

CS442/ECE432: Homework 1

January 31, 2008

General

- This homework is about material covered in Chapter 2 of the text book.
- The answers to this homework are **due February 7, 2008**.
- Submit via e-mail to `riesen@cs.unm.edu` (Mail it before class on the 7th.) Your subject line must say: “Homework 1 Submission”.
- Keep your answers (and programs) succinct.
- Obey the University rules on plagiarism. In particular, do use libraries and the web to find information you need to answer the questions, but do not copy whole answers or programs. Reference your sources. The work you turn in must be *your* work.
- You are allowed to use the example programs on the class web site as a starting point.

Grading

With so many students and programs, we want to make the testing and grading as automated as possible. It is therefore very important that you follow the rules below exactly. If we have to manually inspect your program and modify it to work with our scripts, we will deduct 10 points.

One File Per Program

For each of the exercises below, create a single file name `e_xx.c`, where the `xx` is 01 for the first exercise, 02 for the second, etc.: “`ex_01.c`”, “`ex_02.c`”, ...

For C++ files use “`ex_01.cc`”, “`ex_02.cc`”, ...

Clean, Readable Code

Your code must be clean and readable. Be consistent with you indentation, use descriptive names for your variables and constants, etc.

Your program must compile without warnings (we use the `-Wall` flag for `gcc`). Your program must be valid MPI; e.g., there must be no pending messages when `MPI_Finalize()` is called.

Some of this is subjective, but we will subtract up to 10 points for code we cannot read or understand, even if it works.

Languages

Write your programs in C or C++. They must compile with the GNU C compiler and link against an MPI-2 library. Even when using C++, you must use the C bindings for MPI; i.e., use `MPI_Send()` not `MPI::Comm::Send()`.

Tar File

Put your source code, no executables or object files, into a single tar file and compress it. Name your archive “Hwk1.tgz” It must untar into the current directory; i.e., no sub directories. You can use zip instead of tar. In that case name your file “Hwk1.zip”

The archive must contain a file named “AUTHOR” which contains your name. Thus, a complete archive for this homework would contain the following files **only**: “ex_01.c”, . . . , “ex_05.c”, “AUTHOR”.

Exercise 1: MPIComm_size

Write your own routine to determine how many processes a program is running on, without using `MPI_Comm_size()`. Your program must only use functions from Chapter 2 of the textbook, but not `MPI_Iprobe()` or `MPI_Probe()`.

Call your function that behaves like `MPI_Comm_size()`, `My_Comm_size()`. It must have the same type signature as the original. Place it in a single file without any other functions or a main program. We will compile it separately and then link it to our test program.

10 points For a `My_Comm_size()` that works

10 points If it also works for a 1-process job

Exercise 2: Standard Output

In Example 2.1 in the textbook each process printed its rank number. The order of output is not determined. Modify the program so each rank prints its own message but the order in which they do is sorted by rank number.

Note that this will still not guarantee that the print statements appear in order on the screen. Although your program should send them in the desired order to stdout, system buffering and characteristics beyond MPI’s control will determine when each line is printed.

Your program must print exactly the same strings as the example program from the book, but in rank order.

20 points For a working program

Exercise 3: More Standard Output

The solution to the previous exercise cannot guarantee ordered output, though it will succeed most of the time. Write a function that all processes call with a string as an argument. The function should collect the string data from all processes and print it in rank order on process 0.

Your function must have the following type signature:

```
1 void print_string(char *str);
```

Remember that in C, strings are terminated by a 0 character. You can use the C library function `strlen()` to get the length of a string. Provide a file with only the function

`print_string()` in it. We will compile and link it with our test program. Your program may assume that no string will ever be longer than 1024 characters, including the terminating 0.

All nodes must call `print_string()`, but the string passed as argument can be different on each node. On process 0, print each string in rank order preceded by “[xx]:_” where xx is the rank number padded to two digits.

20 points For a working program

Exercise 4: Double Ring Program

Using the program in Example 2.3 create a new version that sends two tokens around the ring. One token should go clock-wise around and the other, simultaneously, should go counter clock-wise. Read a command line argument that specifies the number of times the tokens go around. Use non-blocking communication so that the tokens travel independently. Let each rank send whichever token arrived first to its neighbor.

Both tokens start at rank 0. The clock-wise token starts with value 0 and goes first to rank 1; just like the example program. The counter-clock-wise token starts with value 99 and goes first to rank $p - 1$.

Whenever a token is received, a message should be printed in the exact same format as the example program.

20 points For a working program

Exercise 5: MPI Latency

Write a program that measures the time a zero-length message needs to travel from one node to another. Repeat the experiment 1000 times to calculate an average. Make sure your program does not send another message until the previous one has been received.

Print the result in microseconds with three digits after the decimal point.

20 points For a working program