

《操作系统》lab2

|  |  |
| --- | --- |
| 学 院： | 计算机与信息技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科1602 |
| 姓 名： | 麻锦涛 |
| 学 号： | 16281262 |

**日期：2019.4.13**

1. **实验目的**

（1）充分理解多线程

（2）练习创建线程并协调线程的运行

1. **实验要求**

（1）第一部分 - 数独解决方案验证器

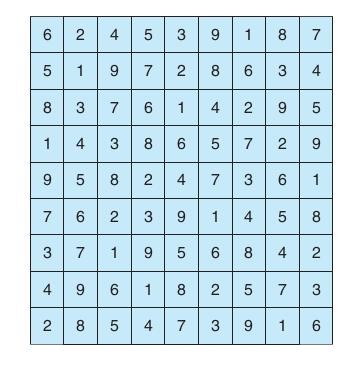
数独谜题使用9×9网格，其中每个列和行以及九个3×3子网格中的每一个必须包含所有数字1···9。下图显示了有效数独的数据示例难题。该项目包括设计一个多线程应用程序，用于确定Sudoku难题的解决方案是否有效。

这种应用程序有多种不同的多线程方式。一个建议的策略是创建检查以下标准的线程：

检查每列是否包含数字1到9的线程

检查每行是否包含数字1到9的线程

九个线程检查3×3子网格中的每一个是否包含数字1到9



这将导致总共11个单独的线程用于验证数独谜题。但是，欢迎您为此项目创建更多线程。例如，您可以创建九个单独的线程并让每个线程检查一列，而不是创建一个检查所有九列的线程。

将参数传递给每个线程

父线程将创建工作线程，为每个工作者传递它必须在Sudoku网格中检查的位置。此步骤需要将几个参数传递给每个线程。最简单的方法是使用struct创建数据结构。例如，传递线程必须开始验证的行和列的结构将如下所示：

/ \*将数据传递给线程的结构\* /

typedef结构

{int row; int列;

参数;

Pthreads和Windows程序都将使用类似于下面所示的策略创建工作线程：

参数\* data =（参数\*）malloc（sizeof（参数））; data-> row = 1; data-> column = 1;

/ \*现在创建传递数据的线程作为参数\* /

数据指针将传递给pthread\_create（）（Pthreads）函数或CreateThread（）（Windows）函数，后者又将其作为参数传递给作为单独线程运行的函数。

将结果返回到父线程

为每个工作者线程分配确定数独谜题的特定区域的有效性的任务。一旦工作人员执行了此检查，它必须将其结果传递回父级。处理此问题的一种好方法是创建一个对每个线程都可见的整数值数组。此数组中的第i个索引对应于第i个工作线程。如果工作人员将其对应值设置为1，则表示其数独谜题的区域有效。值0表示不是这样。当所有工作线程都已完成时，父线程会检查结果数组中的每个条目，以确定数独谜题是否有效。

（2）第II部分 - 多线程排序应用程序

编写一个多线程排序程序，其工作方式如下：整数列表分为两个相同大小的较小列表。两个单独的线程（我们将术语排序线程）使用您选择的排序算法对每个子列表进行排序。然后，这两个子列表由第三个线程合并 - 一个合并线程 - 它将两个子列表合并为一个排序列表。

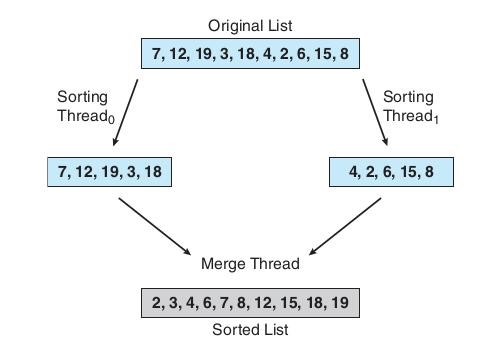


图2.多线程排序。

由于全局数据是跨所有线程共享的，因此设置数据的最简单方法可能是创建全局数组。每个排序线程都可以在这个数组的一半上运行。还将建立与未排序的整数数组大小相同的第二个全局数组。

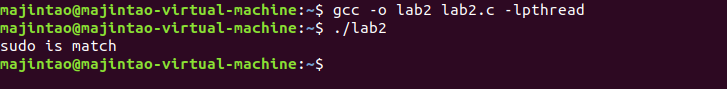
然后，合并线程将两个子列表合并到第二个数组中。在图形上，该程序根据图2构造。该编程项目将需要将参数传递给每个排序线程。特别是，有必要确定每个线程开始排序的起始索引。有关将参数传递给线程的详细信息，请参阅项目1中的说明。

所有排序线程退出后，父线程将输出已排序的数组。

1. **运行截图（代码见于第6部分附录）**
2. 第一部分 - 数独解决方案验证器

输入“gcc –o lab2 lab2.c -lpthread”

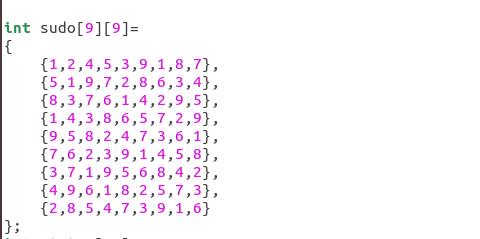
然后再输入“./lab2”得到结果“sudo is match”说明数独谜题有效



向终端输入“gedit lab2.c”打开lab2.c进行修改

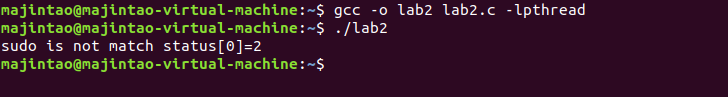
C:\Users\dell\AppData\Local\Temp\1555168135(1).png

将最左上角的6改为1，使之不再满足数独谜题条件

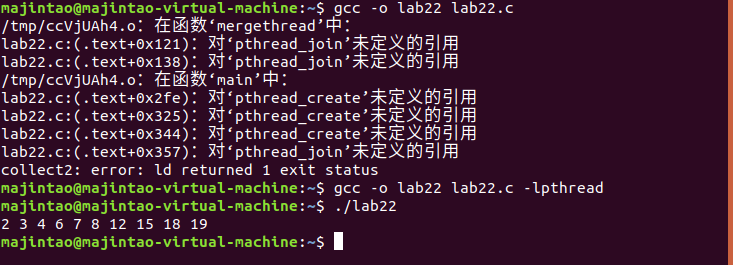
****

输入“gcc –o lab2 lab2.c -lpthread”

然后再输入“./lab2”得到结果“sudo is not match status[0]=2”，这里的status[]里面是0表示还没有退出，若status[0]=2则为不满足条件

****

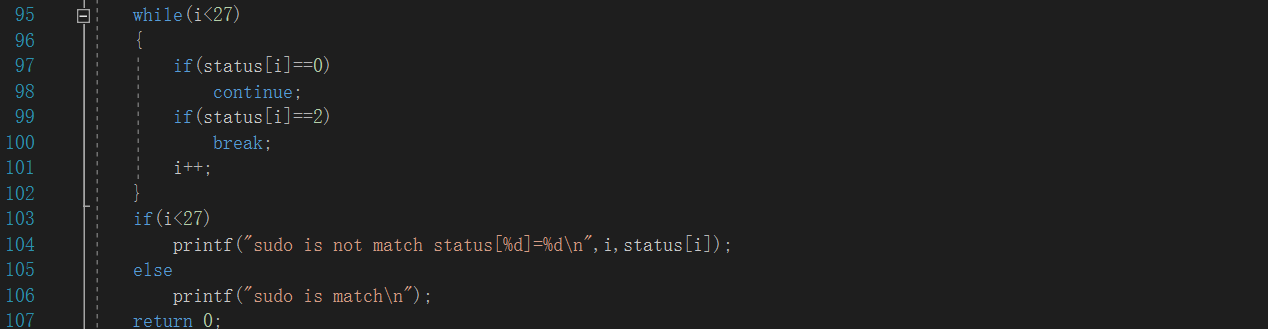
（2）第II部分 - 多线程排序应用程序

****

1. **部分代码部分分析**

（1）实验中用到的线程控制方法：

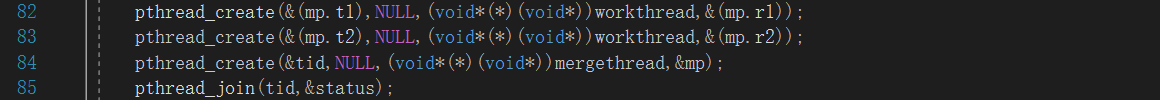
数独解决方案验证器使用的是全局的数组标记来实现线程控制（线程退出）



多线程排序这里使用的是pthread和join实现线程控制

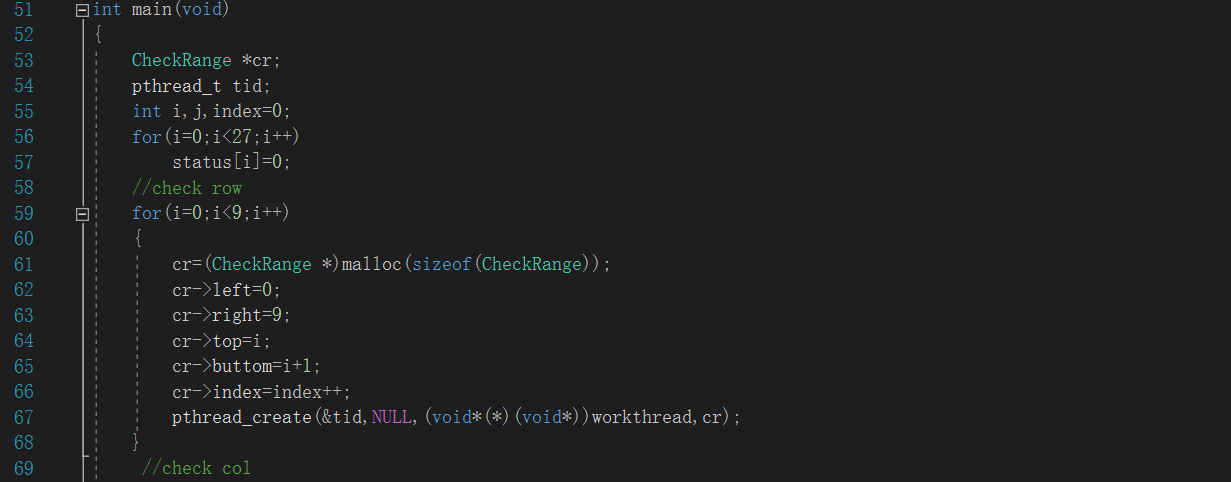


create是创建，join是等待



（2）数独解决方案验证器程序设计思路

对main函数每行每列创建线程，再通过while是检查那些线程是不是全部退出了

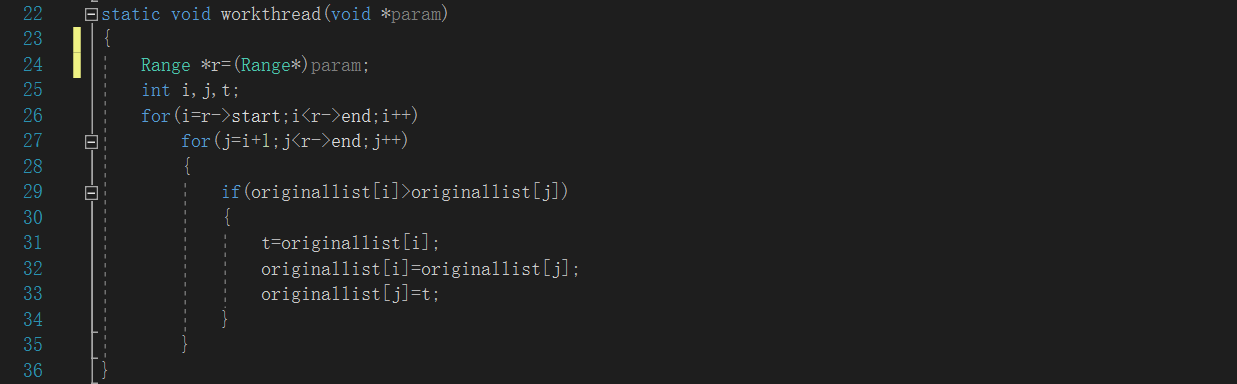


Status后面是0表示还没退出，后面的2表示并不满足条件

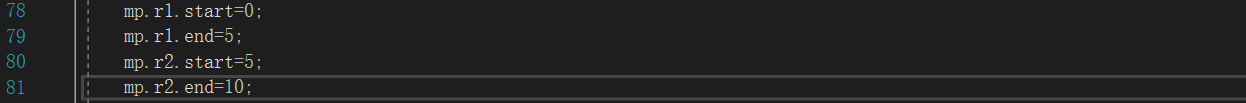


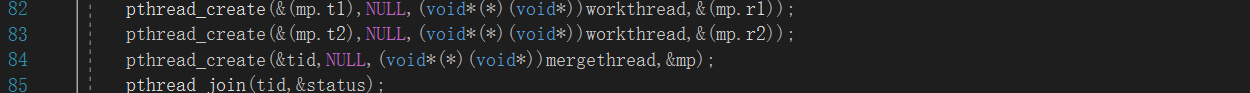
（3）多线程排序部分程序设计思路

这部分对传入的指定空间进行排序

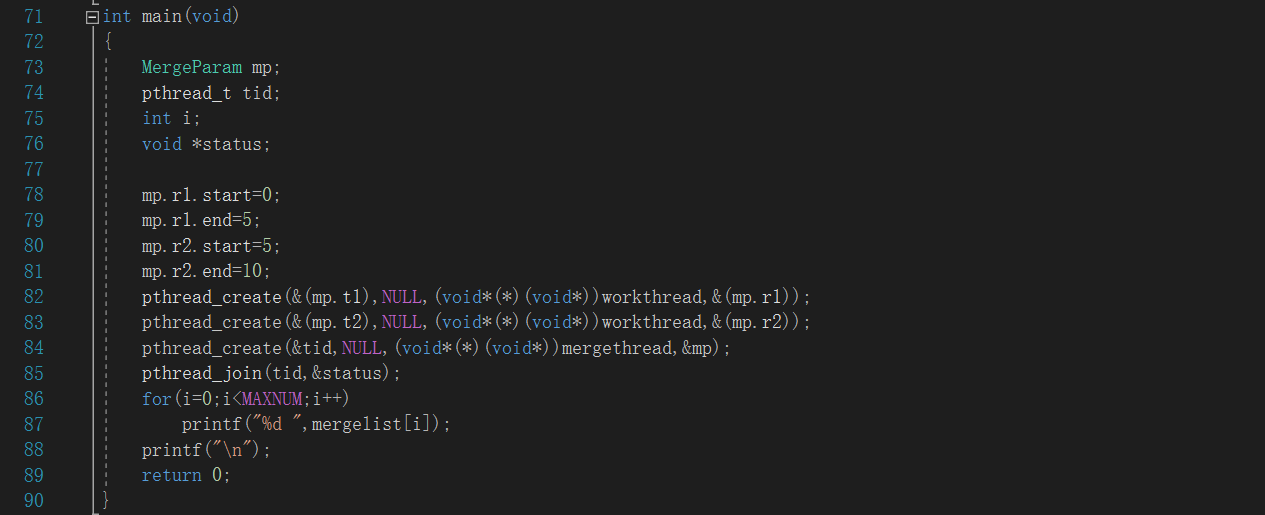


对main函数进行分析：

这里把数组分成了两部分



前2个pthread\_create就是创建好的分别排序的进程。第3个线程去join前2个线程，完成了最后的合并。最后等待第3个线程都退出了，就是排序好了。这个时候打印出结果



1. **实验总结与收获**

（1）前面有几节课迟到了所以导致听课质量不高，做实验课也发现很吃力。自己的英语阅读能力本身不是特别好，对照着百度翻译才能完整看完整个实验要求。在写程序的过程中我发现大一学过的指针部分知识都忘的差不多了，于是重新捡起来看了一遍，发现自己还是太菜了呜呜呜…于是我找到了一位已经工作多年的linux大神，经过一番软磨硬泡，在他的帮助和指导建议之下终于在周末搞完了lab2。特别是第一个数独，这里可能和一般人不同，直接定义了一个结构体表示区域（1\*9 9\*1 3\*3 ）来处理。我这里是一个函数27个线程，有的人可能会用三个函数11个线程的方式，按照我这里的方式，这样代码复用可能会更高。

（2）Linux一些命令我还是不太熟悉，得对着网上的敲。这玩意儿有点难记。比如之前总是忘了在后面加-lpthread,导致出现了一些问题。在实验过程中，首先初步熟悉了 Linux 系统，掌握了它的一些编程方法。其次对Linux的一些系统函数以及结构体也有了一定的了解。在这编程和使用的过程中，尽管出现了很多困难，软件不熟悉，代码不断报错出现 bug，甚至是开发环境不知道如何配置（是真的），但是通过同学的帮助都顺利解决了这些问题。所以这也表明了我们还需要不断的学习和进步，这样才能进一步完善我们的代码，努力做得更好。

（3）英文版教材和课件真的不好理解，特别是对于我们这种英语基础差六级还没过的学生，希望老师可以给个中文版作业。

1. **附录**
2. 数独解决方案验证器

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<pthread.h>

int sudo[9][9]=

{

{6,2,4,5,3,9,1,8,7},

{5,1,9,7,2,8,6,3,4},

{8,3,7,6,1,4,2,9,5},

{1,4,3,8,6,5,7,2,9},

{9,5,8,2,4,7,3,6,1},

{7,6,2,3,9,1,4,5,8},

{3,7,1,9,5,6,8,4,2},

{4,9,6,1,8,2,5,7,3},

{2,8,5,4,7,3,9,1,6}

};

int status[27];

typedef struct

{

int left;

int top;

int right;

int buttom;

int index;

}CheckRange;

static void workthread(void \*param)

{

CheckRange \*cr=(CheckRange\*)param;

int i,j,k;

for(k=1;k<=9;k++)

{

for(i=cr->left;i<cr->right;i++)

{

for(j=cr->top;j<cr->buttom;j++)

if(k==sudo[i][j])

break;

if(j<cr->buttom)

break;

}

if(i>=cr->right||j>=cr->buttom)

break;

}

if(k>9)

status[cr->index]=1;

else

status[cr->index]=2;

}

int main(void)

{

CheckRange \*cr;

pthread\_t tid;

int i,j,index=0;

for(i=0;i<27;i++)

status[i]=0;

//check row

for(i=0;i<9;i++)

{

cr=(CheckRange \*)malloc(sizeof(CheckRange));

cr->left=0;

cr->right=9;

cr->top=i;

cr->buttom=i+1;

cr->index=index++;

pthread\_create(&tid,NULL,(void\*(\*)(void\*))workthread,cr);

}

//check col

for(i=0;i<9;i++)

{

cr=(CheckRange \*)malloc(sizeof(CheckRange));

cr->left=i;

cr->right=i+1;

cr->top=0;

cr->buttom=9;

cr->index=index++;

pthread\_create(&tid,NULL,(void\*(\*)(void\*))workthread,cr);

}

1. 多线程排序

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<pthread.h>

#define MAXNUM 10

int originallist[MAXNUM]={7,12,19,3,18,4,2,6,15,8};

int mergelist[MAXNUM];

typedef struct

{

int start;

int end;

}Range;

typedef struct

{

Range r1;

Range r2;

pthread\_t t1;

pthread\_t t2;

}MergeParam;

static void workthread(void \*param)

{

Range \*r=(Range\*)param;

int i,j,t;

for(i=r->start;i<r->end;i++)

for(j=i+1;j<r->end;j++)

{

if(originallist[i]>originallist[j])

{

t=originallist[i];

originallist[i]=originallist[j];

originallist[j]=t;

}

}

}

static void mergethread(void \*param)

{

MergeParam \*mp=(MergeParam\*)param;

int i,j,k;

void \*status;

pthread\_join(mp->t1,&status);

pthread\_join(mp->t2,&status);

for(i=0,j=mp->r1.start,k=mp->r2.start;i<MAXNUM;i++)

{

if(j<mp->r1.end && k<mp->r2.end)

{

if(originallist[j]<originallist[k])

{

mergelist[i]=originallist[j];

j++;

}

else

{

mergelist[i]=originallist[k];

k++;

}

}

else if(j<mp->r1.end )

{

mergelist[i]=originallist[j];

j++;

}else

{

mergelist[i]=originallist[k];

k++;

}

}

}

int main(void)

{

MergeParam mp;

pthread\_t tid;

int i;

void \*status;

mp.r1.start=0;

mp.r1.end=5;

mp.r2.start=5;

mp.r2.end=10;

pthread\_create(&(mp.t1),NULL,(void\*(\*)(void\*))workthread,&(mp.r1));

pthread\_create(&(mp.t2),NULL,(void\*(\*)(void\*))workthread,&(mp.r2));

pthread\_create(&tid,NULL,(void\*(\*)(void\*))mergethread,&mp);

pthread\_join(tid,&status);

for(i=0;i<MAXNUM;i++)

printf("%d ",mergelist[i]);

printf("\n");

return 0;

}

1. **部分参考资料**

[[总结]Linux下的线程控制 - Linux专题站 - 博客园](https://www.baidu.com/link?url=SWGLGwfeci8jN5P6jo2-inXrkNNTOoua6bMXXgae5s0LDo8ahER6zbA0GlI4AU3M9Px71NAc3Cb2KcokSH9LWa&wd=&eqid=ad8ce17e00039258000000035cb20512)：<https://www.cnblogs.com/caolicangzhu/p/6956377.html>

Linux中的线程控制：<https://www.jianshu.com/p/f4446c823195>