实验三、安装 MPI

一、Linux命令及脚本文件

目的与要求：使学生掌握 Linux 基本命令的使用；学会编写简单的脚本文件。

主要内容及提示：

1.Linux 基本命令

1.1 帮助

所有的 Linux 命令及 C，MPI 函数的使用说明都可以通过联机帮助获得。

man 〈命令|C 语言函数名|MPI 函数名〉 #如 man ls,按 q 退出

〈命令名〉—help

1.2 文件、目录、磁盘操作命令

ls 查看目录中的内容，常用的参数有 -al

pwd 显示当前工作目录

cd 改变当前工作目录，(.表示当前目录；..表示父目录；~表示当前用户的主目录)

mkdir 建立子目录

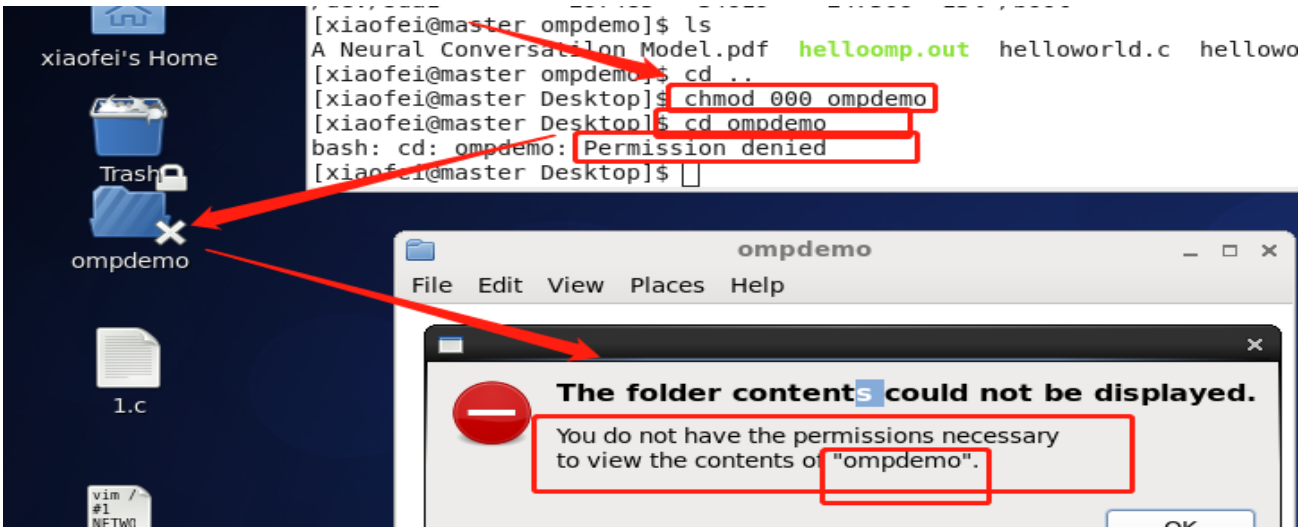
cp 复制文件或子目录，可用参数 -R

cat 显示文件内容

more 分屏输出（回车：上滚一行；空格：上滚一屏；q：退出）

less 分屏输出（q 键退出、还可以使用光标上下移动键）

chmod 改变文件或目录的读(r)写(w)执行(x)属性（三组 rwx 中，第一组 rwx 为该文件主的权限、第二组 rwx 为同一用户组中其他用户的权限、第三组 rwx 为其他用户的权限。例如：chmod 765 <文件名>，则文件的读写执行权限为 rwxrw-r-x，即 7=111B，6=110B，5=101B，这里 B 表示二进制），如下所示：



ln　　建立文件链接，常用参数-s（相当于 Windows 中的“快捷方式”）

```
[fei@master ompdemo]$ ln helloomp.out ho
[fei@master ompdemo]$ ./ho
Hello World from OMP thread 0
Number of threads is 1
```

rm　　删除文件，删除子目录（需要参数-r）

mv　　移动文件（目标文件若存在，则会被覆盖）

gzip　gz 文件的压缩/解压缩，常用参数-d(解压缩)，-9（最大压缩）

tar　　tar，tgz 文件的打包/解包，常用参数：

-cvf（打包成 tar 文件），

-xvf（解 tar 文件包），

-zcvf（打包并压缩成 tgz 文件），

-zxvf(解 tgz 文件)

df　　显示硬盘各分区的使用情况（直接输入回车）

mount　　装载文件系统

umount　　卸载文件系统

1.3　m 命令集——DOS/Windows 格式软盘的使用

mdir 或 mdir a:　　显示软盘当前目录内容

mcopy　　Linux 操作系统中的文件与 DOS/Windos 格式软盘之间复制文件

mtype　　显示软盘中指定文件的内容

mren　　更改软盘中指定文件的文件名

mdel　　删除软盘中指定文件

mcd　　改变软盘的当前目录

mmd　　在软盘上建立子目录

mrd　　删除软盘上的子目录

1.4 C/C++/Fortran 77/Fortran 90 及 MPI 程序的编译链接

cc [–o 〈可执行文件名〉] 〈源程序文件名.c|.cpp〉–l〈库名〉

例如：若 exam1.c 使用了数学函数（如：sqrt(x)）则

命令应为 cc -o exam1 exam1.c -lm （这里 m 表示将链接库文件 libm.a）

若缺省参数-o 〈可执行文件名〉则默认的可执行文件名为 a.out

cc -c 〈目标代码文件名.o〉 〈源程序文件名.c|.cpp〉　　仅编译不链接

gcc, f77, g77, make, mpicc, mpiCC, mpif77, mpif90 等格式相同。

1.5 重新定向与管道操作

>、>> （输出重新定向）；< （输入重新定向）；| （管道）

1.6 系统状态

clear　　清屏

top　　显示系统状态（各进程 CPU、内存）　　#按“q”退出

ps　　显示已经启动的进程，常用参数有 -x

hostname　显示当前计算机名，常用参数有 -i

date　　显示系统日期时间

set　　显示/设置系统环境变量

env　　显示/设置系统环境变量

kill　　终止指定进程

1.7 与用户有关的操作

logout　　退出登录

exit　　退出登录

whoami　　显示当前的用户名（Who am I?）

1.8 远程操作命令

rsh <主机名> [-l 用户名] [〈命令〉[命令的参数]]

rcp [用户名@]主机名:路径/文件名 [用户名@]机器名:路径/文件名

telnet 机器名

ftp 机器名（自行查阅资料）

注：查看主机名—hostname　　查看用户名—whoami

1.9 编辑器的使用

vim 文件名

<ESC>键——进入 vi 的命令状态，击冒号和命令，例如，“:wq”存盘退出，“:q!”不存盘退出）

i 键——进入 vi 的插入状态

1.10 关于 bash（批处理）

用户事先写一个 Shell 脚本(Script)，其中有很多条命令，让 Shell 一次把这些命令执行完，而不必一条一条地敲命令。

1.11 .bashrc 文件(在主目录下使用命令：ls -a 可见)中可以设置一些环境变量，如：命令程序的检索路径 PATH 等。

1.12 shell 提供的小窍门"~"：(~可表示用户的登录主目录，事实上并没有一个目录名为~)。

〈tab〉键的补全功能：在 Linux 命令处理程序中（即命令行提示符$>下）可以使用〈tab〉键由 Linux 系统自动补全可能的文件名。

2.脚本文件

脚本文件是由 Linux 外部命令、内部命令编写的命令程序，可以**实现“批处理”**。脚本文件采用文本文件格式，其文件属性应有可执行成分（可用 chmod +x 〈脚本文件名〉增加这种属性）。在命令提示符下键入脚本文件名（必要时可带多个参数）启动脚本文件执行。脚本文件中除可以执行 Linux 命令外，还可以执行算术计算、条件判断、循环等基本流程控制语句。

例如：欲在所有计算机中的 gcc 用户主目录中建立一个子目录（假设有同学“张山”欲在 n10~n17，共 8 台计算机上都建立 ZSMPI 子目录，即目录/home/gcc/ZSMPI），可以使用如下命令启动下面的脚本文件

命令：　./rmkdirs ZSMPI 10 17　　# 注：其中“rmkdirs”为脚本文件名

脚本文件内容：

```
#FILENAME: rmkdirs
if [ -z $3 ]
         then echo "$0 <subdirname> <num_node_start> <num_node_end>"
       echo "(e.g.)  $0 ZSMPI 10 17"
       exit
fi
m=$2
while [ $m -le $3 ]
do
    echo ”rsh n$m mkdir $1”
    rsh n$m mkdir $1
    let m=$m+1
done
```

注：另一种方法，机器名不一定连续，则需要在脚本文件中列出。脚本文件如下，启动脚本文件只需要一个参数——子目录名。

```
# FILENAME: rmkdirs1
if [ -z $1 ]:then
    echo "$0 < subdirname >"
    echo "(e.g.)  $0 ZSMPI"
    exit
fi
for m in 10 11 12 13 17 15 14 16
do
    echo ”rsh n$m mkdir $1”
    rsh n$m mkdir $1
done
```

本次实验---------建立一个脚本同时在三台主机中建立一个文件夹：

（1） 首先确保各个主机之间可以使用 ssh 进行无密码访问

（2） 使用 **vim 编写实验代码，并对以下的相关路径进行修改，建立脚本文件：vim mkdirs.sh**

```
#!/bin/bash

# judge whether the length of var you input is 0
if [ -z $1 ]
then
        echo "$0 < subdirname >"
        #举例 echo "(e.g.)   $0 ZSMPI"
        exit
fi
```

```
#build the dir in the machine of master, slave1 and slave2 at the same time
if    [ -d /home/fei/$1 ];then    #$1  是传递给该 shell 脚本的第一个参数。
echo "the target dir is exiting"
else
        for m in master slave1 slave2
                do
                        echo ”ssh $m mkdir $1”
                        ssh $m mkdir -p /home/fei/$1
                done
fi
```

（3） 运行以上建立的脚本文件：

- 首先给脚本文件授予运行权限

  chmod 777 mkdirs.sh (777 表示可读可写可执行)

  或者 chmod +x mkdirs.sh

- 然后以当前路径执行文件（注后面的 ZSMPI 是程序的位置参数，此处表示为所要建立的文件夹名称）

  ./mkdirs.sh ZSMPI_2

（4）脚本运行效果和实验过程图示：

在 master 和 slave1 中分别建立了文件夹 ZSMPI_2

```
[root@slave1 ~]# cat mkdirs.sh
if [ -z $1 ]
        then echo "$0 < subdirname >"
        echo "(e.g.)  $0 ZSMPI"
        exit
fi
for m in master slave1
do
        echo ”ssh $m mkdir $1”
        ssh $m mkdir $1
done


[root@slave1 ~]# chmod 777 mkdirs.sh
[root@slave1 ~]# ./mkdirs.sh ZSMPI_2
”ssh master mkdir ZSMPI_2”
Warning: Permanently added 'master,192.168.217.130' (RSA) to the list of known hos
ts.
”ssh slave1 mkdir ZSMPI_2”
Warning: Permanently added 'slave1,192.168.217.201' (RSA) to the list of known hos
ts.
[root@slave1 ~]# ls
anaconda-ks.cfg  helloomp.out  install.log.syslog  zhangsanDir  ZSMPI_2
helloomp.c       install.log   mkdirs.sh           ZSMPI
[root@slave1 ~]#
```

注：以下是同时判断三台主机下的/home/fei/目录下是否有某一目录，如果有则删除的操作：

```
#!/bin/bash
# judge whether the length of var you input is 0
if [ -z $1 ]
```

```
then
        echo "$0 < subdirname >"
        echo "(e.g.)    $0 ZSMPI"
        exit
fi

#remove    the dir in the machine of master,slave1 and slave2 in time
if    [ -d /home/fei/$1 ];then
for m in master slave1 slave2
do
        echo ”ssh $m rm -rf $1”
        ssh $m rm -rf /home/fei/$1
done
else
        echo "the target dir is not exit"
fi
```

二、安装MPI实践

1.实验目的

使学生掌握 MPI 系统的安装、MPI 程序的编译链接及运行。

2 .MPI 介绍

MPI（Message Passing Interface）是目前发展较快、使用面较广的一个公共的消息传递库，顾名思义，就是在现有机器的软硬件通信基础上，实现了并行应用程序中各并行任务之间的相互通信、协调和同步功能，并对这些并行任务进行管理。MPI 是在众多专家经 多次会谈、综合考虑了 NX/2、Express、Vertex、p4、PARMACS 和 PVM 的优点的前提条件下，于 1993 年形成的一个标准，可谓后起之秀。它具有可移植性、高效性、灵活性和易用性的特点，不仅适用于具有分布式内存的大型机、工作站机群，也可用于具有共享内存的大型机。

3.实验内容

本次实验要求安装 OpenMPI，使用 mpi 环境运行程序。

注：（vim 命令打开文件后，按字母'a'进入编辑，按“ctrl+c”进入命令模式，输 :wq 存盘退出，输 :quit 放弃修改 注，冒号为输入符）

OpenMPI 的安装及使用

目的与要求：

一、 openmpi 的安装及使用

目的与要求：使学生掌握 MPI 系统地安装、MPI 程序的编译链接及运行。

主要内容：

1. openmpi 的安装

1.1 在三台主机上使用 root 用户安装 c++编译器

```
# yum install gcc-c++
```

1.2 在三台主机上使用 root 用户安装 gfortran 编译器

```
# yum install gcc-gfortran
```

1.3 使用普通用户创建文件夹：

[fei@master ~]$ mkdir openmpi202

1.4 下载 openmpi

wget https://download.open-mpi.org/release/open-mpi/v2.0/openmpi-2.0.2.tar.gz

或者从百度云盘下载，拷贝到上述创建的文件夹。

1.5 解压

tar -xzvf openmpi-2.0.2.tar.gz

将 MPICH 文件解压缩、解包到某子目录。

1.6 系统配置

cd openmpi-2.0.2

若当前使用得 Shell 的辅助检索路径中没有设置当前目录，则应使用命令

./configure --prefix=/home/fei/openmpi202/openmpi-2.0.2 CC=gcc CXX=g++ FC=gfortran

注：/home/fei/openmpi202/openmpi-2.0.2 是安装路径

此外在配置过程可以指定编译器或选择用 rsh 或 ssh。

1.7 编译

make

1.8 安装

make install

1.9 进入 master 中安装 openmpi 软件的路径下，将 master 中已经安装好的 openmpi 程序通过 scp -r 命令拷贝到 slave1 和 slave2，省去安装过程：

[fei@master ~]$ scp -r openmpi202/ fei@slave1:/home/fei

[fei@master ~]$ scp -r openmpi202/ fei@slave2:/home/fei

部分过程如图：

```
[fei@master ~]$ ls
Desktop     examples_openmi202    openmpi202  Templates
Documents   mpi_installed_folder  Pictures    Videos
Downloads   Music                 Public
[fei@master ~]$ pwd
/home/fei
[fei@master ~]$ scp -r openmpi202/ fei@slave1:/home/fei
```

拷贝好之后，分别进入到对应主机安装的 openmpi 程序下面的 bin 目录中，可以使用命令 "./mpicc" 可以运行测试 mpi 编译命令，部分过程如图：

```
[fei@master openmpi-2.0.2]$ cd bin
[fei@master bin]$ ls
mpic++    mpif90      ompi-ps       orte-clean   orte-server   oshrun
mpicc     mpifort     ompi-server   orted        orte-submit   shmemcc
mpiCC     mpirun      ompi-submit   orte-dvm     orte-top      shmemfort
mpicxx    ompi-clean  ompi-top      orte-info    oshcc         shmemrun
mpiexec   ompi-dvm    opal_wrapper  orte-ps      oshfort
mpif77    ompi_info   ortecc        orterun      oshmem_info
[fei@master bin]$ pwd
/home/fei/openmpi202/openmpi-2.0.2/bin
[fei@master bin]$ ./mpicc
gcc: no input files
[fei@master bin]$
```

1.10 设置辅助检索路径（环境变量），**在三台主机上同时进行以下操作：**

（1）统一添加一条辅助检索路径，方法为在 root 用户下，修改/etc/profile，在文件末尾加上以下三行代码（其中红色字体表示安装路径，需要对应修改）：

export PATH=/home/fei/openmpi202/openmpi-2.0.2/bin:$PATH

export INCLUDE=/home/fei/openmpi202/openmpi-2.0.2/include:$INCLUDE

export LD_LIBRARY_PATH=/home/fei/openmpi202/openmpi-2.0.2/lib:$LD_LIBRARY_PATH

（2）在普通用户 fei 下，修改~/.bashrc 文件，，在文件末尾加上以下三行代码（其中红色字体表示安装路径，需要对应修改）：

export PATH=/home/fei/openmpi202/openmpi-2.0.2/bin:$PATH

export INCLUDE=/home/fei/openmpi202/openmpi-2.0.2/include:$INCLUDE

export LD_LIBRARY_PATH=/home/fei/openmpi202/openmpi-2.0.2/lib:$LD_LIBRARY_PATH

（3）root 用户和下，运行以下命令使修改生效：

source /etc/profile

在 fei 用户，运行以下命令使修改生效：

source ~/.bashrc

注：在/etc/profile 中配置的环境变量使用于所有用户，在系统启动的时候就会加载，如果将其中的环境变量删除后，需要重启计算机方可生效；在~/.bashrc 中设置的环境变量只适用于当前用户；可通过命令$ env | grep -w LD_LIBRARY_PATH 和$ env | grep -w PATH 来查看环境变量修改是否成功。可运行目录 examples 下的例子程序查看是否安装成功

在安装时可能需要一些软件，可以通过 yum install ** 进行安装，例如：

```
*** C++ compiler and preprocessor
checking for g++... no
checking for c++... no
checking for gpp... no
checking for aCC... no
checking for CC... no
checking for cxx... no
checking for cc++... no
checking for cl.exe... no
checking for FCC... no
checking for KCC... no
checking for RCC... no
checking for xlC_r... no
checking for xlC... no
checking whether we are using the GNU C++ compiler... no
checking whether g++ accepts -g... no
checking dependency style of g++... none
checking how to run the C++ preprocessor... /lib/cpp
configure: error: in `/usr/hpl/openmpi-2.0.2':
configure: error: C++ preprocessor "/lib/cpp" fails sanity check
See `config.log' for more details
```

是由于没有编译器，因此可以网上查找软件的名称，然后安装。如下指令可以完成安装该编译器

```
[root@pc1 ~]# yum install gcc-c++
```

2. 运行 MPI 示例程序

使用 mater 主机的普通用户 fei, 在桌面建立一个文件夹， mpi-hello-world，在该文件夹下面建立一个 mpi 程序，名为：mpi_hello_world.c，程序代码如下：

```
// This code is provided freely with the tutorials on mpitutorial.com. Feel
// free to modify it for your own use. Any distribution of the code must
// either provide a link to www.mpitutorial.com or keep this header intact.
//
```

```
// An intro MPI hello world program that uses MPI_Init, MPI_Comm_size,
// MPI_Comm_rank, MPI_Finalize, and MPI_Get_processor_name.
//
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  // Initialize the MPI environment. The two arguments to MPI Init are not
  // currently used by MPI implementations, but are there in case future
  // implementations might need the arguments.
  MPI_Init(NULL, NULL);

  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d out of %d processors\n",
         processor_name, world_rank, world_size);

  // Finalize the MPI environment. No more MPI calls can be made after this
  MPI_Finalize();
}
```

2..1 编译 MPI 源程序

在当前路径/home/fei/Desktop/mpi-hello-world 下将 mpi_hello_world.c 源程序编译成可执行程序 mpi_hello_world，编译后的文件名可以随意取，运行代码如下：

[fei@master mpi-hello-world]$ mpicc -o mpi_hello_world mpi_hello_world.c

2.2 将当前可执行程序拷贝到主机 slave1 和 slave2 的相同路径下：

返回到桌面，执行以下命令，可以将包含 mpi_hello_world 和 mpi_hello_world.c 两个文件的文件夹 mpi-hello-world 拷贝到主机 slave1 和 slave2 的桌面：

[fei@master Desktop]$ scp -r mpi-hello-world fei@slave1:/home/fei/Desktop

[fei@master Desktop]$ scp -r mpi-hello-world fei@slave2:/home/fei/Desktop

2.3 运行 MPI 程序

（1）单机运行 mpi 程序

进入文件夹，输入以下命令即可在本机上使用 4 个进程运行以上创建的可执行程序：

[fei@master mpi-hello-world]$ mpirun -np 4 mpi_hello_world

（-n num 设置执行 MPI 程序的进程总数）

实验结果如图：

```
[fei@master mpi-hello-world]$ mpirun -np 4 mpi_hello_world
Hello world from processor master, rank 0 out of 4 processors
Hello world from processor master, rank 2 out of 4 processors
Hello world from processor master, rank 3 out of 4 processors
Hello world from processor master, rank 1 out of 4 processors
[fei@master mpi-hello-world]$ ^C
```

代码说明：

This starts a four-process parallel application, running four copies of the executable named mpi_hello_world.

（2）多机运行 mpi 程序（请在执行之前，先确保主机 master 可以 ssh 无密码访问 master、slave1 和 slave2，并且三个主机之间 mpiexec 环境变量都已经配置好）

- 首先建立多机运行的配置文件

  在当前路径下建立一个 hosts_file 文件，里面内容如下：

```
[fei@master mpi-hello-world]$ cat hosts_file
master
slave1
slave2
```

即上面文件中存放要执行 mpi 程序的机器名（又称为节点名），其中第一行要写成执行程序的本机名（即此代码开始运行的那个主机名，此处是 master 主机）,机器名（结点名）可重复出现，表示在该机器（结点）上将启动若干个线程运行

- 运行程序：

  [fei@master mpi-hello-world]$ mpiexec -machinefile ./hosts_file -np 12 mpi_hello_world

  运行结果如下：

```
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts file -np 12 mpi hello world
Hello world from processor master, rank 2 out of 12 processors
Hello world from processor master, rank 3 out of 12 processors
Hello world from processor slave2, rank 10 out of 12 processors
Hello world from processor slave2, rank 11 out of 12 processors
Hello world from processor slave1, rank 4 out of 12 processors
Hello world from processor slave1, rank 5 out of 12 processors
Hello world from processor slave1, rank 7 out of 12 processors
Hello world from processor master, rank 1 out of 12 processors
Hello world from processor slave2, rank 8 out of 12 processors
Hello world from processor slave1, rank 6 out of 12 processors
Hello world from processor slave2, rank 9 out of 12 processors
Hello world from processor master, rank 0 out of 12 processors
```

代码解释：

This command will launch one copy of my_parallel_application on each of master、slave1 and slave2 if the machines seted NFS. Note, however, that not all environments require a hostfile. For example, Open MPI will automatically detect when it is running in batch / scheduled environments, and will use host information provided by those systems. Also note that if using a launcher that requires a hostfile and no hostfile is specified, all processes are launched on the local host.

FOR More Information: https://www.open-mpi.org/faq/?category=running#oversubscribing

注意:如果使用的是 root 用户运行(前提是对于 root 用户已经设置了 ssh 无密码登录),则需要加入参数: --allow-run-as-root

```
mpiexec --allow-run-as-root  -machinefile ./hosts_file -np 9  mpi_hello_world
```

运行思考:

在 hosts_file 中配置节点 3 个 master、2 个 slave1 和 1 个 slave2，然后以不同进程数运行 mpi_hello_world 程序，请分析出现该结果的原因。

```
[fei@master mpi-hello-world]$ cat hosts_file
master slots=3
slave1 slots=2
slave2 slots=1

[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 12 mpi_hello_world
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 12 slots
that were requested by the application:
  mpi_hello_world

Either request fewer slots for your application, or make more slots available
for use.
--------------------------------------------------------------------------
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 8 mpi_hello_world
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 8 slots
that were requested by the application:
  mpi_hello_world

Either request fewer slots for your application, or make more slots available
for use.
--------------------------------------------------------------------------
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 7 mpi_hello_world
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 7 slots
that were requested by the application:
  mpi_hello_world

Either request fewer slots for your application, or make more slots available
for use.
--------------------------------------------------------------------------
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 6 mpi_hello_world
```

```
Hello world from processor master, rank 0 out of 6 processors
Hello world from processor master, rank 2 out of 6 processors
Hello world from processor slave2, rank 5 out of 6 processors
Hello world from processor slave1, rank 3 out of 6 processors
Hello world from processor slave1, rank 4 out of 6 processors
Hello world from processor master, rank 1 out of 6 processors
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 5 mpi_hello_world
Hello world from processor master, rank 0 out of 5 processors
Hello world from processor master, rank 1 out of 5 processors
Hello world from processor master, rank 2 out of 5 processors
Hello world from processor slave1, rank 3 out of 5 processors
Hello world from processor slave1, rank 4 out of 5 processors
[fei@master mpi-hello-world]$ mpiexec -machinefile hosts_file -np 4 mpi_hello_world
Hello world from processor master, rank 0 out of 4 processors
Hello world from processor master, rank 1 out of 4 processors
Hello world from processor slave1, rank 3 out of 4 processors
Hello world from processor master
```

三．Pi-MPI+OpenMP混合编程示例

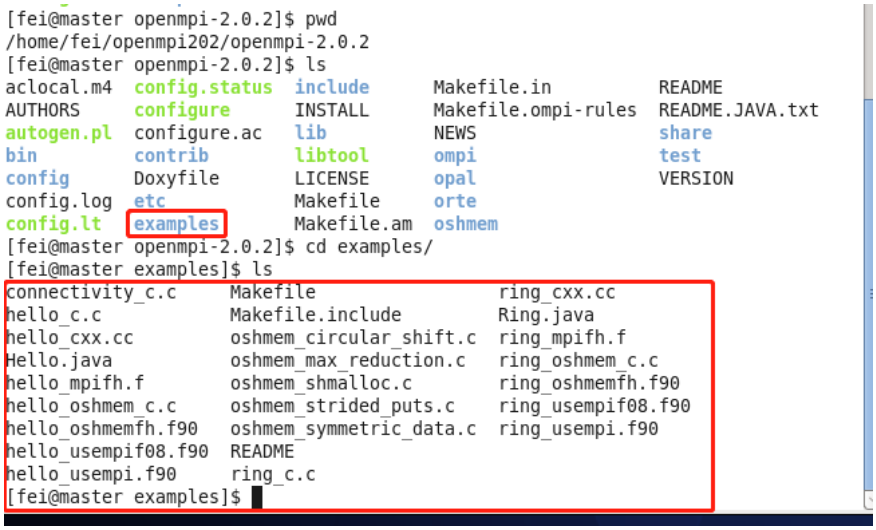
```
#include "mpi.h"
#include "omp.h"
#include <math.h>
#define N 1000000000
int main( int argc, char* argv[] ){
    int rank, nproc;
    int i,low,up;
    double local = 0.0, pi, w, temp;

    MPI_Status    status;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &nproc ); /*get number of processes */
    MPI_Comm_rank( MPI_COMM_WORLD, &rank ); /* get current process id */
    w = 1.0/N; low = rank*(N / nproc); up = low + N/nproc - 1;
#pragma omp parallel for reduction(+:local) private(temp,i)
     for (i=low;i<up; i++){
          temp = (i+0.5)*w;
          local = local + 4.0/(1.0+temp*temp);
     }
    MPI_Reduce(&local, &pi, 1, MPI_DOUBLE, MPI_SUM,
0,MPI_COMM_WORLD);
    if(rank==0) printf("pi = %.20f\n",pi*w);
    MPI_Finalize();
```

}

MPI+OpenMP 程序的编译
编译命令：
$ mpicc -o omp_and_mpi omp_and_mpi.c
运行命令：
mpiexec -machinefile ./hosts_file -np 8 omp_and_mpi

四．其他mpi测试程序（自行运行，不检查）：

（1） 在 openmpi 软件包中的 examples 中，含有大量 mpi 示例程序



（2） 文件拷贝代码示例，自行运行测试

(a)：利用脚本文件实现批量远程拷贝，同时建立机器表配置文件

```
#FILENAME: update
if [ -z $3 ]
    then echo "$0 <filename> <num_node_start> <num_node_end>"
    echo "(e.g.)  $0 hello_c.out 41 48"
    exit
fi

echo "$HOSTNAME"  > hosts
echo "$HOSTNAME 0 $PWD/$1" > pgfile

m=$2
let m=$m+1
while [ $m -le $3 ]
do
        scp $1 n$m:$PWD
```

```
    echo "scp $1 n$m:$PWD"
    echo "n$m" >> hosts
    echo "n$m 1 $PWD/$1" >> pgfile
    let m=$m+1
done
```

注(b)：利用脚本文件实现批量远程拷贝，同时建立机器表配置文件

```
# FILENAME: update1
if [ -z $1 ]
    then echo "$0 <filename>"
    echo "(e.g.)  $0 hello_c.out"
    exit
fi

echo "$HOSTNAME"  > hosts
echo "$HOSTNAME 0 $PWD/$1"  > pgfile

for m in 10 11 12 13 17 15 14 16      #机器名（编号）不一定连续的情形
do
         scp $1 pc$m:$PWD
    echo "scp $1 pc$m:$PWD"
    echo "n$m" >> hosts
    echo "n$m 1 $PWD/$1" >> pgfile
done
```

注 c)：利用上述两个脚本文件之一可以实现批量远程拷贝及建立机器表配置文件。所建立的机器表配置文件分别为 hosts 和 pgfile，它们均为文本文件的格式，分别用于两种不同形式的 mpirun 命令，例如：

```
mpirun -machinefile hosts -np 4 hello_c.out
```

使用上述格式要求可执行文件（如 hello_c.out）在相同的用户（如 gcc）以及相同的子目录中。

```
mpirun -p4pg pgfile -np4 hello_c.out
```

使用上述格式不必要求可执行文件（如 hello_c.out）相同的子目录中。事实上，文件 pgfile 中已经指明。