Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu); Class: MW 2:00-3:50pm @ FAB 40-07; Office Hr: MW 1-2pm & by appt.

Lab 6: Programming with MPI

Download and unzip the file `lab6.zip` from D2L. You'll see a `lab6` directory with some program files.

Preparation

The file `mpihosts-S16` consists of a list of available linux machines for running MPI programs. Copy it into your home directory; then set the environment variable `OMPI_MCA_orte_default_hostfile` to point to it:

```
linux> export OMPI_MCA_orte_default_hostfile='/u/<yourHomeDir>/mpihosts-S16'
```

Simple.c

Read and understand the provided program `simple.c`. It implements a pair of send-receive actions between two processes. The sender sends an integer to the receiver; the receiver decreases the value by one and sends it back. Even though the message-passing happens only between two processes, the program can run with any number of processes. It can also take a command-line argument, an integer to be used as the message value. Compile and run it. Here are some examples:

```
linux> mpicc -o simple simple.c     // compile
linux> mpirun -n 2 simple           // run with 2 procs
linux> mpirun -n 4 simple           // run with 4 procs
linux> mpirun -n 4 simple 100       // run with 4 procs, msg=100
```

FileIn.c

Read and understand the provided program `fileIn.c`. It opens a file; reads two integers from the file; and prints out their values. The input file name is provided as a command-line argument. Note that the input file is byte-encoded. To view its content, use the `od` command:

```
linux> od -i data.txt
```

1. Compile and run this program with different number of processes. Do all the processes print out the same two integers?
2. The second argument to the function `MPI_File_set_view()` controls the starting offset for the read operation. Change the current value 0 to a number that is a multiple of 4; re-compile and run the program. What do you observe now?
3. Change this parameter so that each process will read two different integers from the input file.

FileOut.c

Read and understand the provided program `fileOut.c`. This program is similar to the previous one, except that it is for writing to files.

1. Compile and run this program with different number of processes. What do you observe? How many output files are created?

2. Change this program so that there is only one output file, `out.all`; and all processes' output are written to this file, each to a different offset.

Ring.c

Now write a program `ring.c` based on this program. Instead of send-receive actions between two processes, your program will involve *all* active processes. In the program, process 0 (*i.e.* process with `rank==0`) sends an integer to process 1; upon receiving the integer, process 1 decreases its value by 1, and sends the new number to process 2; process 2 does the same thing, and sends a new number to process 3; and this action goes on. The last process in the active set sends its modified number back to process 0. Like in `simple.c`, each process should make the sending and receiving actions visible by printing out a message showing its rank, its host name, and the involved integer's value. Note that the total number of active processes is not controlled by the MPI program itself. Compile your program with `mpicc`, and test it with multiple combinations of runtime parameters.