College of Engineering

School of Electrical Engineering & Computer Science
CS311 – Operating Systems I
R. Jesse Chaney – chaneyr@eecs.orst.edu

IPC using Message Queues, Introduction

Introduction to IPC using Message Queues

- Unix pipes and FIFOs – brief review.
- Message Queues – What are they?
- How are Message Queues like Pipes?
- How are Message Queues different from Pipes?
- What other kinds of Message Queues are there?

- How are we going to study Message Queues?

1

CS311 – Message Queues
R. Jesse Chaney – chaneyr@eecs.orst.edu

Oregon State UNIVERSITY

- There are many different IPC implementation out there that are called “Message Queues”.
    - You might here message broker, message systems, or mail boxes.

- The focus for this class for message queue is on kernel message queues, System V messages queues or POSIX message queues.

- Message queues are like pipes, but different. We’ll look at how they are alike and different.

- I’ll mention some of the other implementations of message queues, but only briefly.

- Lastly in this lecture, we’ll see how we will study message queues.

Pipes and FIFOs – brief review

Pipes and FIFOs are covered in TLIP, chapter 44.

- Pipes are the oldest form of IPC in Unix
- A pipe is unidirectional
- Pipes are bytes streams, there is no concept of boundaries for the data written to a pipe
- Data passes through a pipe sequentially.
- Writes on a pipe *can* be done atomically.
- Pipes have a limited capacity.
- Pipes can only be used between *related* processes
- FIFOs can be used between *unrelated* processes

CS311 – Message Queues
R. Jesse Chaney – chaneyr@eecs.orst.edu

Oregon State UNIVERSITY

Here are a few of the highlights for pipes for Unix IPC.

- Pipes have been about since the 70's, the 1970's.
- Pipes are commonly used on the command line to chain processes together.
- A pipe is unidirectional. In programming, when you create a pipe, you create 2, a read side and a write side. Each side is one or the other, within a process.
- A pipe is a simple byte stream. There are no boundaries imposed by the operating system between chunks of data beyond that of a simple byte.
  - If you write a string into a pipe, the process on the other end has to know how to read the data as a string to make sense of it.
- If a programmer is not careful, it is possible to mix data together when a pipe has multiple writers.
- Data passes through a pipe sequentially. You cannot seek forward in a pipe. The only way to get to the 100th byte in a pipe is to read the 99 bytes ahead of it.
- All data are of equal importance in a pipe. You cannot have data moved to the front of the pipe based on a measure of value of the data.
- While writes below PIPE_BUF are guaranteed to be atomic, it is difficult to reassemble larger chunks of data when you have multiple writers and even worse if you have both multiple writers and multiple readers.
- Writes done to a pipe will block when the capacity of the pipe has been reached.
- Reads done on an empty pipe block until data are available to pull from the pipe.
- Pipes (not FIFOs) are limited to being used with *related* processes.
  - Related processes (from TLPI page 894) are processes that have a common ancestor through a series of fork() calls.
- A FIFO (aka a named pipe) can be used between unrelated processes. A FIFO exists as an entry in the file system.

Message Queues – What are they?

Message queues are covered in TLPI, chapter 46 (SysV message queues) and chapter 52 (POSIX message queues).

- A message queue is a form of IPC that allows processes (or threads) to communicate in complete units of data, beyond simple byte streams.
- Reads and writes of messages are atomic.
- A message has a *type* field that allows a read to occur from a location in the queue other than the message at front of the queue.

- Boundaries between messages are preserved.
  - This is an important distinguishing difference between pipes and message queues.
- A complete message can be a structure within your program.
- A message will either be completely written into the queue or it will completely fail.
- A message will either be completely read from the queue or it will completely fail and remain in the queue.
- It is not possible to read part of a message.
- Because each message has a type field, it is possible to read messages out of order from a queue. This is a good thing.
  - It is like cutting in line, which for lines is not normally a good thing, but is okay for message queues.
- Message queues are also sometimes called mailboxes.
  - It is a complete chunk of data.

How are Message Queues like Pipes?

- Both pipes and message queues are a form of IPC between multiple processes or threads.
- Only one process can receive the next bytes/message.
- Both pipes and message queues allow processes to be developed in different languages.

- You could use both pipes and message queues to communicate within a single process, but there are better ways to handle that communication.

- Multiple processes can read from a pipe and multiple processes can read from a queue, but only 1 process will receive the "next" data.
    - The only way for multiple processes to receive the same data is to have either multiple pipes/queues on which the same data are sent or to have processes forward data to other processes.

- So long as the data API is carefully written, you can have processes developed in different languages inter-operate on the data sent on pipes of message queues.
    - You could have a Python program read data from a file, pass it to a FORTRAN program over a pipe to process, which passes it to a C program over another pipe for visualization.
    - The same is true for message queues.

- Both pipes and message queues have calls to do blocking and non-blocking reads and writes.

When ever message queues are initially described, they are said to be like pipes, but with preserved data boundaries.

We all like to have our boundaries.

How are Message Queues different from Pipes?

- Message queues act on chunks of data beyond simple bytes.
- Message queues can have a lifetime beyond the life of the creating process.
- The functions of writing to and reading from a pipe are like writing to and reading from a file. Writing to and reading from a message queue is different.
- Pipes are always read in the order in which data was written into the pipe. Messages in a message queue can be read out of order.
- Having multiple writers and multiple readers on a pipe is going to be really messy. With message queues, it is easy.
- There are more resource limits on message queues than on pipes.

Oregon State UNIVERSITY

- I've mentioned the data boundary feature of message queues several times already, but it happens to be something I really like about them.
- Message queues, once created, can outlive the lifetime of the process that created them.
- Process that use message queues do not need to have a common fork() ancestor.
  - FIFOs share this feature.
- Writing to a pipe or reading from one has the same form as writing to and reading from a file. Pipes use write() and read(). Message queues are different.
  - We'll go over the specific message queue calls in other lectures.
- The data written to a pipe is read out in the same order. Data in a message queue can be read out of order.
  - We'll review how data are read out of order in another lecture.
- Due to the lack of boundaries for data in pipes, it will be very difficult to implement pipe connected processes where there are multiple readers and multiple writers on a pipe.
- There are very few kernel or system related resource limits placed on pipes. There are quite a few placed on SysV message queues and POSIX message queues. We'll cover some of those as we review each one.

What other kinds of Message Queues are there?

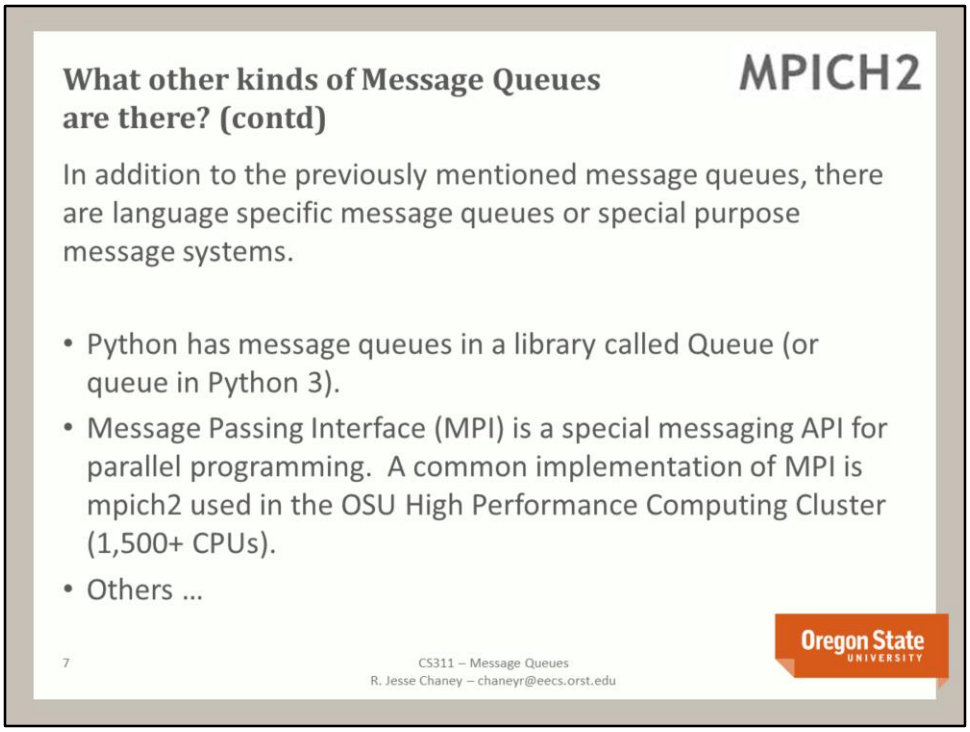
There are many other different kinds of and implementations of message queues. Some popular ones are:

- Java Message Service (JMS)
- ActiveMQ from Apache
- RabitMQ by Pivotal
- MSMQ by Microsoft
- WebSphereMQ by IBM
- HornetQ by JBoss
- And others … others … others …

Oregon State UNIVERSITY

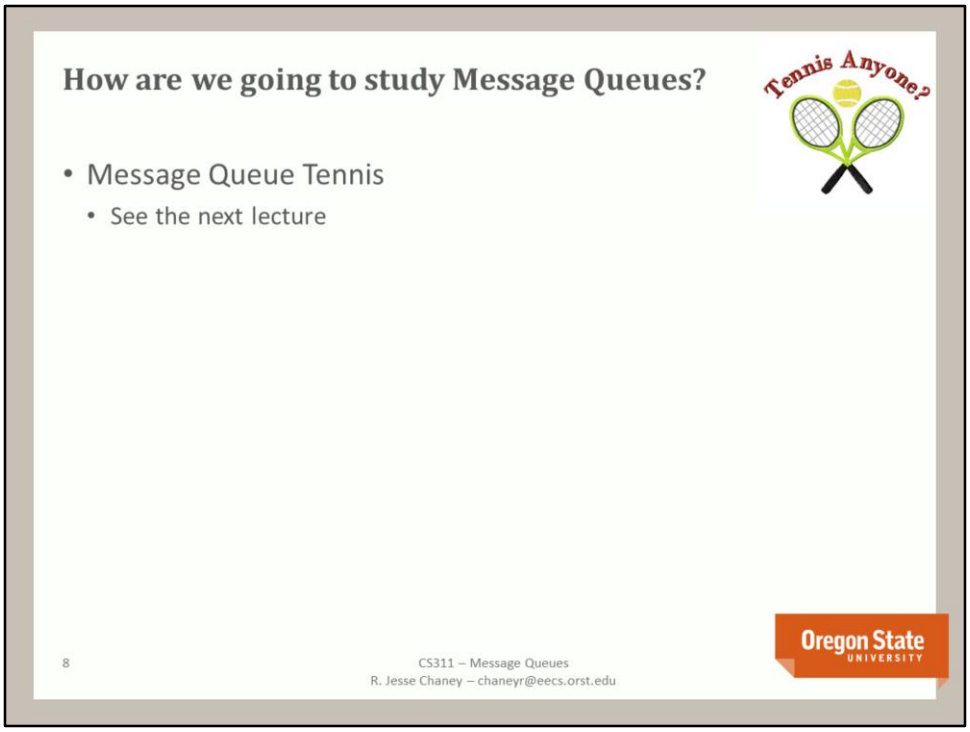
- One of the things that distinguishes the message queue system listed here is that they are designed to span multiple systems.
  - Both SysV message queues and POSIX message queues exist only on the system where they were created.
  - The ones above allow messages to be sent and received over a network.

- Some of the above systems provide an additional message semantic of a message **topic**.
  - Multiple processes can subscribe to a message topic, just like a message queue.
  - The difference with a message topic is that when a message is sent to a topic, all subscribers receive the message, not just one.
  - This is a sort of a message broadcast. The sender of a message to a topic does not know how many (if any) processes are subscribed to the topic.
  - This is often called a publish-subscribe or more simply, pub-sub paradigm.



When used gingerly, the message queues in Python work well in either a multi-threaded Python application or a multi-process Python application.

MPI is probably not really a message queue system, but I just could not leave it out.



- I've always hated it when a new concept is introduced and only a very simple example is provided to demonstrate it.
- The tennis example is complex enough that it shows how message queues can be used, but (hopefully) simple enough to easily understand and use.
- The C source is about 900 lines.
- We'll go over the message queue tennis application quite a bit in following lectures.