CS 220: Introduction to Parallel Computing

Blocking vs. Non-Blocking Calls

Lecture 15

Welcome Back!

- Hope everyone had a great spring break!
- Time to remember: What is this MPI thing?!
- …

A Note: Mini office Hours

- Since several folks have class during my usual office hours, I'm adding a couple more times:
  - 3:00-3:30pm MW
- We'll see how this goes over the next couple weeks and if it works well I'll make the change permanent

Today's Agenda

- Transferring files
- I/O Buffering and Blocking

Today's Agenda

- **Transferring files**
- I/O Buffering and Blocking

Transferring Files

- Once you have ssh working, you're set!
  - If you use vim, emacs, nano, etc to edit your files on the jet machines themselves…
- We can also use ssh to copy files to the CS machines
- Once the files are on **any** CS machine, they'll be available everywhere
  - NFS

Transferring with scp

```
# Copies into my home directory:

scp local-file.txt mmalensek@stargate.cs.usfca.edu:

# Make sure you have the trailing ':' character!


# Copies and renames the file:

scp local-file.txt
        mmalensek@stargate.cs.usfca.edu:other-name.txt


# Copies to a particular folder/directory:
scp local-file.txt
        mmalensek@stargate.cs.usfca.edu:my_great_dir/subdirectory/
```

Transferring with a GUI

- scp works great, but it's not so user-friendly
- A recommendation:
  Cyberduck (https://cyberduck.io)
  - Works on Mac and Windows
  - Allows you to remote-edit files
- Another option: FileZilla
  - Available on Linux too. Watch out for crapware installers though!
- Ubuntu: Files > Connect to Server
- When you set them up, use SFTP to connect
  - sftp://stargate.cs.usfca.edu

Today's Agenda

- Transferring files
- **I/O Buffering and Blocking**

Buffering

- When calling MPI_Send, MPI may decide to **buffer** the operation
- The message contents are copied into a buffer managed by MPI
  - Kind of like doing a `strcpy(dest, src)`
- The function returns immediately!
  - In other words, nothing has been sent but your program goes on to the next line
  - This is an **asynchronous** or **buffered** send

Synchronous Send

- We are used to synchronous functions in C
  1. Call the function
  2. It does its work
  3. **Then** it finally returns
- Upside: no buffering required here
  - Reduces memory consumption
- Downside: if the next steps in our program are printing "hello world" or computing pi, do we really need to wait for the message to reach its destination?

Standard Send

- The MPI_Send we've seen is a **standard send**
- It decides whether or not the operation should be buffered
  - MPI tries to choose the option that gives best performance
- To determine this, a **cut off** size is used
  - Message less than the cut off? Buffer it
  - Too big? Send it synchronously

Receiving Data

- MPI_Recv is considered a **blocking** call
- When you use MPI_Recv, it will wait until data arrives before doing anything
- This is kind of like our programs that use **scanf**
  - The function waits until we actually type a line before it resumes execution

Monitoring Blocked Processes

- We can see what processes are doing on our machine with the **top** command
- On Linux, we have a nice status column:
  - D uninterruptible sleep
  - R running
  - S sleeping (in the **blocked** state)
  - T stopped
  - Z zombie