CSC425 – Project 3 – Shared Memory

This project will implement the file transfer program from project 1 using shared memory concepts such as message queues and shared memory regions instead of sockets. This code will transfer data between processes running on the same server. As such, there is no need to be concerned with IP addresses and sockets.

File transfer

The project will implement two programs - a shm_client and a shm_server. The client will request a file from the server using shared memory techniques. The server will create a command channel utilizing a POSIX Message Queue. The client will create a POSIX Shared Memory segment for the actual file transfer.

System Flow

- The server will create a POSIX message queue to be used as the command channel.
- The client will subscribe to the message queue.
- The client will create a POSIX shared memory region.
- The client will use the command channel to request a file from the server. The client will provide the address and size of the shared memory region as part of the request. As part of your design, you may want to provide other information as well.
- The server will map to the shared memory region
- The server will open the file and put a "chunk" of file data into the shared memory region.
- The client will retrieve the file data from shared memory and write it to a file.
- The server will retrieve and send the next "chunk" of the file, the client will receive over and over until transfer is complete.
- When all file requests have completed, both the client and server will shutdown gracefully.

Rules

- You must use a POSIX message queue and shared memory implementation (Cannot use System V versions.)
- You will need to implement that semaphores as the appropriate synchronization construct to protect shared memory.
- Shared memory buffer size may not exceed 4096 bytes.
- It is acceptable to hardcode the name of the message queue in your client code. This name will be assigned by the server when it creates the queue.
- The client should be flexible enough to start prior to the server. This means that it should attempt to connect to the message queue and if it isn't there, wait a period of time and try again.
- If a file is requested that doesn't exist on the server, the server will return a FNF message in shared memory.
- File list - Same files that were used in the previous project.
- The client will initialize, in a loop request multiple files from the server, then shut down gracefully.

- Client and server will need a mechanism to know when the client is done in order to shut down the message queue and the shared memory segment.
- Both client and server will need to properly clean up allocated memory before exiting.

Resources

The Linux Programming Interface – Michael Kerrisk (Chapters 51 to 54)

man7.org - https://man7.org/linux/man-pages/man7/shm_overview.7.html

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

Things to think about

1. How do I connect to the message queue on both ends and how do I wait for it to be created?
2. What kind of structure to I need to implement in shared memory to handle data transfer (amount sent, total file size, memory buffer, etc.)
3. How do I handle completion?
4. How can I break the project into smaller pieces and attack in an orderly fashion?