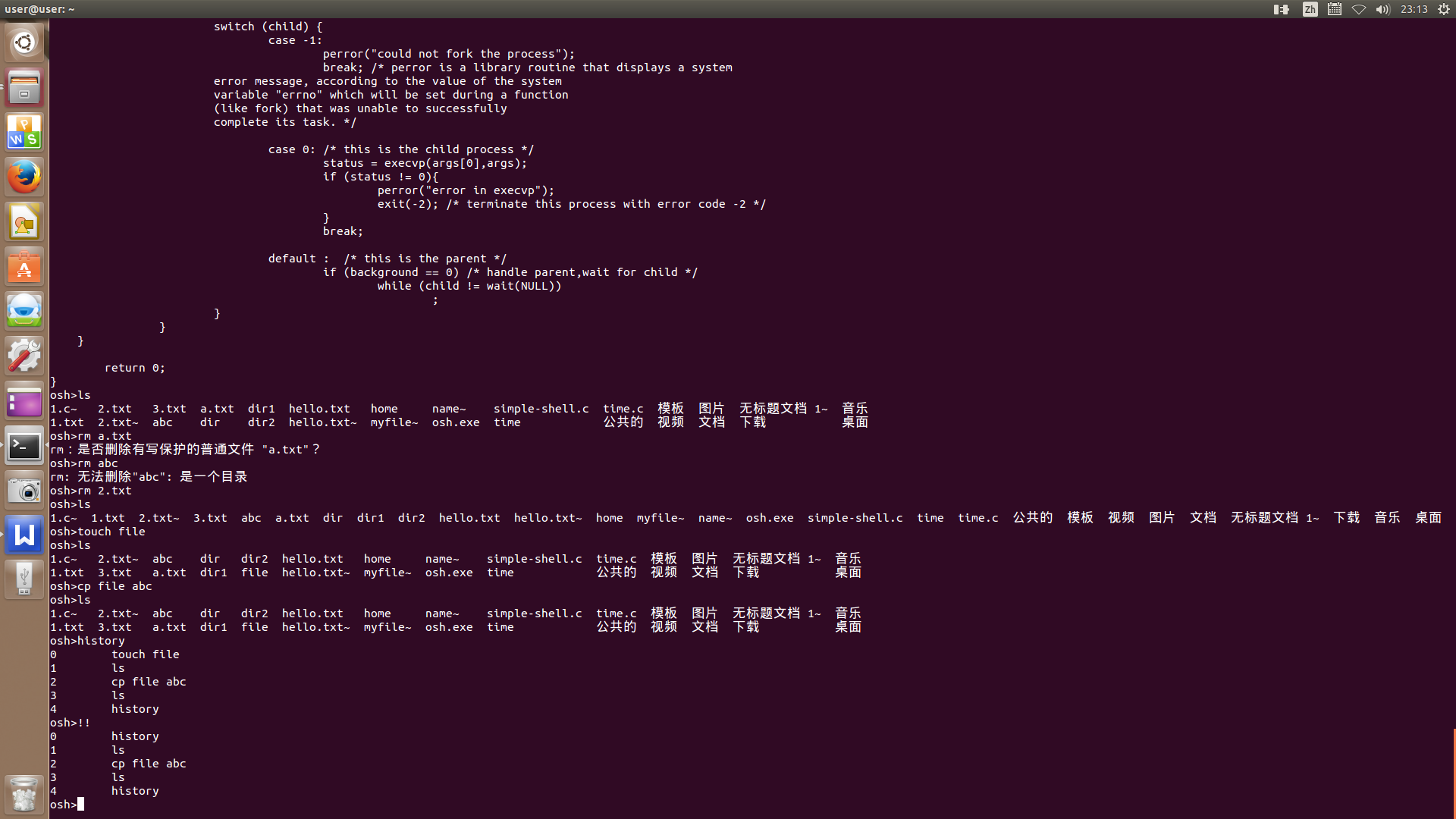
**rm 命令：**



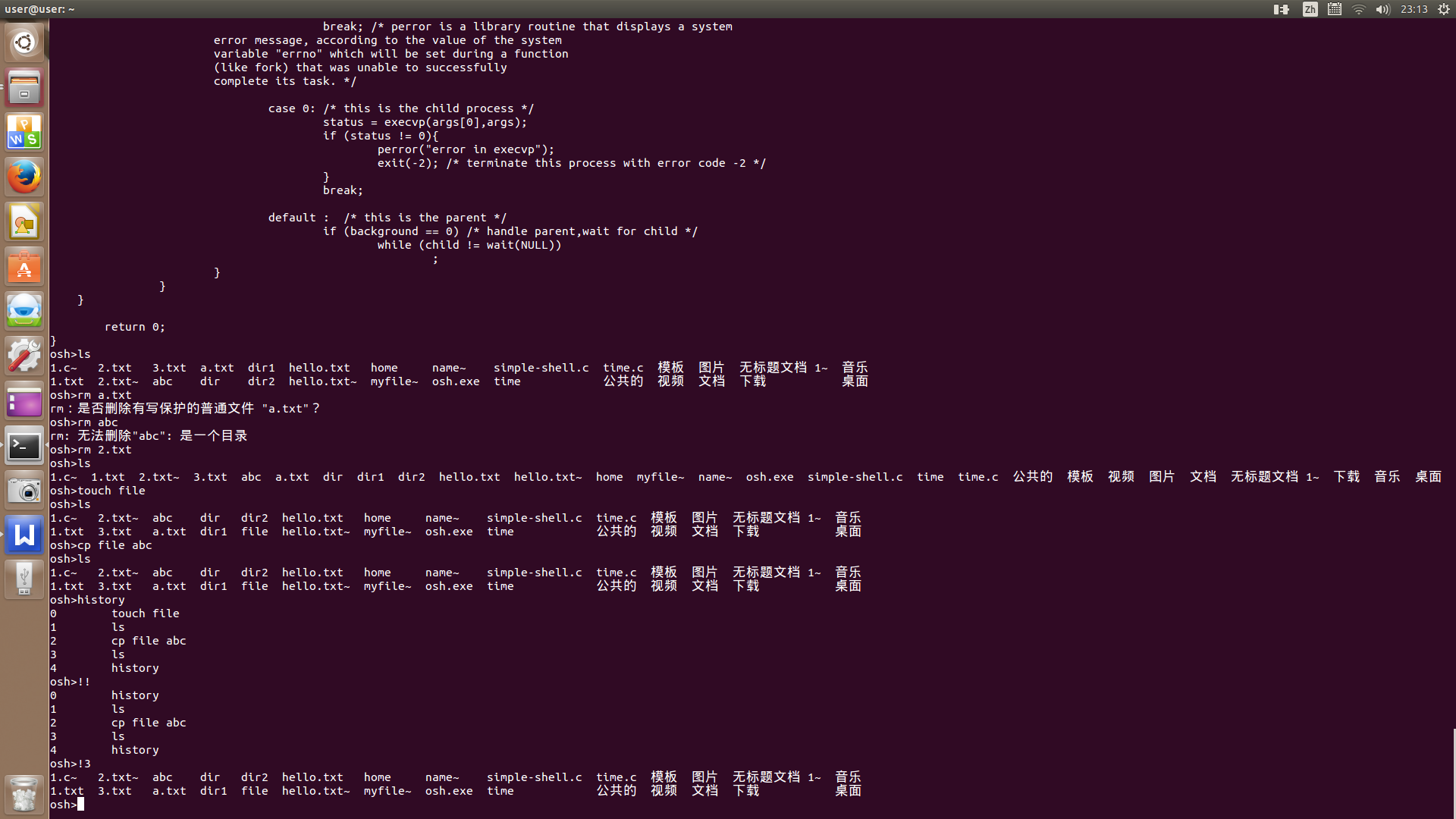
**history 命令：**



**！！命令：**



**！3命令：**



# 六、收获、体会和建议

**学会了进程和线程的具体操作，掌握了pipe、excel、dup2函数的使用。对linux命令指令有了更清晰的认识，明白了linux的如何实现。**

# 附录

# 1.源碼

**/\*\***

**\* Solution to the shell interface program.**

**\***

**\*/**

**#include <stdio.h>**

**#include <unistd.h>**

**#include <errno.h>**

**#include <stdlib.h>**

**#include <sys/types.h>**

**#include <string.h>**

**#define MAX\_LINE 80 /\* 80 chars per line, per command, should be enough. \*/**

**#define MAX\_COMMANDS 50 /\* size of history \*/**

**char history[MAX\_COMMANDS][MAX\_LINE]; //历史记录**

**char display\_history [MAX\_COMMANDS][MAX\_LINE];**

**int command\_count = 0;//总数**

**/\*\***

**\* Add the most recent command to the history.**

**\*/**

**void addtohistory(char inputBuffer[]) {**

**int i = 0;**

**// add the command to history**

**strcpy(history[command\_count % MAX\_COMMANDS], inputBuffer); //复制inputBuffer到history**

**// add the display-style command to history**

**while (inputBuffer[i] != '\n' && inputBuffer[i] != '\0') {**

**display\_history[command\_count % MAX\_COMMANDS][i] = inputBuffer[i];**

**i++;**

**}**

**display\_history[command\_count % MAX\_COMMANDS][i] = '\0';**

**++command\_count;**

**return;**

**}**

**/\*\***

**\* The setup function below will not return any value, but it will just: read**

**\* in the next command line; separate it into distinct arguments (using blanks as**

**\* delimiters), and set the args array entries to point to the beginning of what**

**\* will become null-terminated, C-style strings.**

**\*/**

**int setup(char inputBuffer[], char \*args[],int \*background)**

**{**

**int length, /\* # of characters in the command line \*/**

**i, /\* loop index for accessing inputBuffer array \*/**

**start, /\* index where beginning of next command parameter is \*/ //命令的第一个字符位置**

**ct, /\* index of where to place the next parameter into args[] \*/ //下一个参数存入args[]的位置**

**command\_number; /\* index of requested command number \*/**

**ct = 0;**

**/\* read what the user enters on the command line \*/**

**do {**

**printf("osh>");**

**fflush(stdout);**

**length = read(STDIN\_FILENO,inputBuffer,MAX\_LINE);**

**}**

**while (inputBuffer[0] == '\n'); /\* swallow newline characters \*/**

**/\*\***

**\* 0 is the system predefined file descriptor for stdin (standard input),**

**\* which is the user's screen in this case. inputBuffer by itself is the**

**\* same as &inputBuffer[0], i.e. the starting address of where to store**

**\* the command that is read, and length holds the number of characters**

**\* read in. inputBuffer is not a null terminated C-string.**

**\*/**

**start = -1;**

**if (length == 0)**

**exit(0); /\* ^d was entered, end of user command stream \*/**

**/\*\***

**\* the <control><d> signal interrupted the read system call**

**\* if the process is in the read() system call, read returns -1**

**\* However, if this occurs, errno is set to EINTR. We can check this value**

**\* and disregard the -1 value**

**\*/**

**if ( (length < 0) && (errno != EINTR) ) {**

**perror("error reading the command");**

**exit(-1); /\* terminate with error code of -1 \*/**

**}**

**/\*\***

**\* Check if they are using history**

**\*/**

**if (inputBuffer[0] == '!') {**

**if (command\_count == 0) {**

**printf("No history\n");**

**return 1;**

**}**

**else if (inputBuffer[1] == '!') {**

**// restore the previous command //restore修复、归还**

**int upper;**

**printf("command\_count=%d\n",command\_count);**

**if(command\_count <=10)**

**upper = command\_count-1;**

**else**

**upper = 9 ;**

**for(i=command\_count-upper;i<=command\_count;i++)**

**{**

**printf("%d \t %s\n",i,display\_history[i]);**

**}**

**//strcpy(inputBuffer,"!!");**

**length = strlen(inputBuffer) + 1;**

**}**

**else if (isdigit(inputBuffer[1])) { /\* retrieve the nth command \*/ //取回、重新得到**

**command\_number = atoi(&inputBuffer[1]);**

**//strcpy(inputBuffer,history[command\_number]);**

**printf("%d \t %s\n",command\_number,display\_history[command\_number]);**

**length = strlen(inputBuffer) + 1;**

**}**

**}**

**/\*\***

**\* Add the command to the history**

**\*/**

**addtohistory(inputBuffer);**

**/\*\***

**\* Parse the contents of inputBuffer**

**\*/**

**//printf("inputBuffer:%s\n",inputBuffer);**

**for (i=0;i<length;i++) {**

**/\* examine every character in the inputBuffer \*/**

**switch (inputBuffer[i]){**

**case ' ':**

**case '\t' : /\* argument separators \*/**

**if(start != -1){**

**args[ct] = &inputBuffer[start]; /\* set up pointer \*/**

**ct++;**

**}**

**inputBuffer[i] = '\0'; /\* add a null char; make a C string \*/**

**start = -1;**

**break;**

**case '\n': /\* should be the final char examined \*/**

**if (start != -1){**

**args[ct] = &inputBuffer[start];**

**ct++;**

**}**

**inputBuffer[i] = '\0';**

**args[ct] = NULL; /\* no more arguments to this command \*/**

**break;**

**default : /\* some other character \*/**

**if (start == -1)**

**start = i;**

**if (inputBuffer[i] == '&') {**

**\*background = 1;**

**inputBuffer[i-1] = '\0';**

**}**

**} /\* end of switch \*/**

**} /\* end of for \*/**

**/\*\***

**\* If we get &, don't enter it in the args array**

**\*/**

**if (\*background)**

**args[--ct] = NULL;**

**args[ct] = NULL; /\* just in case the input line was > 80 \*/**

**return 1;**

**} /\* end of setup routine \*/**

**void run\_shell(char \*args[MAX\_LINE/2 + 1])**

**{**

**int pipe\_fd[2];**

**int status;**

**pipe(pipe\_fd);**

**pid\_t child1,child2;**

**if ((child1=fork())!=0)//父进程**

**{**

**if ( (child2 = fork()) == 0 )//子进程**

**{**

**close ( pipe\_fd[1] );**

**close ( fileno ( stdin ) );**

**dup2 ( pipe\_fd[0] , fileno(stdin));**

**close ( pipe\_fd[0] );**

**execvp(args[0],args);**

**}**

**else**

**{**

**close ( pipe\_fd[0]);**

**close ( pipe\_fd[1]);**

**waitpid ( child2 , &status , 0 );**

**}**

**waitpid ( child1 , &status , 0 );**

**}**

**else**

**{**

**printf ( "subshell 3cmd %d\n" , getpid() );**

**close ( pipe\_fd[0] );**

**close ( fileno(stdout));**

**dup2 ( pipe\_fd[1] , fileno(stdout));**

**close ( pipe\_fd[1] );**

**execvp(args[0],args);**

**}**

**}**

**int main(void)**

**{**

**char inputBuffer[MAX\_LINE]; /\* buffer to hold the command entered \*/**

**int background; /\* equals 1 if a command is followed by '&' \*/**

**char \*args[MAX\_LINE/2 + 1]; /\* command line (of 80) has max of 40 arguments \*/**

**pid\_t child; /\* process id of the child process \*/**

**int status; /\* result from execvp system call\*/**

**int shouldrun = 1;**

**int i, upper;**

**while (shouldrun){ /\* Program terminates normally inside setup \*/**

**background = 0;**

**shouldrun = setup(inputBuffer,args,&background); /\* get next command \*/**

**if (strncmp(inputBuffer, "exit", 4) == 0)**

**return 0;**

**else if (strncmp(inputBuffer,"history", 7) == 0) {**

**if (command\_count < MAX\_COMMANDS)**

**upper = command\_count;**

**else**

**upper = MAX\_COMMANDS;**

**for (i = 0; i < upper; i++) {**

**printf("%d \t %s\n", i, display\_history[i]);**

**}**

**continue;**

**}**

**if (shouldrun) {**

**child = fork(); /\* creates a duplicate process! \*/**

**switch (child) {**

**case -1:**

**perror("could not fork the process");**

**break; /\* perror is a library routine that displays a system**

**error message, according to the value of the system**

**variable "errno" which will be set during a function**

**(like fork) that was unable to successfully**

**complete its task. \*/**

**case 0: /\* this is the child process \*/**

**//printf("args:%s\n",args[0]);**

**status = execvp(args[0],args);//args[0]是指令,args存的是全部**

**if (status != 0){**

**perror("error in execvp");**

**exit(-2); /\* terminate this process with error code -2 \*/**

**}**

**break;**

**default : /\* this is the parent \*/**

**if (background == 0) /\* handle parent,wait for child \*/**

**while (child != wait(NULL)) //父进程回收子进程**

**//如果不在后台运行就回收子进程**

**;**

**}**

**}**

**}**

**return 0;**

**}**

# 主要参考文献

1. 汤小丹、梁红兵、哲凤屏、汤子瀛（2014）。计算机操作系统 第四版，西安电子科技大学出版社。
2. 梁红兵、汤小丹、汤子瀛（2014）。计算机操作系统 第四版 学习指导与题解，西安电子科技大学出版社。
3. A. Silberschatz, G. Gagne, P. B. Galvin (2013). Operating system concepts. John Wiley & Sons.

信息科学与工程学院 **学院课程设计成绩评价表**

课程名称：操作系統課程設計

设计题目：操作系统整合性shell用户接口和历史纪录实践

专业： 班级： 姓名： 学号：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 评审项目 | 分 数 | 满分标准说明 |
| 1 | 内 容 | 30 | 思路清晰；语言表达准确，概念清楚，论点正确；完整详细的程序说明与批注，分析归纳合理；结论严谨，设计有应用价值。任务饱满，做了有质量的工作。 |
| 2 | 创 新 | 10 | 设计内容新颖，编写的程序能反映新技术，对前人工作有改进或突破，或有独特见解。 |
| 3 | 完整性、实用性 | 10 | 整体构思合理，理论依据充分，设计完整，实用性强 |
| 4 | 数据准确、可靠 | 10 | 程序功能完全正确，15项指令说明清楚详细完整。 |
| 5 | 规 范 性 | 10 | 程序编排方式清楚明了，设计格式等符合有关标准和规定 |
| 6 | 纪 律 性 | 10 | 能很好的遵守各项纪律，设计过程认真； |
| 7 | 答 辩 | 20 | 准备工作充分，回答问题有理论依据，基本概念清楚。主要问题回答简明准确。在规定的时间内作完报告。 |
| 总 分 |  | | |
| 综  合  意  见 | 指导教师 年 月 日 | | |