**基于Oranges’ OS 框架的操作系统说明文档**

24组：组长石稼晟

1851632 石稼晟

­­1853115 李家瑞

1751022 李翠琪

1650350 乔 宇

目录

[1.任务分工 4](#_Toc20406)

[2.操作说明 4](#_Toc739)

[2.1输入help指令，跳出command提示表 5](#_Toc3786)

[2.2输入processHelp指令，跳出当前控制器和help指令 5](#_Toc11002)

[2.2.1输入killproc pid可以停止进程，例如killproc 6 6](#_Toc17964)

[2.2.2输入pauseproc pid可以暂停进程，例如pauseproc 5 6](#_Toc22277)

[2.2.3输入resumeproc pid可以恢复进程，例如resumeproc 5 7](#_Toc10270)

[2.3输入fileHelp，显示文件管理部分的帮助文档 7](#_Toc26897)

[2.3.1输入ls，显示当前目录下文件 7](#_Toc3054)

[2.3.2输入cd test，进入test目录 8](#_Toc10981)

[2.3.3回到根目录，输入cd /test/testSon，以绝对路径方式进入 8](#_Toc19786)

[2.3.4输入rm root，移除root目录 8](#_Toc26221)

[2.3.5输入touch me.txt，创建文件me.txt 8](#_Toc24445)

[2.3.6输入vim me.txt，为me.txt添加内容 8](#_Toc5779)

[2.3.7输入cat me.txt，读取me.txt里面的内容 8](#_Toc31595)

[2.3.8输入rm me.txt，删除me.txt 9](#_Toc7110)

[2.3.9输入store，将该文件系统存储在磁盘的myConfig中 9](#_Toc18044)

[2.4输入memoryHelp，显示内存管理部分的帮助命令。 9](#_Toc31159)

[2.4.1输入memoryAlloc，一个模拟内存动态分区分配的程序就会运行 9](#_Toc476)

[2.4.2输入requestPage运行模拟请求调页小程序 10](#_Toc4127)

[2.5 游戏 11](#_Toc3253)

[2.5.1 管理牧场程序 11](#_Toc3715)

[2.5.2 扫雷小游戏 11](#_Toc6486)

[2.5.3 五子棋小游戏 11](#_Toc9046)

[2.5.4 计算器 12](#_Toc13405)

[3.实现详解 13](#_Toc10470)

[3.1控制台实现 13](#_Toc30300)

[3.1.1 实现思路 13](#_Toc26361)

[3.1.2 核心代码 13](#_Toc15257)

[3.2进程管理部分 14](#_Toc30211)

[3.2.1核心函数介绍 14](#_Toc23254)

[3.2.2主要思想： 18](#_Toc31067)

[3.3内存管理部分 18](#_Toc23361)

[3.3.1 动态分区分配： 18](#_Toc20449)

[3.3.1.1核心函数介绍 18](#_Toc27388)

[3.3.2.2主要思想 18](#_Toc5863)

[3.3.3.3核心代码示例： 19](#_Toc8931)

[3.3.2 请求调页程序实现 21](#_Toc31076)

[3.3.2.1核心函数介绍 21](#_Toc18805)

[3.3.2.2 主要思想 21](#_Toc21941)

[3.3.2.3 核心代码示例： 22](#_Toc5643)

[3.4文件系统的实现 26](#_Toc22140)

[3.4.1 实现概述 26](#_Toc16064)

[3.4.2 FCB结构 27](#_Toc16219)

[3.4.3函数说明 27](#_Toc11969)

[3.4.4核心函数代码 28](#_Toc29852)

[3.5 游戏部分 30](#_Toc28628)

[3.5.1扫雷 30](#_Toc15622)

[3.5.1.1函数介绍 30](#_Toc26964)

[3.5.1.2主要思想 35](#_Toc30574)

[3.5.1.3核心代码示例： 35](#_Toc26627)

[3.5.2五子棋 38](#_Toc2505)

[3.5.2.1函数介绍 38](#_Toc15702)

[3.5.1.2主要思想 40](#_Toc15813)

[3.5.1.3核心代码示例： 41](#_Toc31268)

[3.5.3管理牧场 44](#_Toc7988)

[3.5.3.1程序说明 44](#_Toc10216)

[3.5.3.2主要思想 45](#_Toc21203)

[3.5.3.3核心代码示例： 45](#_Toc15601)

[3.6开机画面 48](#_Toc20521)

# 1.任务分工

1851632 石稼晟：进程管理系统，开机画面，目录，应用级程序扫雷，计算器。

­­1853115李家瑞：模拟内存管理：动态分区分配，请求调页算法，PPT，文档制作

1751022李翠琪：文件系统，控制台。

1650350乔宇：应用级程序五子棋，牧场管理小程序

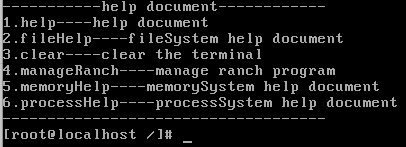
# 2.操作说明

本次项目使用Oranges’操作系统作为基本框架，添加了文件系统，进程调度，内存管理三个个操作系统程序，并自己加入扫雷，五子棋，具体使用步骤如下：

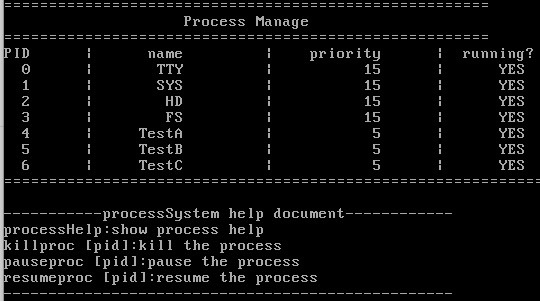
1. 打开ubuntu终端，输入bochs -f bochsrc并输入c进入调试模式。
2. 经过开机动画后，进入初始界面：

进入进程TestA，该进程用于运行所有用户接口

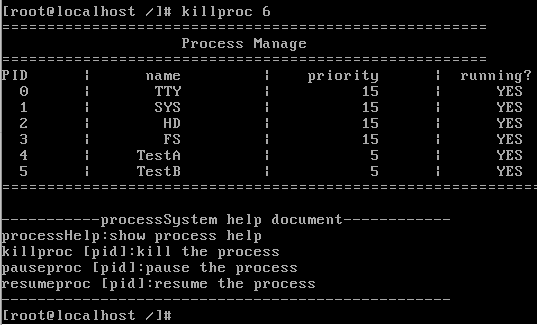
## 2.1输入help指令，跳出command提示表

****

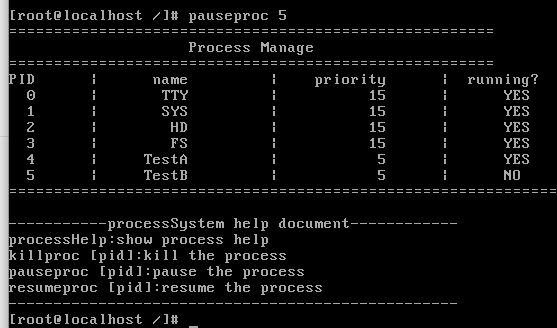
## 2.2输入processHelp指令，跳出当前控制器和help指令

****

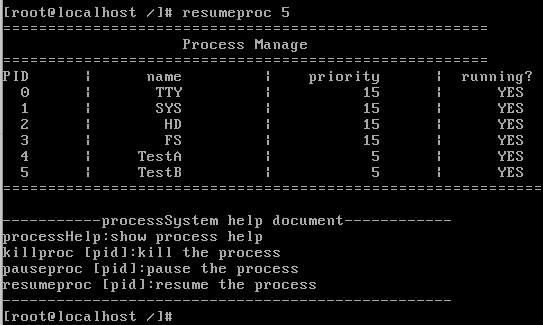
###### 2.2.1输入killproc pid可以停止进程，例如killproc 6

****

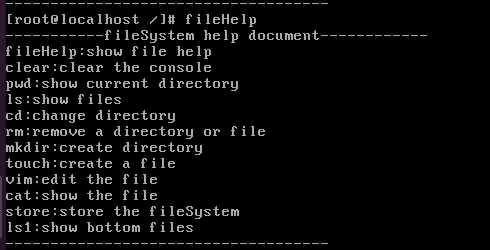
###### 2.2.2输入pauseproc pid可以暂停进程，例如pauseproc 5

****

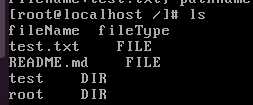
###### 2.2.3输入resumeproc pid可以恢复进程，例如resumeproc 5



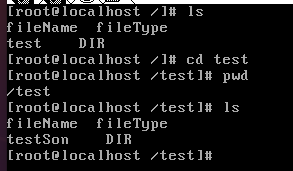
## 2.3输入fileHelp，显示文件管理部分的帮助文档



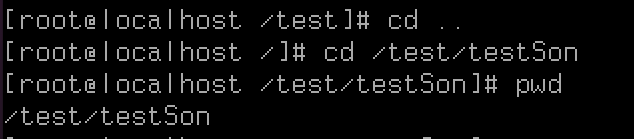
###### 2.3.1输入ls，显示当前目录下文件



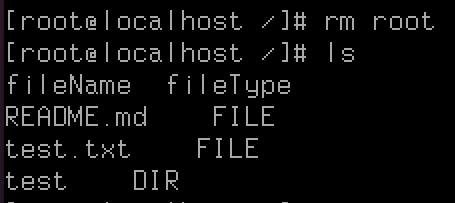
###### 2.3.2输入cd test，进入test目录



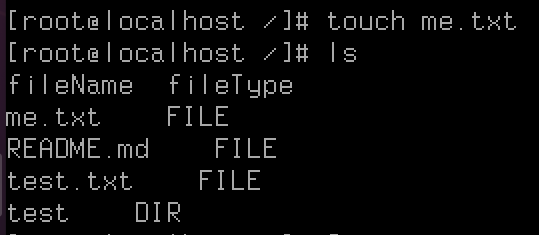
###### 2.3.3回到根目录，输入cd /test/testSon，以绝对路径方式进入



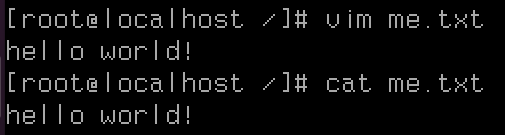
###### 2.3.4输入rm root，移除root目录



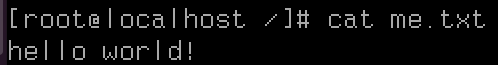
###### 2.3.5输入touch me.txt，创建文件me.txt



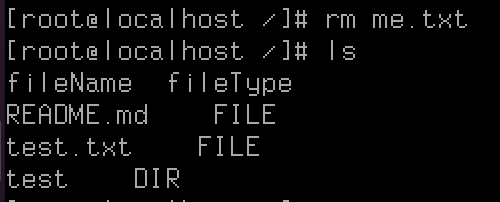
###### 2.3.6输入vim me.txt，为me.txt添加内容



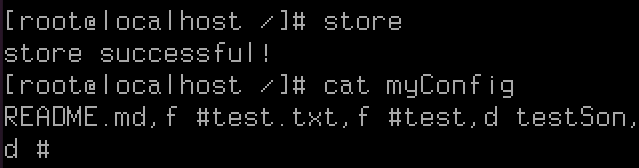
###### 2.3.7输入cat me.txt，读取me.txt里面的内容



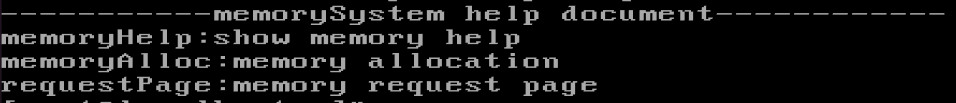
###### 2.3.8输入rm me.txt，删除me.txt



###### 2.3.9输入store，将该文件系统存储在磁盘的myConfig中

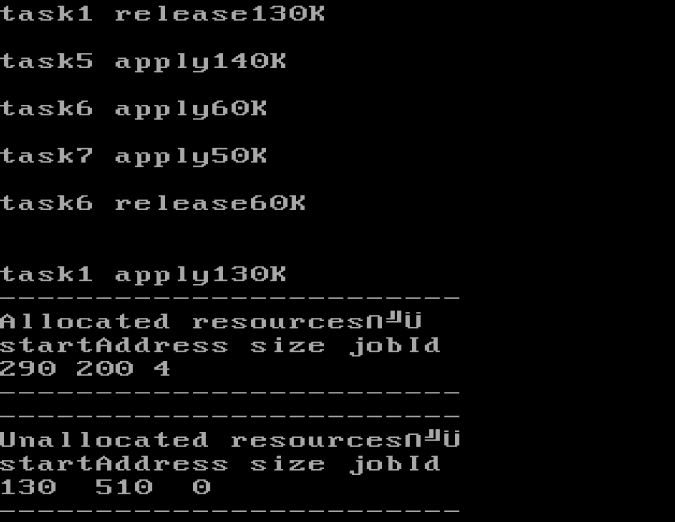


#### 2.4输入memoryHelp，显示内存管理部分的帮助命令。



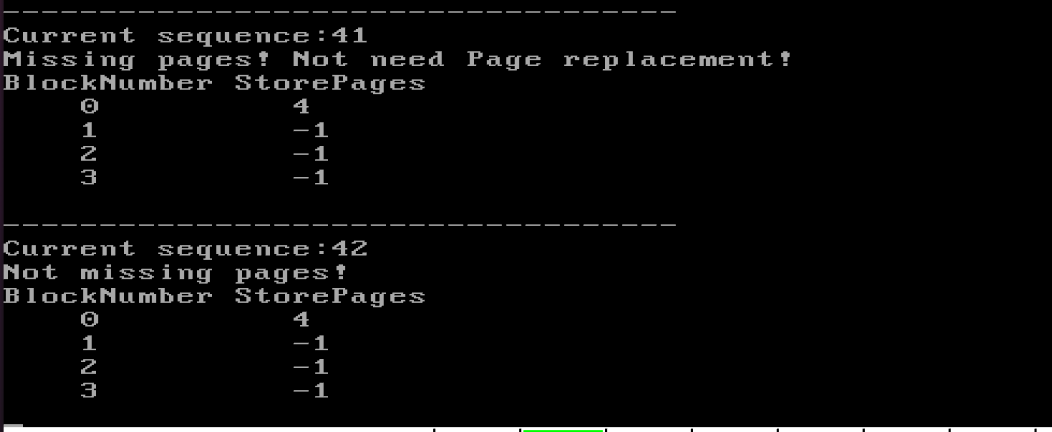
2.4.1输入memoryAlloc，一个模拟内存动态分区分配的程序就会运行

提示用户输入一个数字，1表示首次适应算法，2表示最佳适应算法



（首次适应算法）

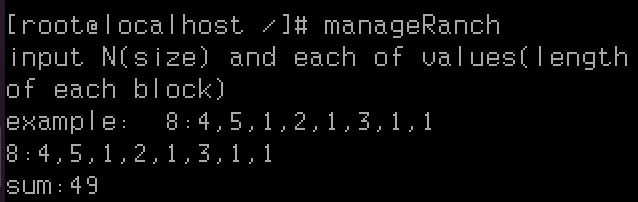
###### 2.4.2输入requestPage运行模拟请求调页小程序



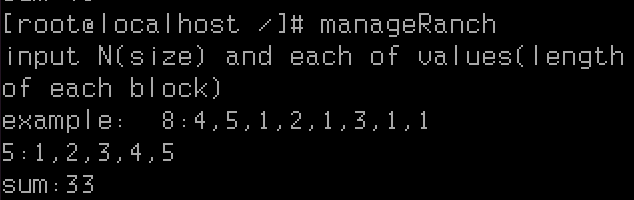
## 2.5 游戏

###### 2.5.1 管理牧场程序

输入指令序列为: 8:4,5,1,2,1,3,1,1

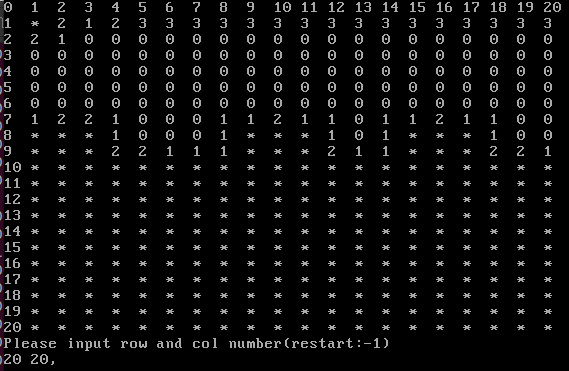


输入指令序列为: 5:1,2,3,4,5



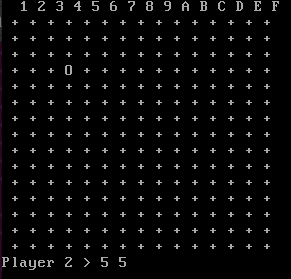
###### 2.5.2 扫雷小游戏

输入坐标（格式为横坐标，空格，纵坐标，逗号）



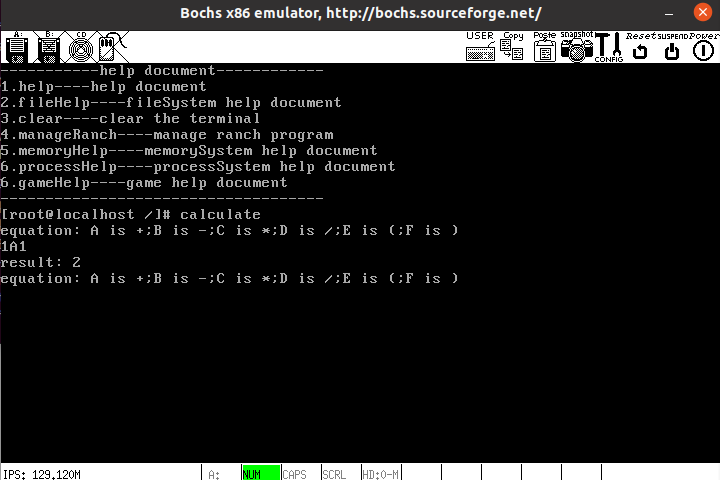
###### 2.5.3 五子棋小游戏

输入坐标（格式为横坐标，空格，纵坐标，逗号）



###### 2.5.4 计算器

输入算式，不用加“=”，输入回车，即可获得答案。



# 3.实现详解

## 3.1控制台实现

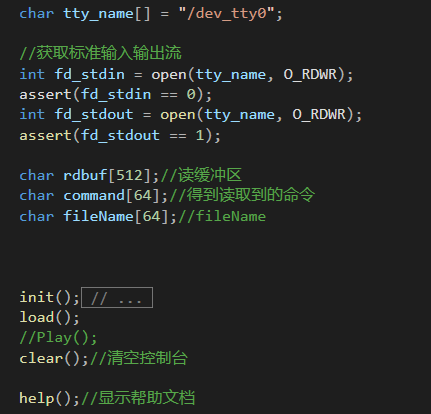
###### 3.1.1 实现思路

首先获取标准输入输出流。然后对之前的代码进行一些初始化，例如加载文件系统，显示帮助文档等等。

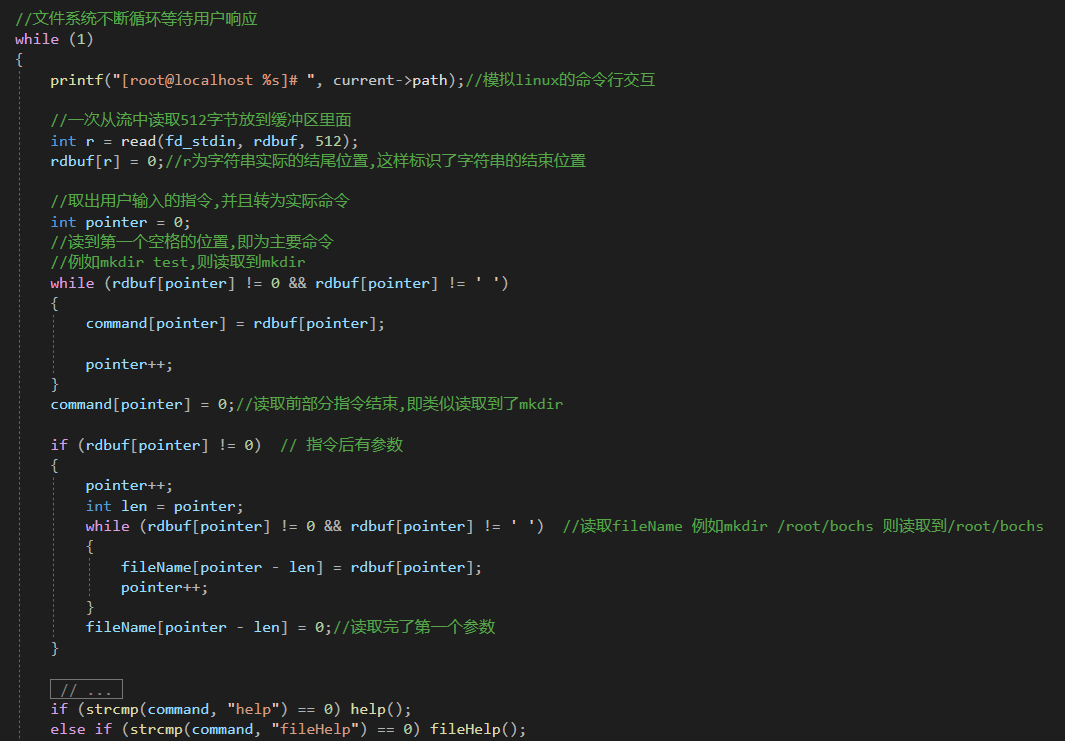
随后进入循环。首先打印当前所在目录，获取用户输入的指令，并且将用户指令进行处理，得到指令内容和参数（例如mkdir test,则拆分为指令mkdir和参数test）。然后根据用户输入的指令的不同，调用不同的函数实现功能。

###### 3.1.2 核心代码

###### 初始操作：

****

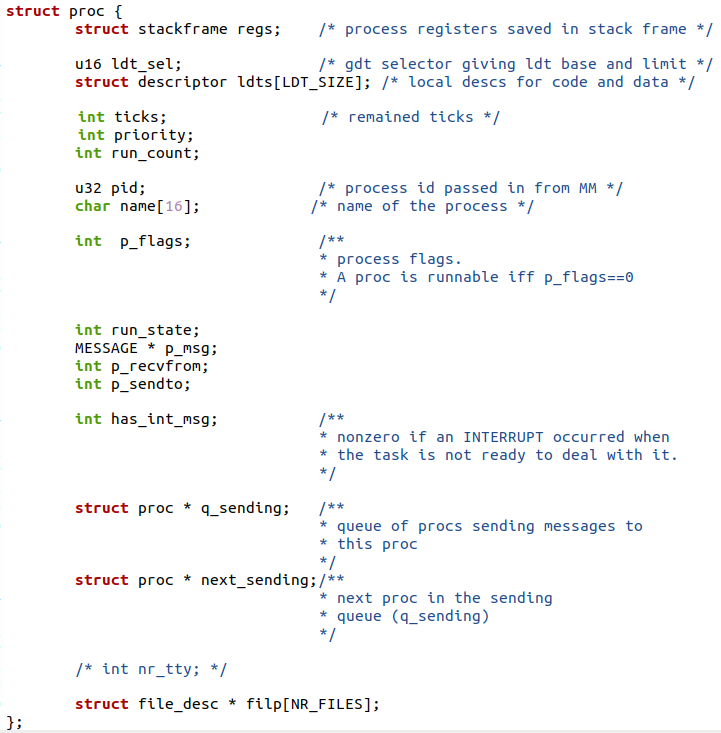
**与用户交互的循环：**

****

## 3.2进程管理部分

###### 3.2.1核心函数介绍

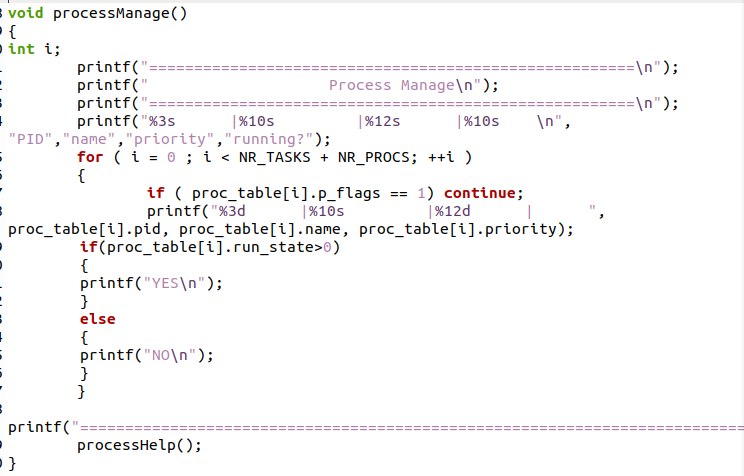
在proc.h文件中，设置process进程的变量。

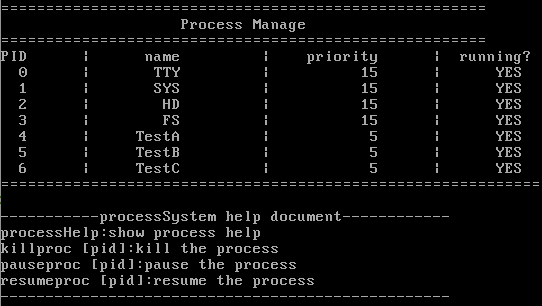


###### 3.2.1.1：

其中需要用户获取的内容为pid（process ID），name（进程的名字），priority（优先级）p\_flags（创建后是否完成进程），run\_state（运行状态）。

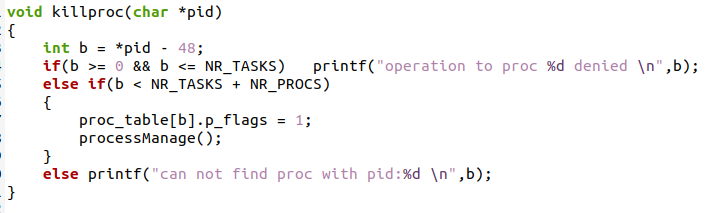
首先需要设置进程管理器：





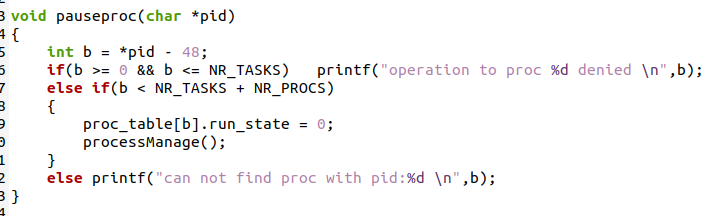
###### 3.2.1.2

进程管理器展现出不同的进程（包括系统进程和用户进程）的编号，名字，优先级，运行状态。其中，运行状态为通过run\_state表示，其中，每个进程当被创建的时候，run\_state默认值为1，如果调用pauseproc函数则会将run\_state变为0.在进程控制器中，如果run\_state为0，则表示为没有running，如果run\_state为1，则表示为running。其中标号ID通过数字表示，从0开始，名字为字符串，优先级通过数字表示，系统进程的优先级更高，用户进程默认比系统进程低。进程若能展现，则表示p\_flags为1，如果为0则表示运行完成。



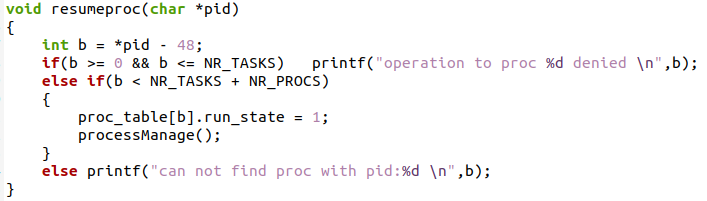
3.2.1.3 kill process

通过控制命令加上进程编号，代表为终止进程，即通过p\_flags变为0，也就是说进程结束，在进程管理器中就因为p\_flags为0而不能表示出来。



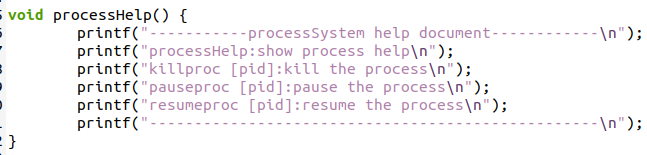
3.2.1.4 pause process

通过控制命令加上进程编号，代表着暂停进程，通过改变run\_state变为0，暂停进程，同时在进程管理器中显示NO。



###### 3.2.1.5 resume process

通过控制命令加上进程编号，代表着恢复进程，通过改变run\_state变为1，恢复进程，仍在running的进程默认为恢复进程，即不改变，同时在进程管理器中显示YES。



###### 3.2.1.6 目录

分别展现process system目录和其功能函数。

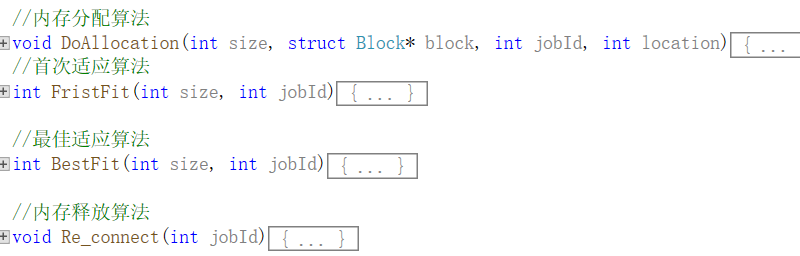
## 3.2.2主要思想：

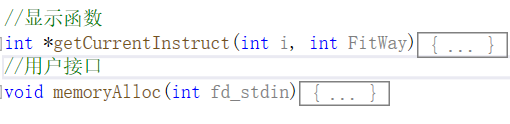
进程管理通过停止进程，暂停进程，恢复进程等用户级操作完成进程的管理，将以上操作封装到管理当中。

## 3.3内存管理部分

###### 3.3.1 动态分区分配：

###### 3.3.1.1核心函数介绍

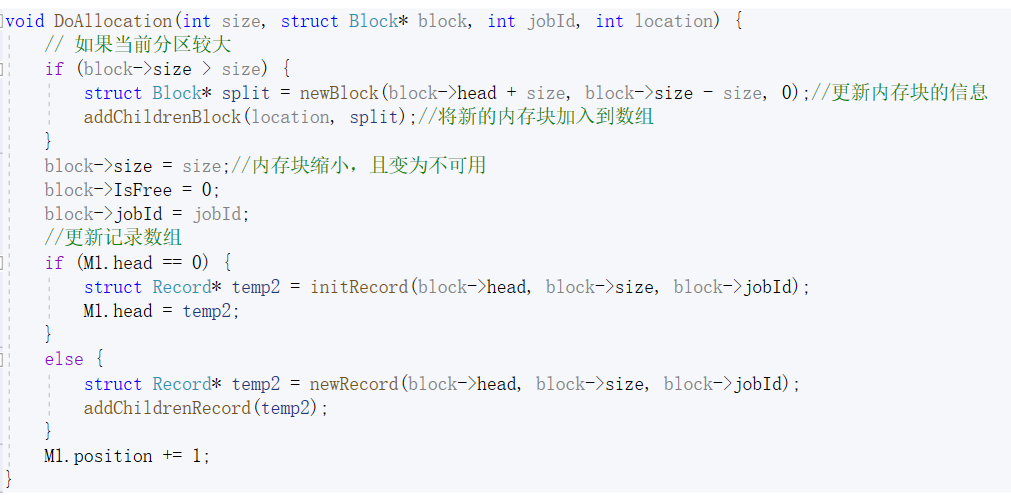




###### 3.3.2.2主要思想

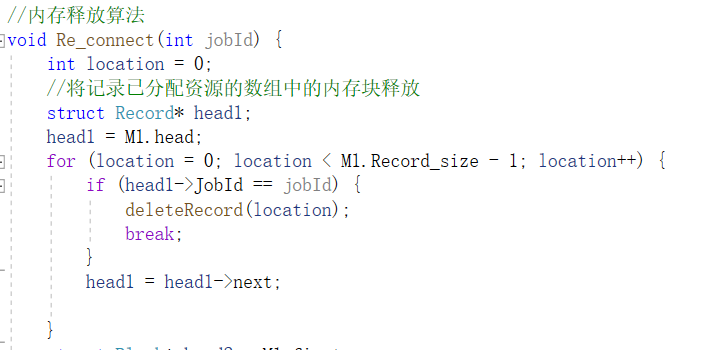
* 在模拟过程中，依次执行11条指令，每执行一条指令会显示当前已分配的内存空间和未分配的内存空间，并转到下一条指令；
* 动态分区分配算法使用首次适应算法和最佳适应算法
* 具体实施方法(参考):
* 执行第一条指令，根据不同算法选择合适内存空间装入作业，将装入空间的部分设置为不可用并标记作业号，碎片加入到可使用的内存空间部分
* 顺序执行下一条指令，即序号为m+1的指令
* 释放内存空间时，首先将作业对应的内存空间设为可用，然后判断其头指针或尾指针是否有空闲区域，有则合并，无则正常加入对应链表数组。
* 直到执行完最后一条指令，模拟程序结束。

###### 3.3.3.3核心代码示例：





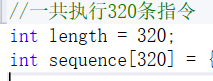






###### 3.3.2 请求调页程序实现

###### 3.3.2.1核心函数介绍

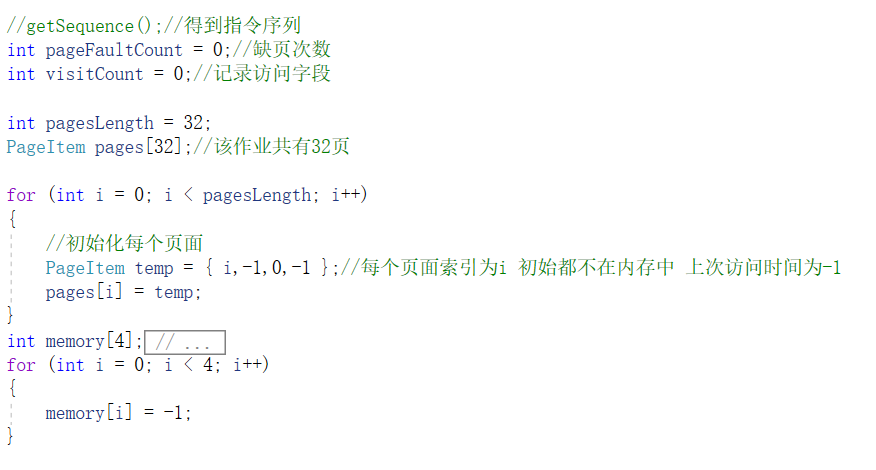




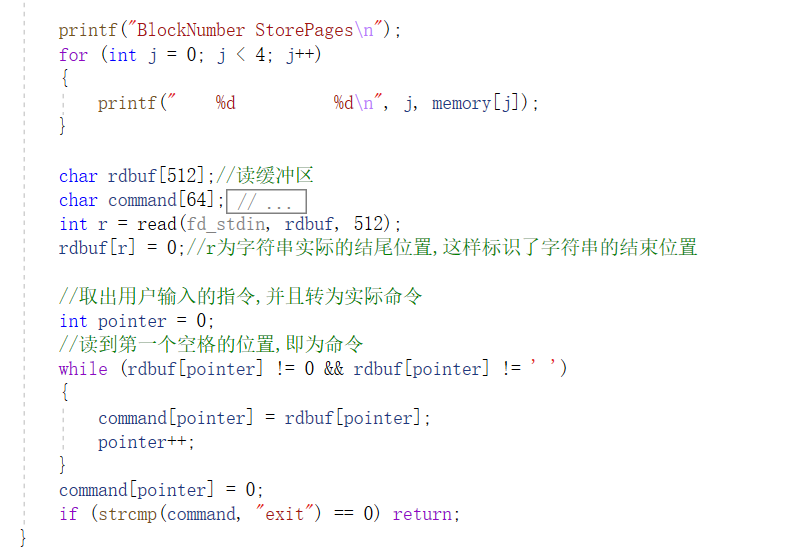
###### 3.3.2.2 主要思想

* 在模拟过程中，如果所访问指令在内存中，则显示其物理地址，并转到下一条指令；如果没有在内存中，则发生缺页，此时需要记录缺页次数，并将其调入内存。如果4个内存块中已装入作业，则需进行页面置换。
* 所有320条指令执行完成后，计算并显示作业执行过程中发生的缺页率。
* 置换算法可以选用FIFO或者LRU算法
* 作业中指令访问次序可以按照下面原则形成:50%的指令是顺序执行的，25%是均匀分布在前地址部分，25％是均匀分布在后地址部分
* 具体实施方法(参考):
* 在0－319条指令之间，随机选取一个起始执行指令，如序号为m
* 顺序执行下一条指令，即序号为m+1的指令
* 通过随机数，跳转到前地址部分0－m-1中的某个指令处，其序号为m1
* 顺序执行下一条指令，即序号为m1+1的指令
* 通过随机数，跳转到后地址部分m1+2~319中的某条指令处，其序号为m2
* 顺序执行下一条指令，即m2+1处的指令。
* 重复跳转到前地址部分、顺序执行、跳转到后地址部分、顺序执行的过程，直到执行完320条指令。

###### 3.3.2.3 核心代码示例：







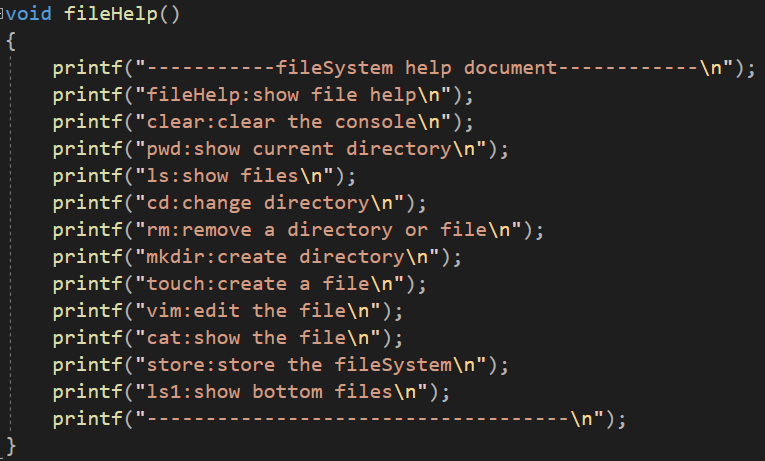
###### 3.4文件系统的实现

###### 3.4.1 实现概述

使用FCB对文件系统进行管理,总共分为文件夹和文件两种类型。目录树存储在内存中，通过调用系统的open(),write()等函数进行实际磁盘的读取。在退出操作系统时，用户可以调用指令将当前文件系统目录树内容序列化存储于硬盘中,进行持久化操作，下次再进入操作系统就会自动读取之前的目录树信息。

对文件夹可进行操作有：显示当前目录，列出当前文件夹下所有文件，进入目录，创建目录，删除目录。对文件则可进行增删改查操作。

主要功能说明：



3.4.2 FCB结构

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **作用** |
| **fileName** | char[] | 文件名称 |
| **path** | char[] | 文件绝对路径名称 |
| **fileType** | int | 标识文件类型（目录或文件） |
| **size** | int | 存储该目录的子文件数量（如果是文件则为0） |
| **next** | FCB\* | 为寻找children而设置的，指向同级的下一个文件的FCB |
| **parent** | FCB\* | 指向他上级文件夹对应的FCB |
| **children** | FCB\* | 指向他拥有的第一个子文件FCB节点 |

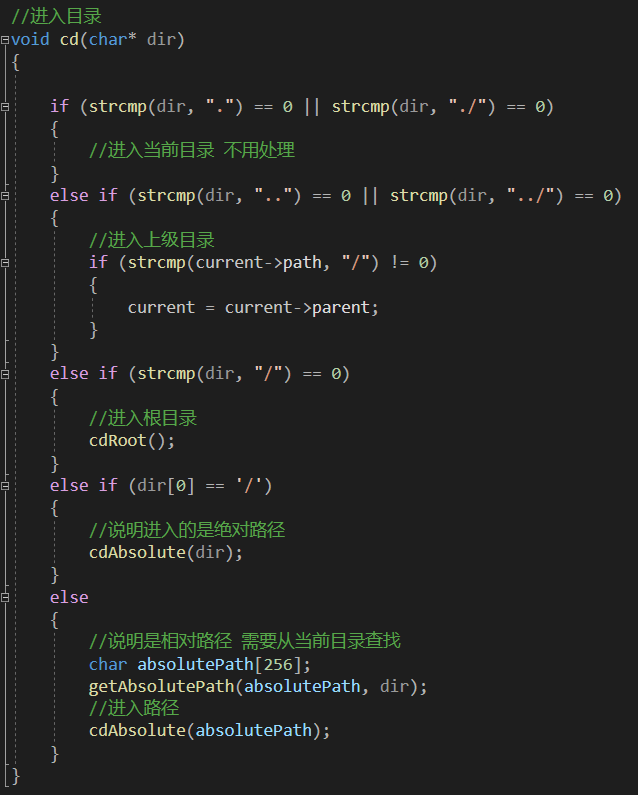
3.4.3函数说明

很多对文件系统的操作都基于当前目录：**FCB \* current; //当前路径的FCB**

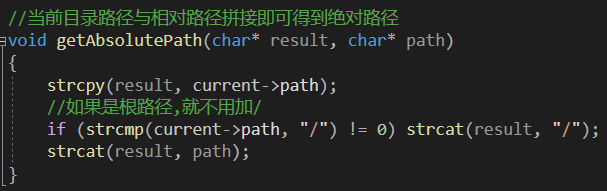
|  |  |
| --- | --- |
| **名称** | **作用** |
| **init()** | **将当前目录初始化为根目录** |
| **addChildren(FCB\* parent, FCB\* son)** | **在某个目录下添加FCB子节点** |
| **showChildren(FCB\* parent)** | **展示当前目录下的所有文件信息** |
| **getAbsolutePath(char\* result, char\* path)** | **将当前目录路径与相对路径拼接得到的绝对路径** |
| **cd(char\* dir)** | **进入某个目录，可以有绝对路径，相对路径两种写法** |
| **cdRoot()** | **进入根目录** |
| **cdAbsolute(char\* absolutePath)** | **辅助cd()的函数,循环更改current,使之进入某绝对路径** |
| **pwd()** | **展示当前路径** |
| **list()** | **列出当前目录下文件** |
| **createFile(char\* dir, int type)** | **创建目录或者文件 根据type标识** |
| **rm(char\* fileName)** | **删除文件** |
| **writeFile(char\* fileName)** | **将内容写入文件** |
| **readFile(char\* fileName)** | **读取文件内容** |
| **store()** | **将目录树对应的FCB文件以特定形式序列化存储在磁盘中** |
| **serialize(char\* files, FCB\* father, int i, int height)** | **辅助store()函数，用于将目录树序列化为一个特定类型字符串** |
| **load()** | **系统启动时运行, 将磁盘中数据以约定的形式取出,重构目录树** |
| **fileHelp()** | **文件系统的帮助文档，展示文件系统能执行的指令** |

3.4.4核心函数代码

**进入目录函数**：根据用户输入的指令的不同，执行不同的进入目录操作。如：.代表当前路径，../代表上一级路径，以/开头为进入绝对路径，否则为进入相对路径。



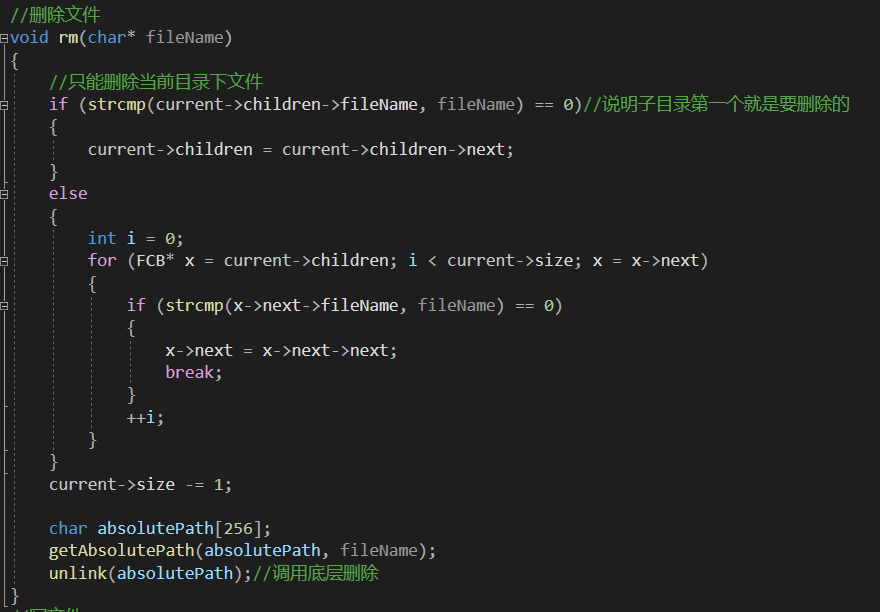
**获取绝对路径函数：**

****

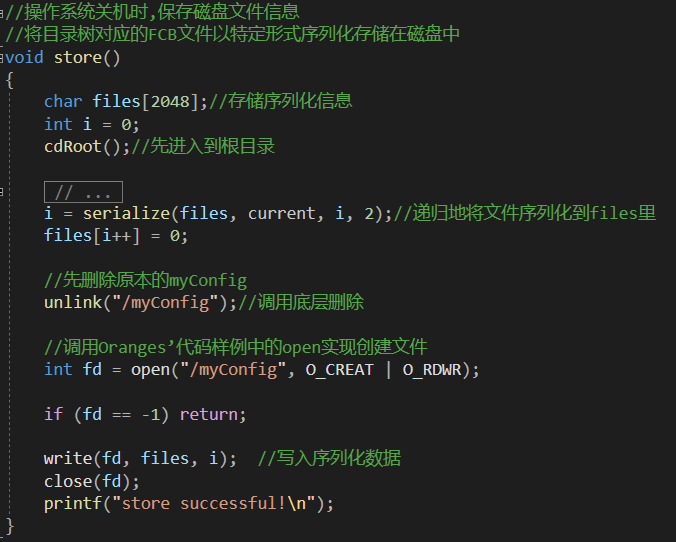
**展示当前目录下文件信息：**

****

**删除文件：**

****

**文件持久化操作：**

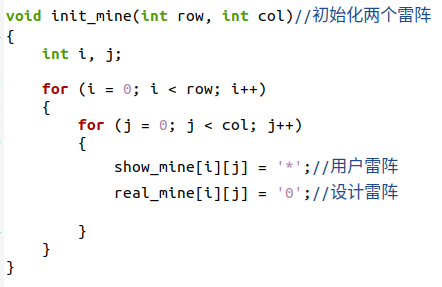
****

## 3.5 游戏部分

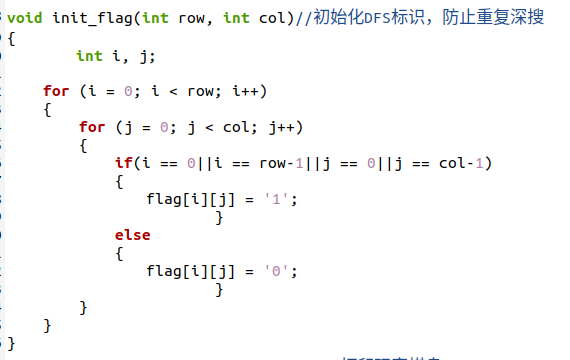
###### 3.5.1扫雷

###### 3.5.1.1函数介绍

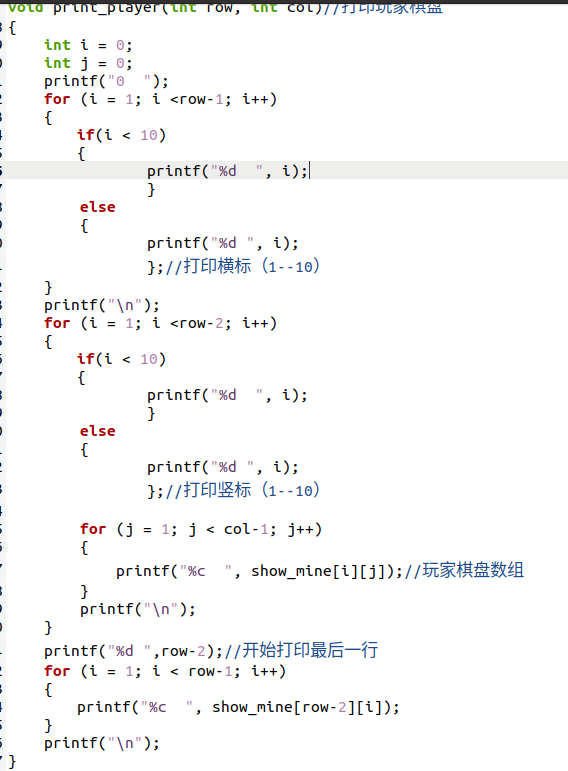
void init\_mine(int row, int col)初始化两个扫雷的矩阵，一个为char类型，为在命令行展示的内容，默认\*，表示为未点击，数字代表着周围8个点共有几个地雷。同理，int类型，如果为0，表示为未点击。



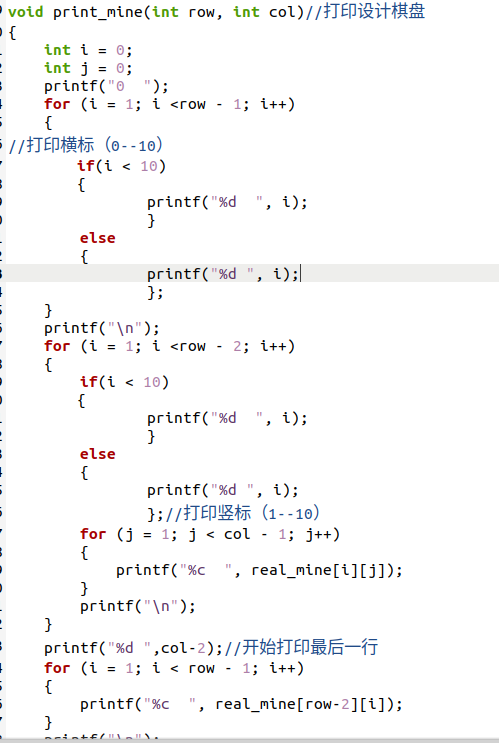
void init\_flag(int row, int col)初始化DFS标识，防止重复深搜



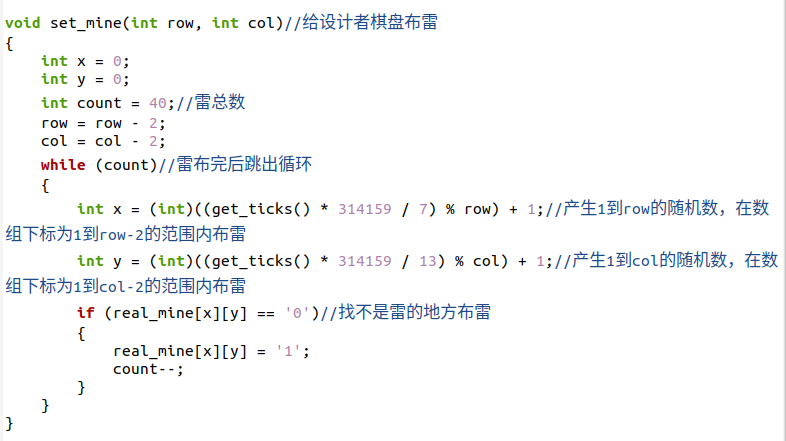
void print\_player(int row, int col)打印棋盘矩阵，首先打印横标，然后随着打印竖标，若没有点击，则为\*，若点击且不为雷，则会有数字。



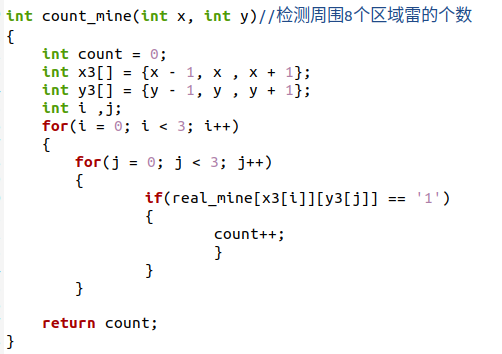
void print\_mine(int row, int col)打印设计棋盘，方便检查。



void set\_mine(int row, int col)给设计者棋盘布雷，设置的总雷数为40个，共20\*20的棋盘，通过get\_ticks()产生随机数，然后布雷，通过调用count\_mine函数，将其他没有雷的地方变为数字，每个数字的产生根据周围八个点的地雷数完成。



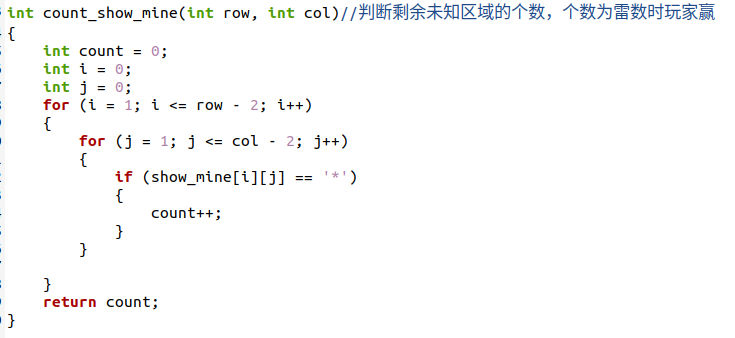
int count\_mine(int x, int y)检测周围8个区域雷的个数，设置八个方向，通过3\*3横纵坐标决定，每个方向上分别有x，x-1，x+1；y，y-1，y+1。



void open\_mine\_dfs(int x, int y)空白区炸开函数 DFS。通过深度搜索，完成搜索所有的空白，也就是说该点的周围八个点没有地雷。



int count\_show\_mine(int row, int col)判断剩余未知区域的个数，个数为雷数时玩家赢



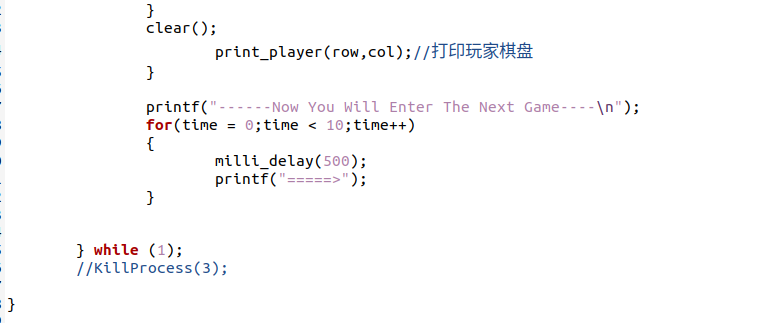
###### 3.5.1.2主要思想

首先调用主要函数startSwipe(int fd\_stdin)，首先设置地雷的矩阵地图初始化，然后通过输入x，y反复输入函数点击的位置，若为空白区，空白区炸开函数 DFS ，如果点击地雷，则游戏失败，游戏结束，若满足剩余未点击的点数为设置雷数则表示为剩下的40个都为地雷，则表示完美避开所有地雷，游戏胜利，游戏结束。

###### 3.5.1.3核心代码示例：



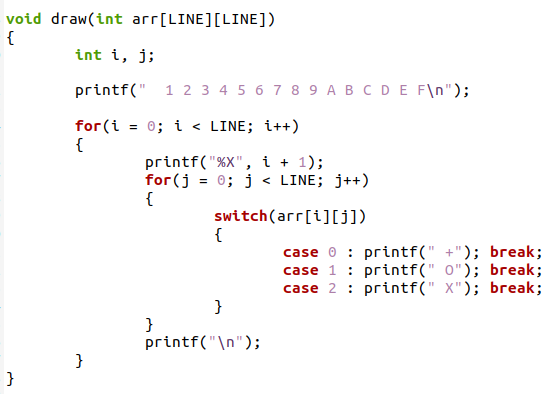




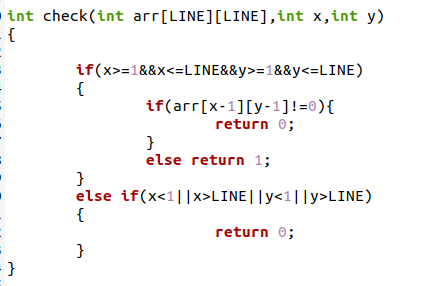
###### 3.5.2五子棋

###### 3.5.2.1函数介绍

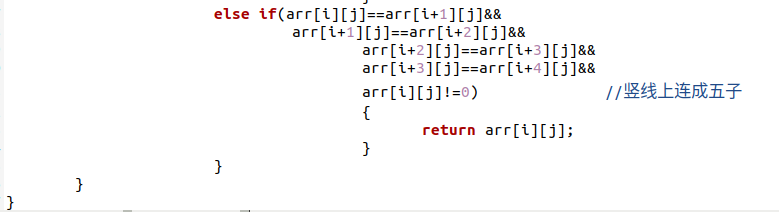
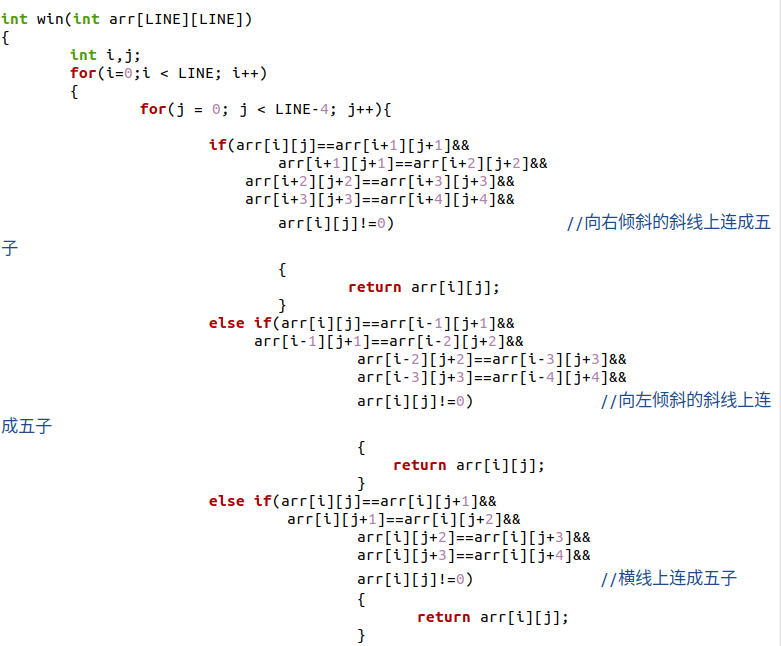
void draw(int arr[LINE][LINE])绘制五子棋面板。设置了横坐标



int check(int arr[LINE][LINE],int x,int y)检查五子棋面板坐标，如果已被修改则返回0，如果未被修改，则返回1。



int win(int arr[LINE][LINE])判断是否为赢。判断条件为横着五个点，竖着五个点，斜着五个点都为同一个玩家所下，返回该五个点的共同所属。



int startFive(int fd\_stdin)主函数

###### 3.5.1.2主要思想

通过调用int startFive(int fd\_stdin)函数，首先每回合打印当前棋盘，分为两个玩家，玩家1先手，为O，玩家2为后手，为X通过输入坐标完成，当满足评判标准，则游戏结束。

###### 3.5.1.3核心代码示例：





###### 3.5.3管理牧场

###### 3.5.3.1程序说明

请人锯木的酬金跟这段木头的长度成正比。例如，要将长度为20的木头锯成长度为8，7和5的三段，第一次锯木头将木头锯成12和8，花费20；第二次锯木头将长度为12的木头锯成7和5花费12，总花费32元。如果第一次将木头锯成15和5，则第二次将木头锯成7和8，那么总的花费是35（大于32）。

输入格式：输入第一行给出正整数N（N<100），表示要将木头锯成N块。第二行给出N个正整数，表示每块木头的长度。

输出格式：输出一个整数，即将木头锯成N块的最小花费。

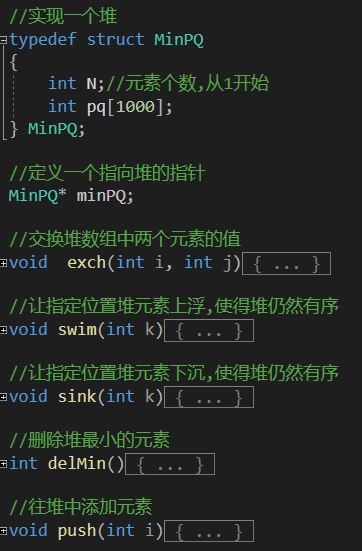
###### 3.5.3.2主要思想

直观上来说，就是在一堆木块碎片中，每次都把最小的两块碎片拼成一块，不断地拼，直到最后就只剩下最初的完整木块，这样的花费最小。

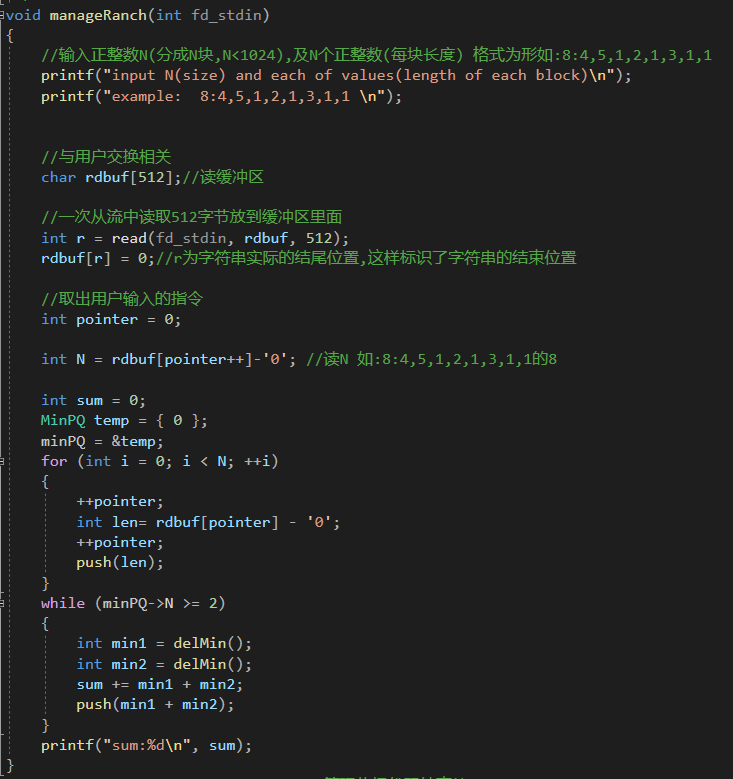
首先实现一个最小堆。把输入的N个正整数逐个放入最小堆中，然后不断循环直到堆只剩下一个元素即为结果。在循环中，每次都从堆中取出最小的两个元素，让总的花费sum加等于他们两个的值。并且把他们两个加起来的值继续放入堆中。

###### 3.5.3.3核心代码示例：

**堆实现：**

****

**管理牧场主函数：**

****

**3.5.4计算器**

**3.5.4.1函数介绍:**

通过输入算式表示，输入回车完成计算，给出计算结果。

由于操作系统oranges本身的问题，所以将加减乘除、左括号、右括号、分别换成ABCDEF共6个大写字母，来表示。



int startCalculate(int fd\_stdin)该函数为计算器封装函数，可以直接调用函数进行计算。

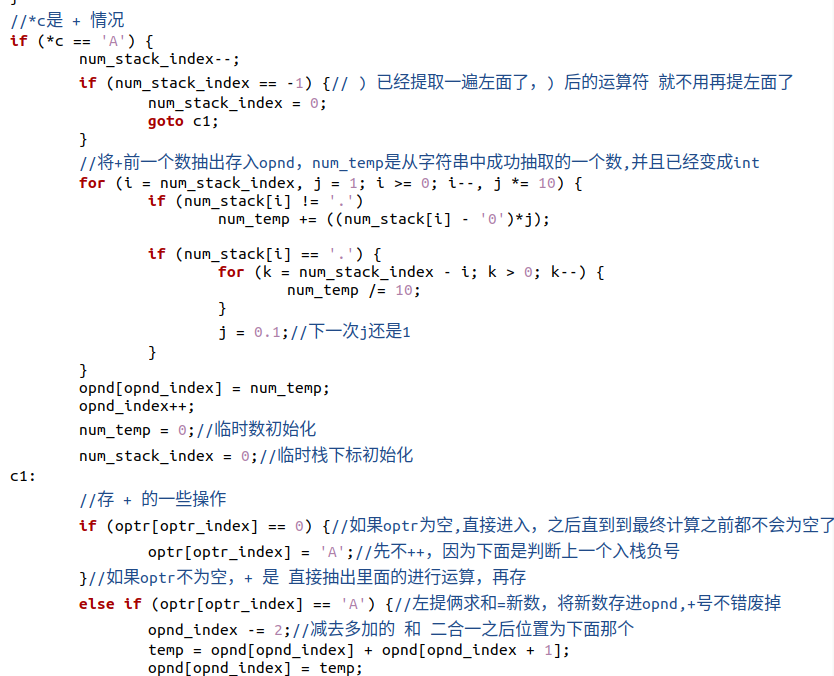
int calculate(int lenth, char \*c)为主计算函数，通过输入算式和算式的长度，来完成计算出结果。

原理是用了三个栈,指针不断分析字符串里每个字符,不断存进三个栈中,一个存数字的个位十位等每一位，一个存运算符，一个存数字来实现。

**3.5.4.2主要思想:**

主要思想为，首先将输入的算式以字符串形式存储，然后通过识别，识别出所有的数字以及符号，当识别数字的时候，从第一位数字识别起，分别识别下一位，如果下一位为符号，则数字识别完毕，否则将之前的数字计算乘10再加上下一位，依次反复，直到出现符号位，符号共有6个符号，分别为加减乘除、左括号、右括号。左右括号匹配。如果识别出的是数字则压入数字栈，如果是符号则压入符号栈，前提是所有的输入必须符合标准，否则系统报错。不同符号的优先级不同，所以会优先处理乘号和除号，再处理加号和减号，如果有括号则优先处理括号里的数。不断生成的数也会压入数字栈。

**3.5.4.3主要代码:（仅展示加法部分代码）**



###### 3.6开机画面

开机画面通过clear（）函数和静态页面时间差，完成动态开机画面的实现。具体表现为鸽子的移动。使用逐帧printf的方式实现，通过设置较小的延迟时间形成的动画效果。

****

