**实验报告**

课程名称：

学生姓名：

学号：

指导老师：

实验1：OpenMP 基础实验

1. **实验目的**
2. 了解OpenMP的运行环境
3. 掌握OpenMP编程的基本要素、编译方法，可运用相关知识独立完成一个基本的OpenMP程序的编写与调试过程。
4. **实验要求**
5. 掌握OpenMP运行环境

在ubuntu环境中打开一个终端界面。尝试在图形操作界面左侧寻找终端的图标进行点击，或直接使用快捷键**Ctrl+Alt+T**打开终端界面进行Shell环境。

1. 运行一个简单OpenMP程序

程序代码见程序1-1、1-2

1. OpenMP兼容性检查

通过检查预处理宏\_OPENMP 是否定义来进行条件编 译。如果定义了\_OPENMP，则包含omp.h 并调用OpenMP 库函数。

程序代码见程序1-3

1. 常用线程操作库函数语句

在OpenMP编程过程中，一旦涉及线程操作，有较大的概率使用三个常用的库函数，分别为：

(1) int omp\_get\_num\_threads(void) 获取当前线程组（team）的线程数量，

如果不在并行区调用，则返回1。

(2) int omp\_get\_thread\_num(void) 返回当前线程号。

(3) int omp\_get\_num\_procs(void) 返回可用的处理核个数。

注意区别这三个库函数的外形及意义，特别是前两个库函数，初始使用时很容易混淆。

程序代码见程序1-4

1. parallel语句的练习

parallel 用来构造一个并行区域，在这个区域中的代码会被多个线程（线程组）执行，在区域结束处 有默认的同步（隐式路障）。 我们可以在parallel 构造区域内使用分支语句，通过omp\_get\_thread\_num 获得的当前线程编号来指定不同线程执行区域内的不同代码。

程序代码见程序1-5、1-6

1. critical和reducation语句的练习

为了保证在多线程执行的程序中，出现资源竞争的时候能得到正确结果， OpenMP 提供了3种不同类型的多线程同步机制：排它锁、原子操作和临界区。其中临界区是最简单最常见，也是最不灵活效率最低的一种。 编译制导critical 声明一个临界区，临界区一次只允许一个线程执行，因此它在并行代码提供了一种串行的机制。critical 结束处没有隐式路障。

程序代码见程序1-7、1-8、1-9、1-10

1. parallel for语句的练习

我们在最求自动并行处理的道路上还想取得进一步的优化效果。使用OpenMP 的目的就是要尽可能地降低并行编程的难度。从程序1-10中我们可以看到，对于累加数据分配到线程的操作是通过线程编号计算得到的，能不能让OpenMP 自动去处理。这就是parallel for 子句的作用。

程序代码见程序1-11

1. **实验程序及结果截图**

程序1-1

#include<iostream>

#include<mpi.h>

using namespace std;

int main(int argc, char\* argv[])

{

int rank,size,len;

char version[MPI\_MAX\_LIBRARY\_VERSION\_STRING];

MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

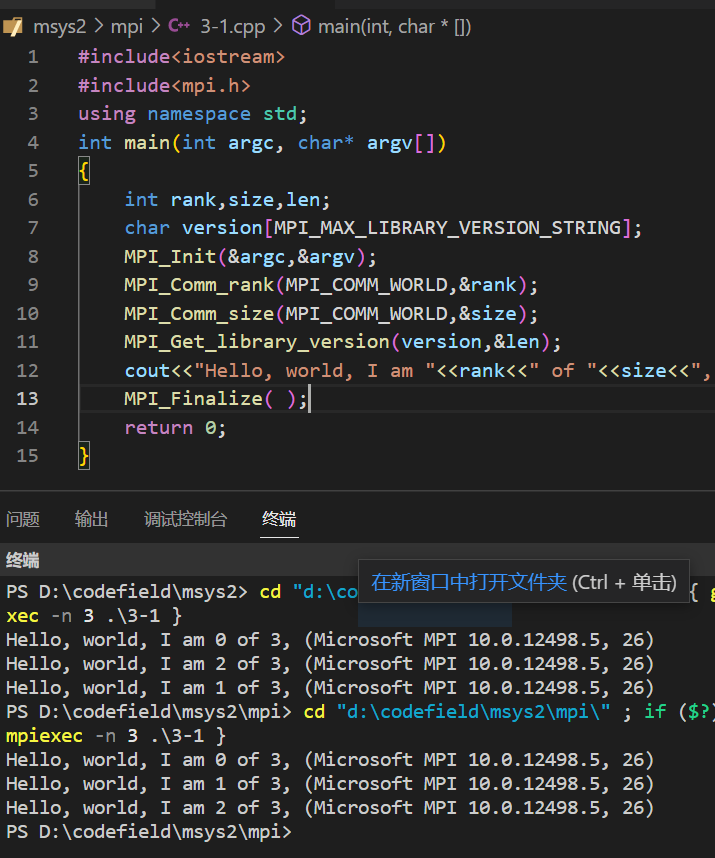
MPI\_Get\_library\_version(version,&len);

cout<<"Hello, world, I am "<<rank<<" of "<<size<<", ("<<version<<", "<<len<<")\n";

MPI\_Finalize( );

return 0;

}

****

程序1-2

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size,tag=1;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

MPI\_Send(&senddata,1,MPI\_INT,1,tag,MPI\_COMM\_WORLD);

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,tag,MPI\_COMM\_WORLD,&status);

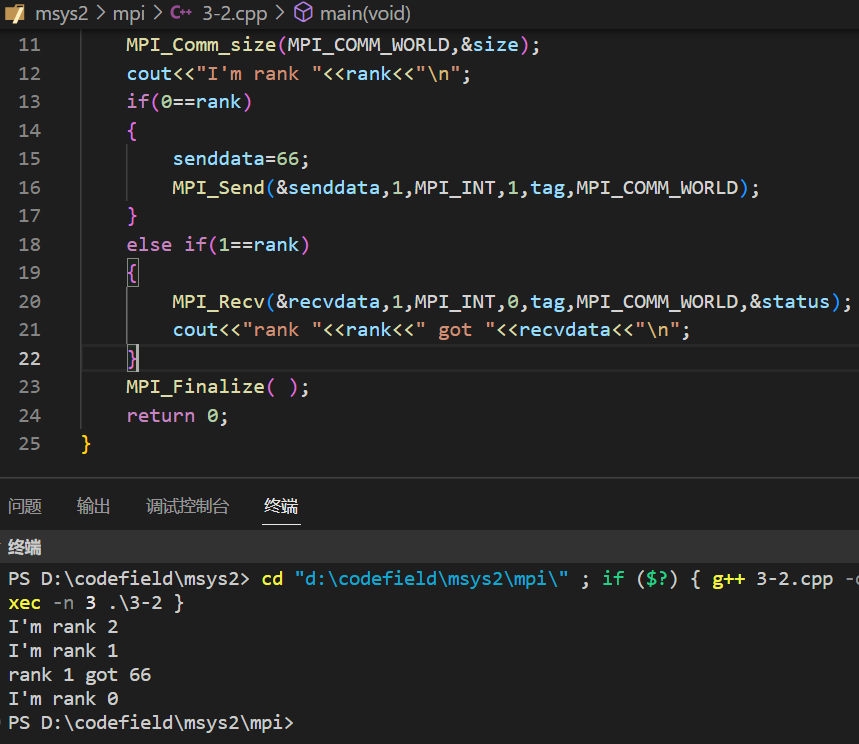
cout<<"rank "<<rank<<" got "<<recvdata<<"\n";

}

MPI\_Finalize( );

return 0;

}



程序1-3

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

MPI\_Send(&senddata,1,MPI\_INT,2,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<"\n";

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,1,MPI\_COMM\_WORLD,&status);

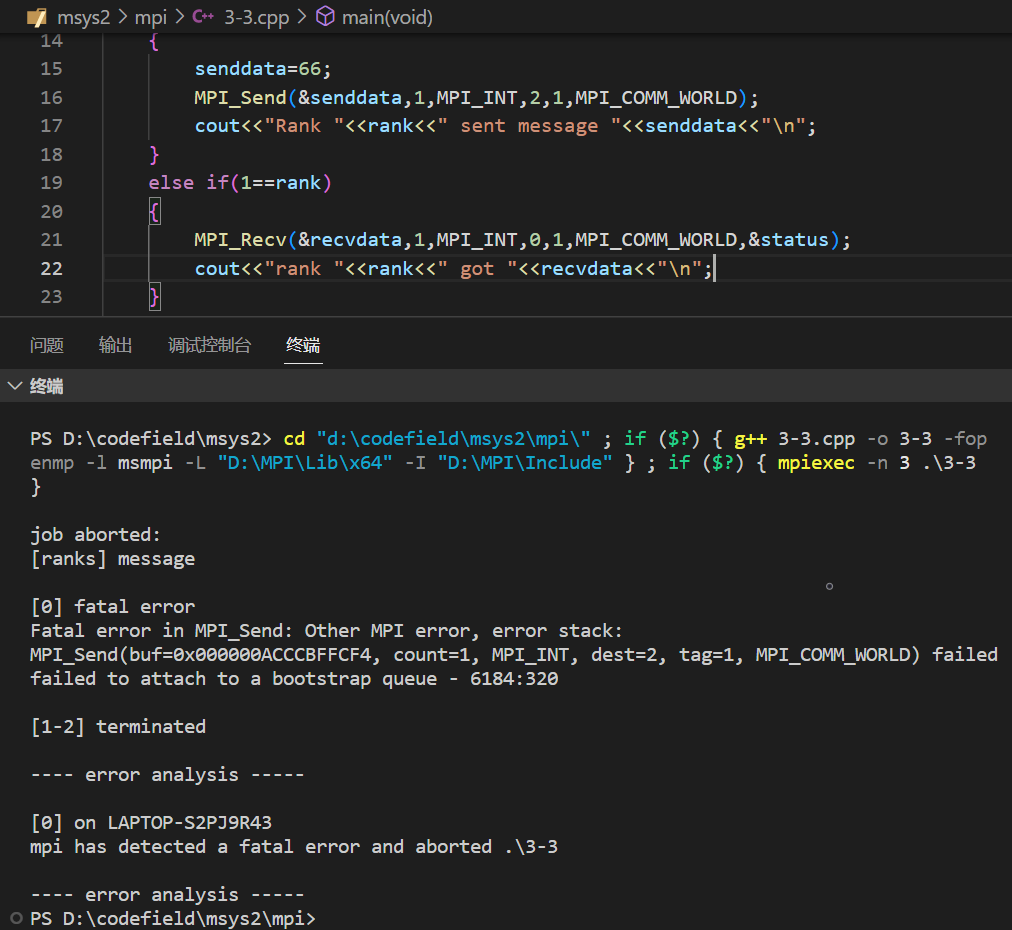
cout<<"rank "<<rank<<" got "<<recvdata<<"\n";

}

MPI\_Finalize( );

return 0;

}



程序1-4

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,2,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,3,MPI\_COMM\_WORLD,&status);

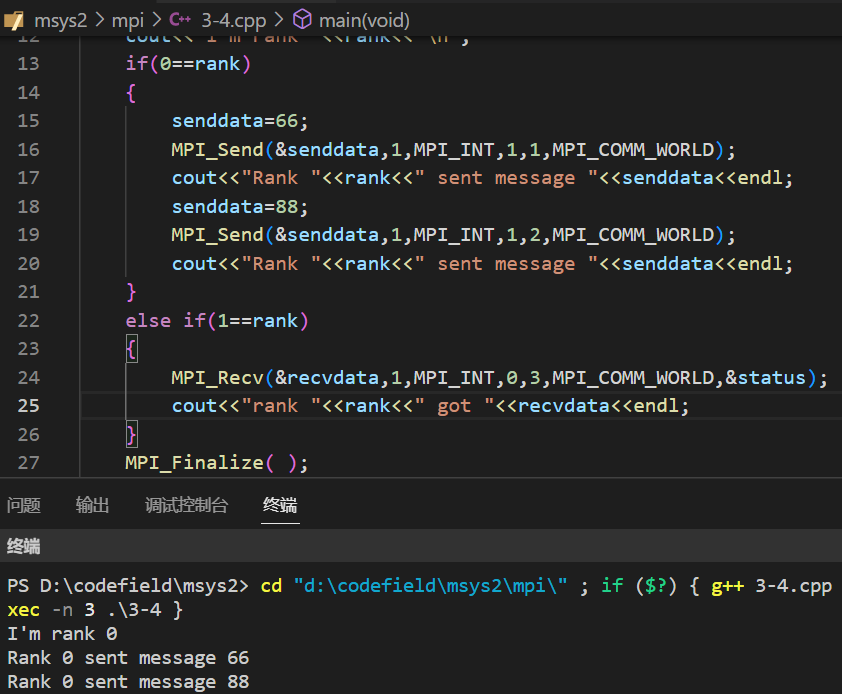
cout<<"rank "<<rank<<" got "<<recvdata<<endl;

}

MPI\_Finalize( );

return 0;

}



程序1-5

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,2,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,1,MPI\_COMM\_WORLD,&status);

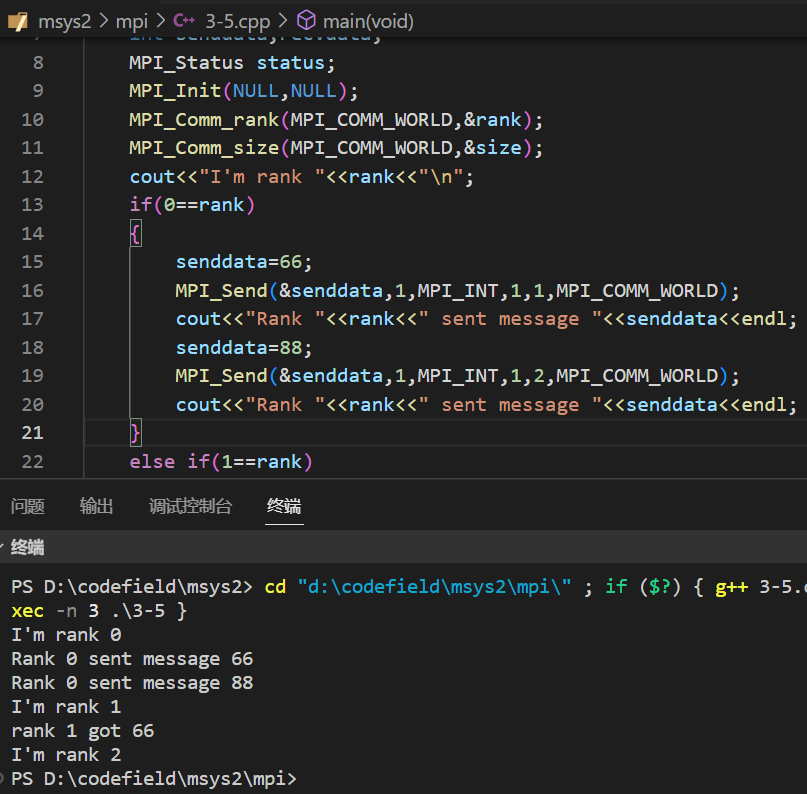
cout<<"rank "<<rank<<" got "<<recvdata<<endl;

}

MPI\_Finalize( );

return 0;

}



程序1-6

#include<iostream>

#include<mpi.h>

#include<unistd.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

sleep(20);

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(2==rank)

{

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,1,MPI\_COMM\_WORLD,&status);

cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

MPI\_Recv(&recvdata,1,MPI\_INT,2,1,MPI\_COMM\_WORLD,&status);

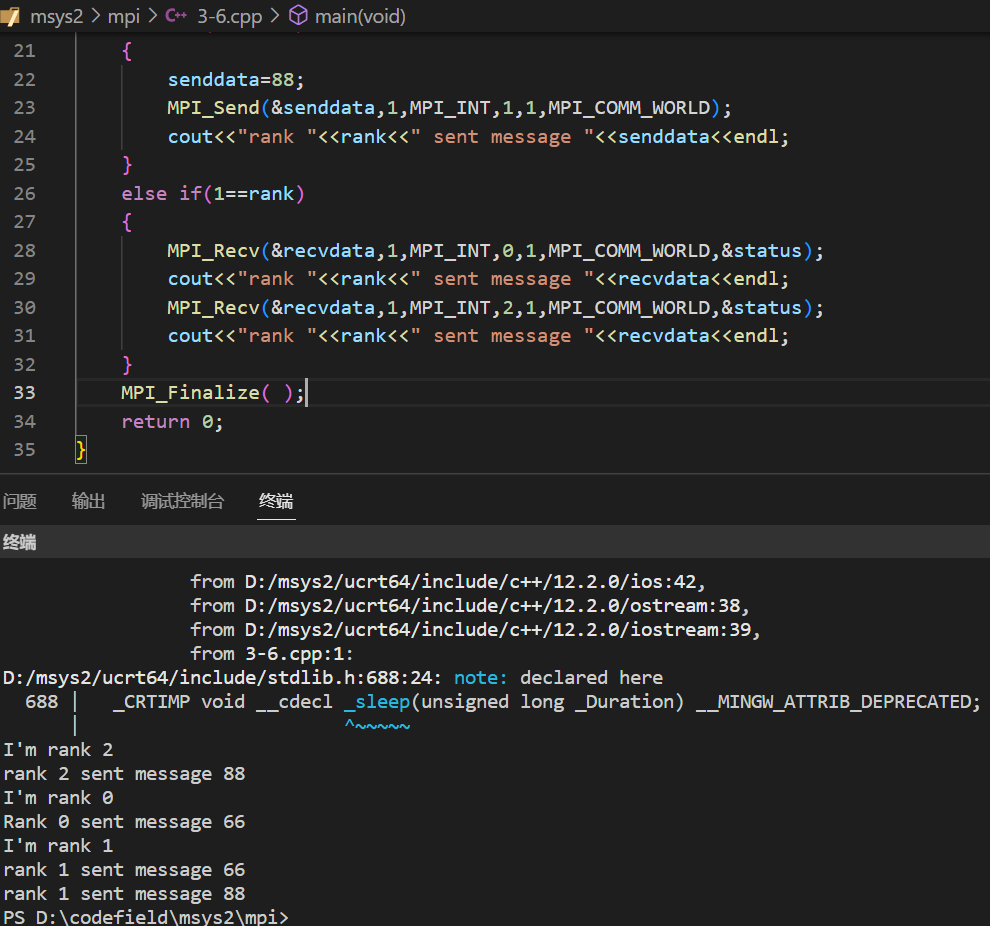
cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

}

MPI\_Finalize( );

return 0;

}



程序1-7

#include<iostream>

#include<mpi.h>

#include<unistd.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

sleep(20);

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(2==rank)

{

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,MPI\_ANY\_SOURCE,1,MPI\_COMM\_WORLD,&status);

cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

MPI\_Recv(&recvdata,1,MPI\_INT,MPI\_ANY\_SOURCE,1,MPI\_COMM\_WORLD,&status);

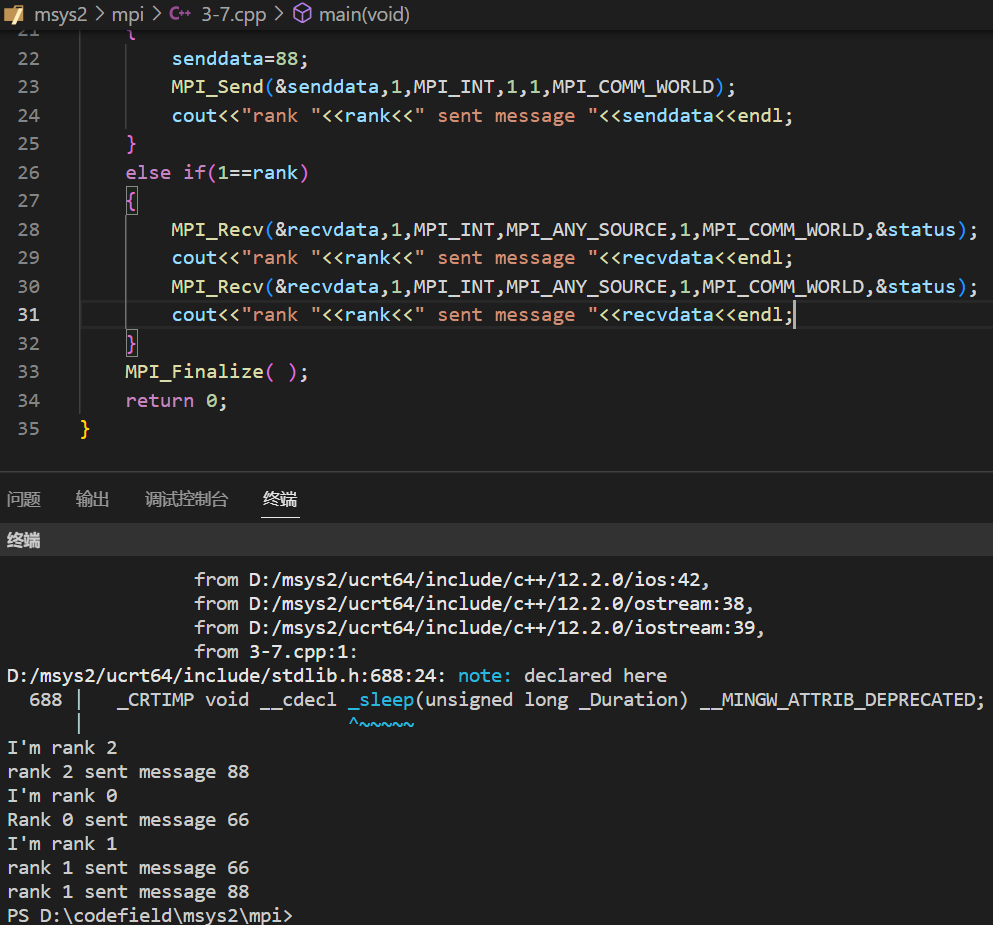
cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

}

MPI\_Finalize( );

return 0;

}



程序1-8

#include<iostream>

#include<mpi.h>

#include<unistd.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

sleep(20);

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(2==rank)

{

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"rank "<<rank<<" sent message "<<senddata<<endl;

}

else if(1==rank)

{

MPI\_Recv(&recvdata,1,MPI\_INT,0,MPI\_ANY\_TAG,MPI\_COMM\_WORLD,&status);

cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

MPI\_Recv(&recvdata,1,MPI\_INT,0,MPI\_ANY\_TAG,MPI\_COMM\_WORLD,&status);

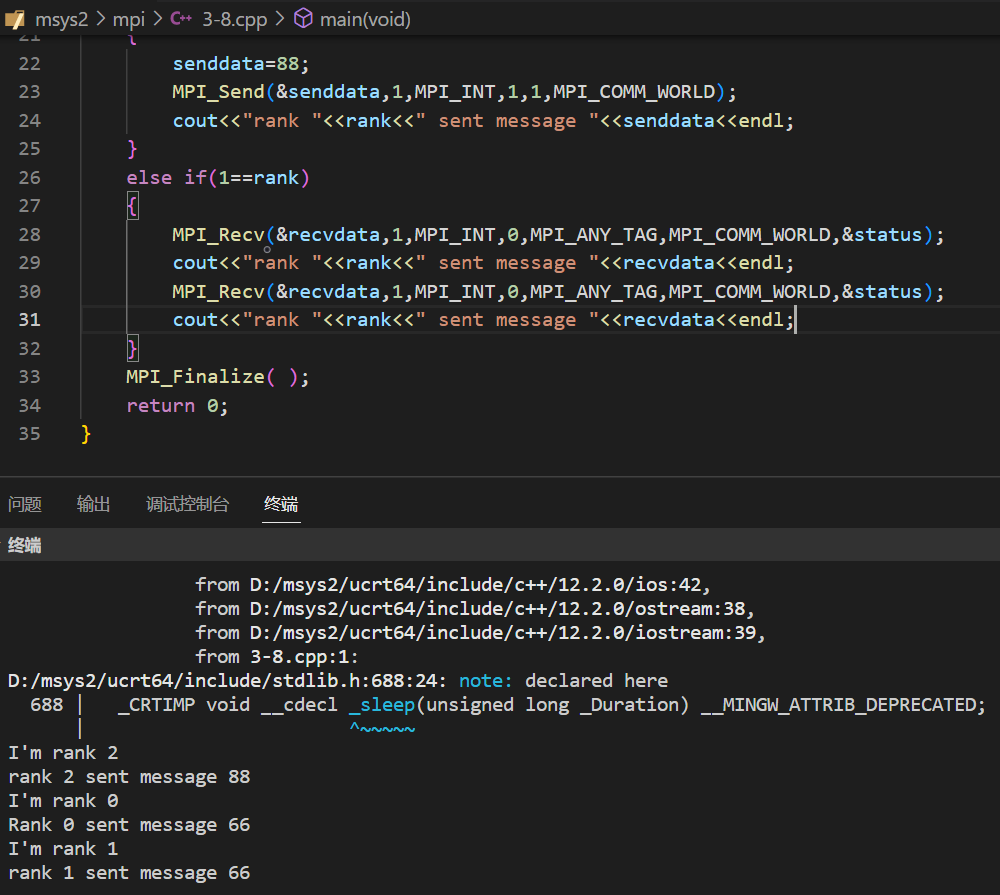
cout<<"rank "<<rank<<" sent message "<<recvdata<<endl;

}

MPI\_Finalize( );

return 0;

}



程序1-9

#include<iostream>

#include<mpi.h>

#include<unistd.h>

using namespace std;

int main(void)

{

int rank,size;

int senddata,recvdata;

MPI\_Status status;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

cout<<"I'm rank "<<rank<<"\n";

if(0==rank)

{

senddata=66;

sleep(20);

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"Rank "<<rank<<" sent message "<<senddata<<" with tag 1"<<endl;

}

else if(2==rank)

{

senddata=88;

MPI\_Send(&senddata,1,MPI\_INT,1,1,MPI\_COMM\_WORLD);

cout<<"rank "<<rank<<" sent message "<<senddata<<" with tag 1"<<endl;

sleep(5);

MPI\_Send(&senddata,1,MPI\_INT,1,2,MPI\_COMM\_WORLD);

cout<<"rank "<<rank<<" sent message "<<senddata<<" with tag 2"<<endl;

}

else if(1==rank)

{

int flag=1;

while(flag)

{

MPI\_Recv(&recvdata,1,MPI\_INT,MPI\_ANY\_SOURCE,MPI\_ANY\_TAG,MPI\_COMM\_WORLD,&status);

switch (status.MPI\_TAG)

{

case 1:

cout<<"Rank "<<rank<<" got message "<<recvdata<<" with tag 1"<<endl;

break;

case 2:

cout<<"Rank "<<rank<<" got message "<<recvdata<<" with tag 2"<<endl;

flag=0;

break;

}

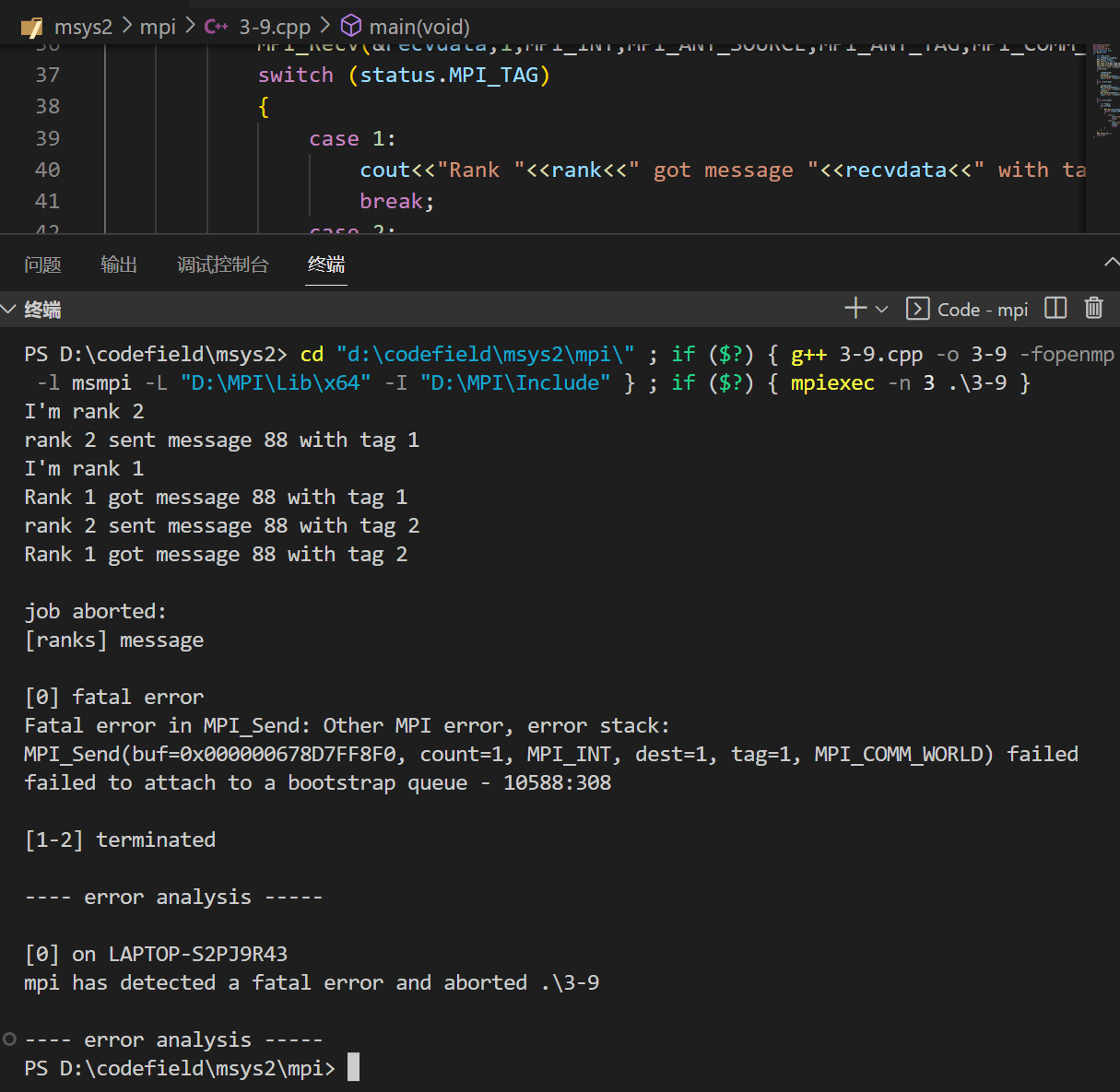
}

}

MPI\_Finalize( );

return 0;

}



1. **总结**

谈谈实验中遇到的问题及如何解决的，在实验中的一些个人体会。

基于windows，vscode配置mpi还是比较简单的，下载两个mpi的库，在coderunner的编译命令里链接上mpi库，就可以运行并行程序了。

Mpi信息传递还是比较好理解的，根据计网来理解就是tag是信道，comm是广播域。MPI\_ANY\_SOURCE 和MPI\_ANY\_TAG可以解决一部分阻塞问题。