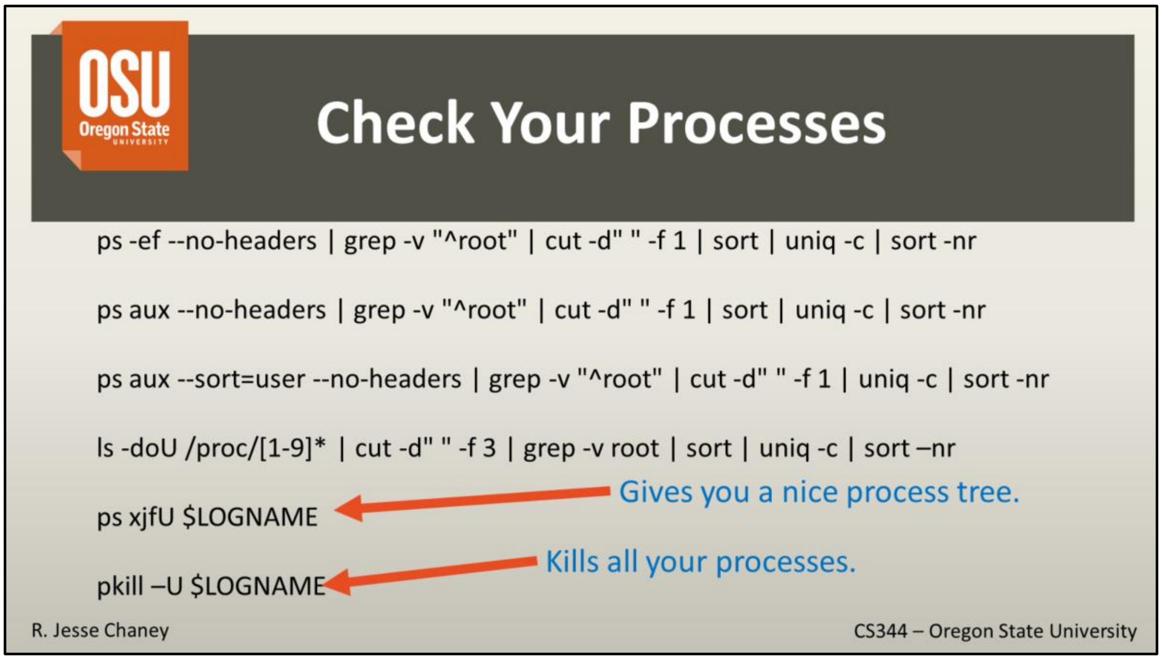


Check Your Processes

```
ps -ef --no-headers | grep -v "^root" | cut -d" " -f 1 | sort | uniq -c | sort -nr

ps aux --no-headers | grep -v "^root" | cut -d" " -f 1 | sort | uniq -c | sort -nr

ps aux --sort=user --no-headers | grep -v "^root" | cut -d" " -f 1 | uniq -c | sort -nr

ls -doU /proc/[1-9]* | cut -d" " -f 3 | grep -v root | sort | uniq -c | sort –nr

ps xjfU $LOGNAME

pkill –U $LOGNAME
```

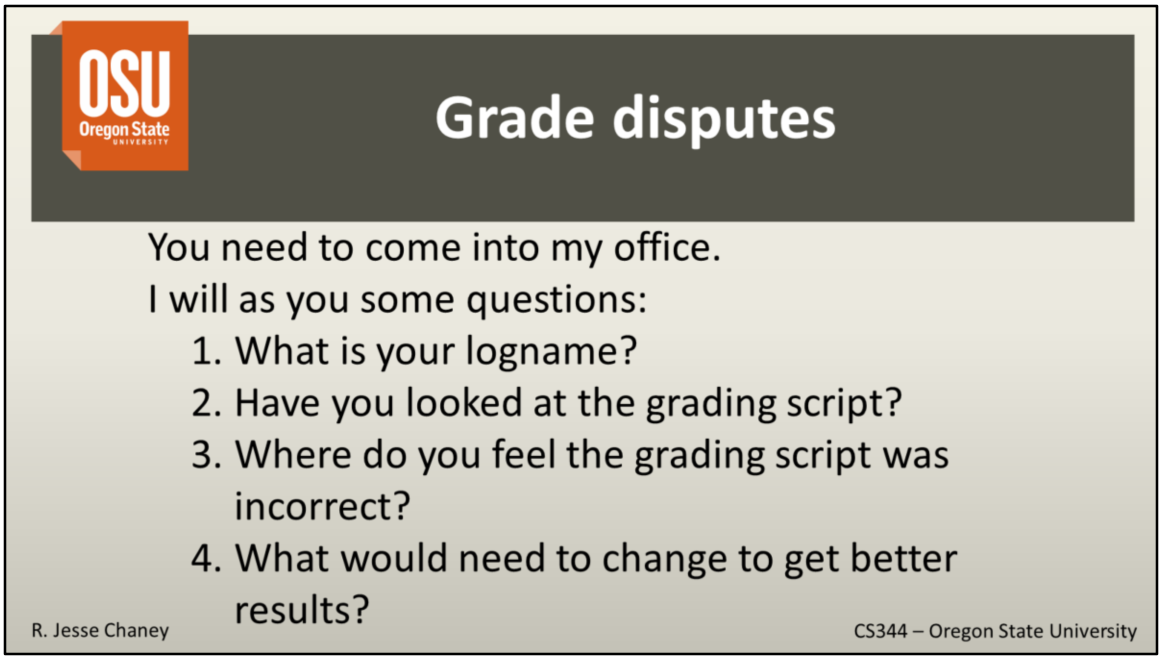
Gives you a nice process tree.

Kills all your processes.

R. Jesse Chaney

CS344 – Oregon State University

Some folks have been sloppy about creating bunches of processes and neglecting to kill them off. The first 4 of these commands will show you how many processes each user has running. If you see yourself in the list with more than 25-30 processes, kill them. If you leave fifo_server processes running and have not deleted the "FifoServer__" fifo in your home directory, you are probably creating problems for yourself. Old fifo_server processes may still be attached to it and could steal the messages your fifo_client processes are writing to it.



Grade disputes

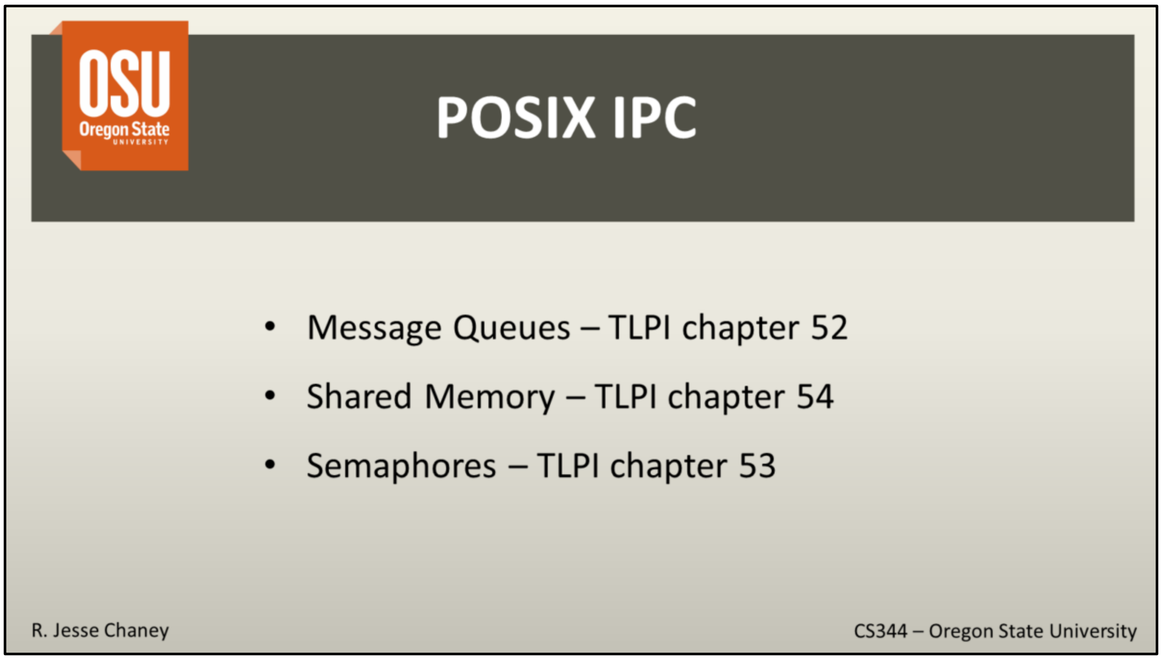
You need to come into my office.

I will as you some questions:

1. What is your logname?
2. Have you looked at the grading script?
3. Where do you feel the grading script was incorrect?
4. What would need to change to get better results?

R. Jesse Chaney

CS344 – Oregon State University



POSIX IPC

- Message Queues – TLPI chapter 52
- Shared Memory – TLPI chapter 54
- Semaphores – TLPI chapter 53

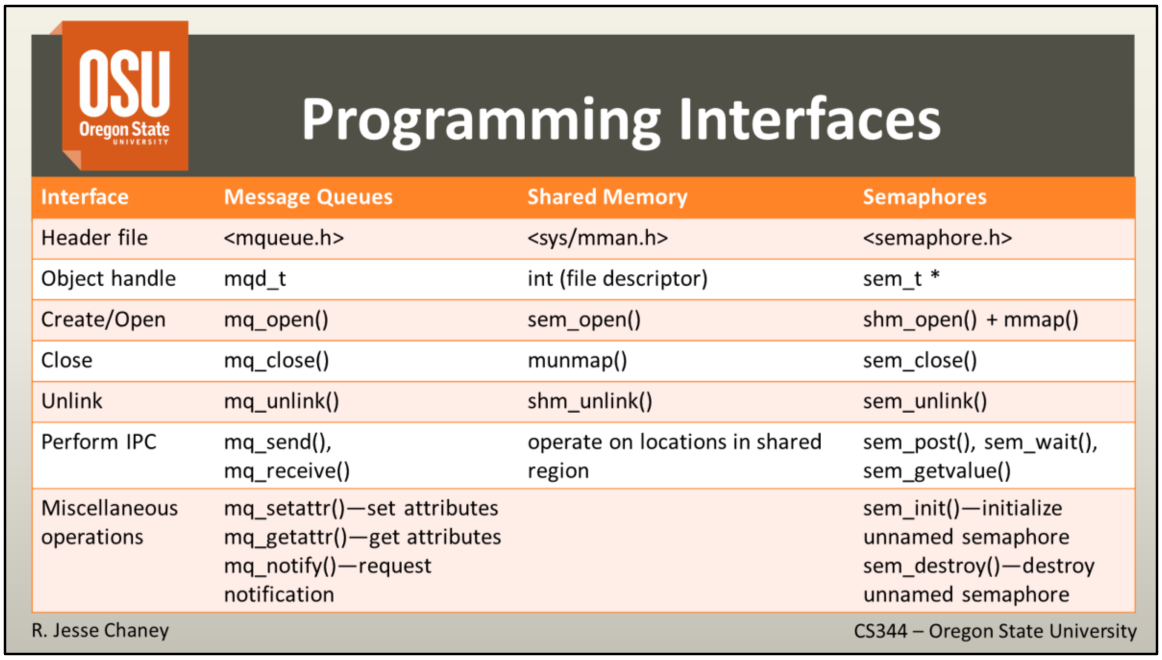
R. Jesse Chaney

CS344 – Oregon State University

I'm going to go out of order from how I have it in the class schedule.

- Message queues can be used to pass messages between processes. Message boundaries are preserved, so that readers and writers communicate in units of messages.
- Shared memory enables multiple processes to share the same region of memory. Once one process has updated the shared memory, the change is immediately visible to other processes sharing the same region.
- Semaphores permit multiple processes to synchronize their actions. These are a lot like the mutex-es we looked with threads, but more general.

The books goes through a lot of the differences between the POSIX and System V IPC. We are really only interested in the POSIX IPC (unless there happens to be some extra credit on the next assignment that deals with the System V calls).



Programming Interfaces

Interface	Message Queues	Shared Memory	Semaphores
Header file	<mqueue.h>	<sys/mman.h>	<semaphore.h>
Object handle	mqd_t	int (file descriptor)	sem_t *
Create/Open	mq_open()	sem_open()	shm_open() + mmap()
Close	mq_close()	munmap()	sem_close()
Unlink	mq_unlink()	shm_unlink()	sem_unlink()
Perform IPC	mq_send(), mq_receive()	operate on locations in shared region	sem_post(), sem_wait(), sem_getvalue()
Miscellaneous operations	mq_setattr()—set attributes mq_getattr()—get attributes mq_notify()—request notification		sem_init()—initialize unnamed semaphore sem_destroy()—destroy unnamed semaphore

R. Jesse Chaney

CS344 – Oregon State University

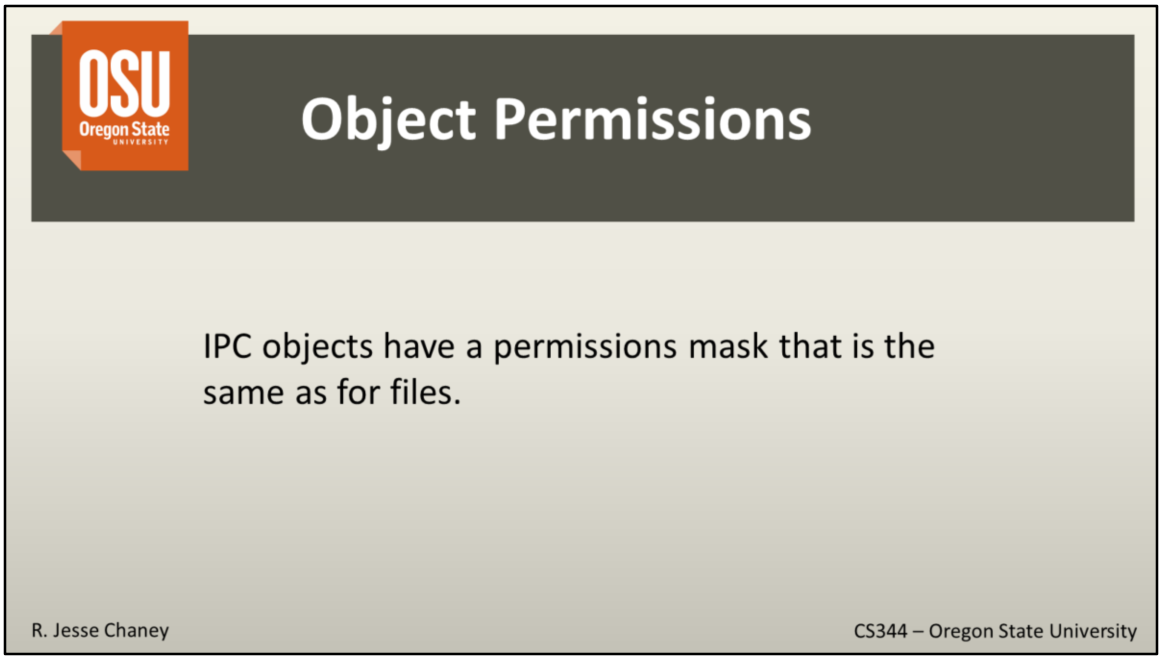
the IPC open call either:

- creates a new object with the given name, opens that object, and returns a handle for it; or
- opens an existing object and returns a handle for that object.

The handle is analogous to the file descriptor returned by the open() file call.

An IPC close call that indicates that the calling process has finished using the object and the system may de-allocate any resources that were associated with the object for this process.

IPC objects are automatically closed if the process terminates or performs an exec().



Object Permissions

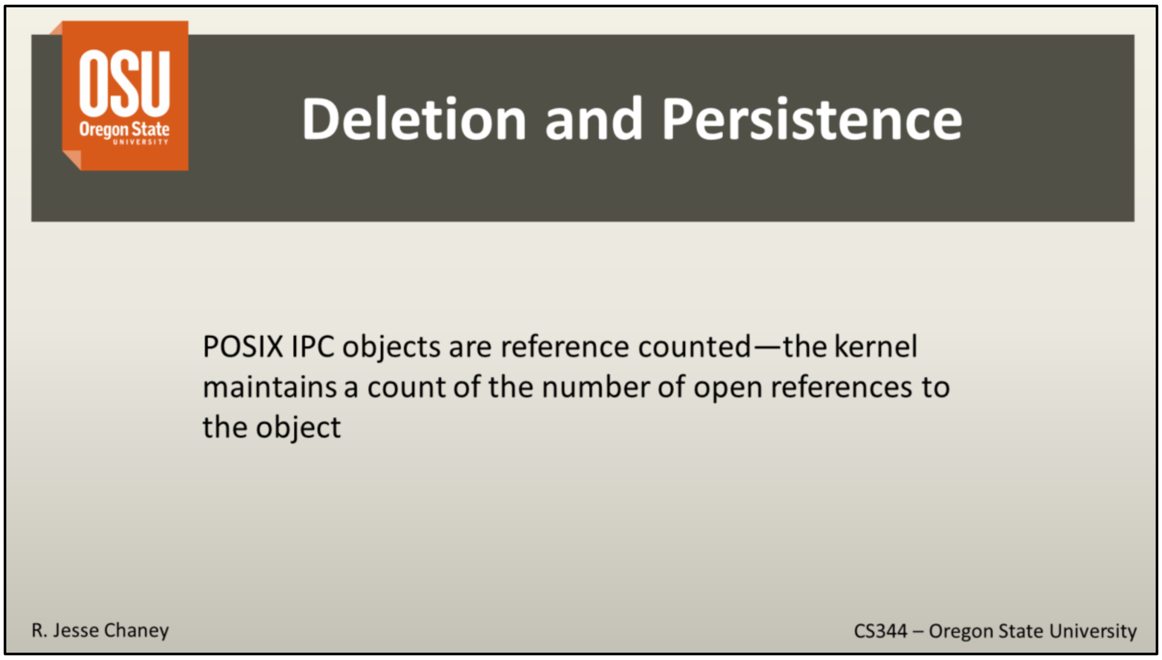
IPC objects have a permissions mask that is the same as for files.

R. Jesse Chaney

CS344 – Oregon State University

Permissions for accessing an IPC object are similar to those for accessing files, except that execute permission has no meaning for POSIX IPC objects.

You can allow sharing of the objects within a group of for all. It is up to you.



Deletion and Persistence

POSIX IPC objects are reference counted—the kernel maintains a count of the number of open references to the object

R. Jesse Chaney

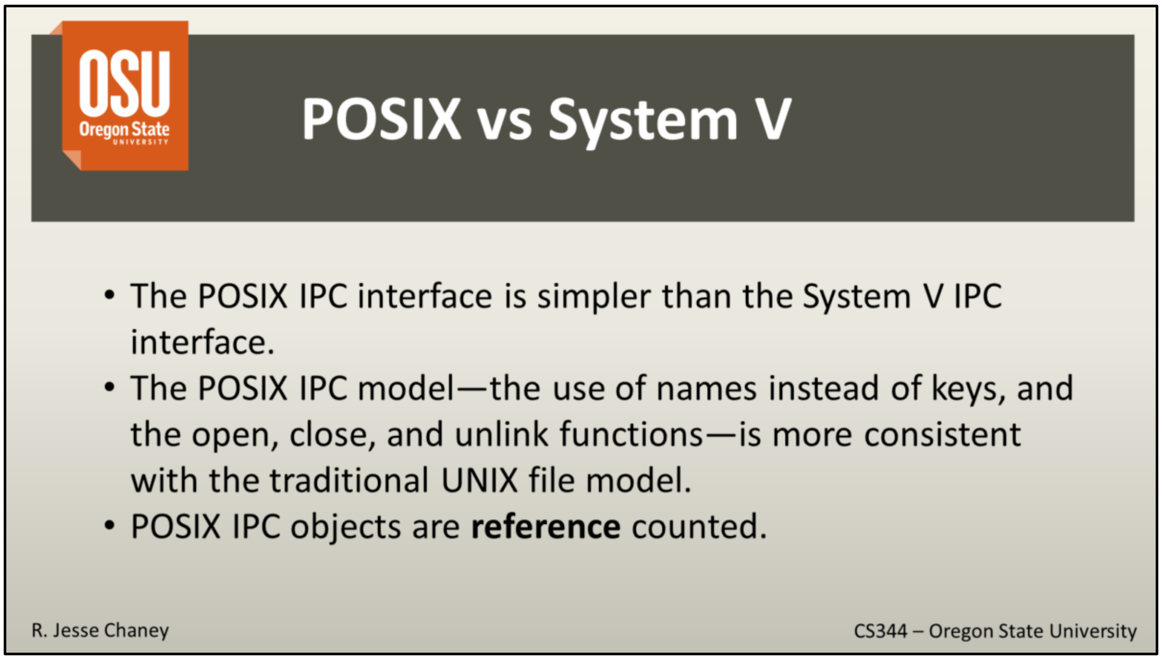
CS344 – Oregon State University

As with open files, POSIX IPC objects are reference counted—the kernel maintains a count of the number of open references to the object. This is the same thing we saw when we looked at files and noticed the count of the number of hard links. For a file, when the number of hard links drops to zero, the space is reclaimed by the file system.

Each IPC object has a corresponding unlink call whose operation is analogous to the traditional unlink() system call for files. The unlink call immediately removes the object's name, and then destroys the object **once all processes cease using** it (i.e., when the reference count falls to zero).

For message queues and semaphores, this means that the object is destroyed after all processes have closed the object; for shared memory, destruction occurs after all processes have unmapped the object using munmap().

When using message queues or shared memory, it is to your advantage to de-allocate and remove objects when you are done with them. It causes a lot of unnecessary grief if you keep finding stray messages in the message queue.



POSIX vs System V

- The POSIX IPC interface is simpler than the System V IPC interface.
- The POSIX IPC model—the use of names instead of keys, and the open, close, and unlink functions—is more consistent with the traditional UNIX file model.
- POSIX IPC objects are **reference** counted.

R. Jesse Chaney

CS344 – Oregon State University

However,

System V IPC is specified in SUSv3 and supported on nearly every UNIX implementation.

Despite the SUSv3 specification for POSIX IPC object names, the various implementations follow different conventions for naming IPC objects.

Various details of POSIX IPC are not specified in SUSv3. In particular, no commands are specified for displaying and deleting the IPC objects that exist on a system.