ECE 473 --Assignment 00
Due Date Specified in BlackBoard
MPI Hello World

In this class we will be making extensive use of Linux and/or Linux-like programming environments. Palmetto is a Linux cluster computer, and as such, having access to a local development platform on your laptop would likely be useful in this course. If you already have Linux installed, all you need to additionally install is an MPI implementation, such as mpich or OpenMPI. This can normally be accomplished via your distribution's package manager GUI or via the commandline as in these examples for mpich:

In Fedora type:
```
sudo yum install mpich2 mpich-devel mpich-doc
```
In Ubuntu type:
```
sudo apt-get install mpich2 mpich2-doc
```

Procedures, dependencies, and package contents vary from distribution to distribution, therefore you may wish to consult with online documentation and help to accomplish this. However, if you run Windows, you may wish to install a dual-boot Linux distribution alongside Windows. Or, you could use Ubuntu Wubi installer to get a similar effect using Ubuntu. Finally, you could potentially run a Linux distribution via a virtual machine, like VirtualBox or VMware, and put Linux inside this, provided you have enough memory and your Windows install is not too hosed. Lastly, if you have Mac OSX, you can do all the MPI programming natively, as with Linux, you'll just need to get the compiler and development tools installed. One way to
do this is to install Xcode via the Apple Store (it is free). Then install mpich or OpenMPI directly from source, a package or via Mac Ports as a package manager. You can read more about this from:

```
http://www.mcs.anl.gov/research/projects/mpich2/
http://www.open-mpi.org/
```

I would also suggest installing Valgrind to get memory bugs worked out. This can normally be installed from source or via your package manager. More on this later.

Create a file called hello.c with the following contents:

```
/* C Example */
#include <stdio.h>
#include <mpi.h>

int main (int argc, char *argv[])
{
int rank, size; /* rank is your pid, staring with 0 */
                /* size, is the number of processes you */
                /* run the program with */

/* never make MPI calls before this and */
/* never touch argc and argv before doing this */
MPI_Init (&argc, &argv);
/* get current process id */
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
/* get number of processes */
MPI_Comm_size (MPI_COMM_WORLD, &size);

printf( "Hello world from process %d of %d\n", rank, size
);
MPI_Finalize(); /* don't make MPI calls after this */
return 0;
}
```

At the commandline, compile hello.c and run it with 4 processes like this:

```
mpicc -g -Wall -Wstrict-prototypes -o hello ./hello.c
mpirun -np 4 ./hello
Hello world from process 1 of 4
Hello world from process 0 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
```

-g means turn on debugging, -Wall turns on most warnings, and -Wstrict-prototypes will enable further checking you will likely need during the course.

This was simply a test to see if you got everything installed, your actual assignment is to modify hello.c to have a loop that will produce output like this when run with 6 processes:

```
_______________Hello world from process 5 of 6
____________Hello world from process 4 of 6
____________Hello world from process 4 of 6
______Hello world from process 2 of 6
______Hello world from process 2 of 6
______Hello world from process 2 of 6
______Hello world from process 2 of 6
Hello world from process 0 of 6
Hello world from process 0 of 6
Hello world from process 0 of 6
Hello world from process 0 of 6
Hello world from process 0 of 6
Hello world from process 0 of 6
_________Hello world from process 3 of 6
_________Hello world from process 3 of 6
_________Hello world from process 3 of 6
___Hello world from process 1 of 6
___Hello world from process 1 of 6
___Hello world from process 1 of 6
___Hello world from process 1 of 6
___Hello world from process 1 of 6
```

Where the printing that each process is doing will be preceded by a number of underscores equal to 3 times it's rank. Note also the number of times each process prints, make sure you account for this.

Try your program with various number of processes to make sure it works. Submit your hello.c file on Blackboard, however, before you upload them, rename this so that the names are first_last_hw00_hello.c (where first and last are your first and last names).

Grading Rubric:

A
fully works, only one print statement is used in program, and no possible bounds overflow of dynamically declared array regardless of number of processes specified during invocation
B
fully works, but might include two print statements and no array usage
C
nearly complete adherence to output specifications
F
doesn't come close to matching specifications

Appendix:

Check out how manual (man) pages work. Man pages are very useful when trying to understand what a function does and what parameters it takes. Type this at the commandline:

```
man MPI_Init
```

and you should see something like:

```
NAME
     MPI_Init - Initialize the MPI execution environment
SYNOPSIS
     #include "mpi.h"
     int MPI_Init(int *argc, char ***argv)
INPUT PARAMETERS
     argc - Pointer to the number of arguments
     argv - Pointer to the argument vector
COMMAND LINE ARGUMENTS
```