Mohamed KhalifaSection 1[mkhalifa@csu.fullerton.edu](mailto:mkhalifa@csu.fullerton.edu)

Mark Perez [perezmd3129@csu.fullerton.edu](mailto:perezmd3129@csu.fullerton.edu)

Conrad Chu [chuconrad1@gmail.com](mailto:chuconrad1@gmail.com)

Ahsan [ahsanamin90@csu.fullerton.edu](mailto:ahsanamin90@csu.fullerton.edu)

Design Document for "Sender & Receiver" Program

sender.cpp

The Main module connects to the shared memory and the message queue using the init function. It then sends the name of the file using ‘sendFileName(argv[1])’. It then sends the file with ‘sendFile(argv[1])’. Lastly, it cleans up the parameters using ‘cleanUp(shmid, msqid, sharedMemPtr)’.

Module

Main(int argc, char\*\* argv)

sendFileName(argv[1])

sendFile(argv[1])

cleanUp( shmid, msqid, sharedMemPtr)

init(shmid, msqid, sharedMemPtr)

Sub-Module

The init creates a file called keyfile.txt containing the string "Hello world". Using ftok("keyfile.txt", 'a'), it generates a key. Using the same key for the queue and the shared memory segment it allocates a shared memory segment using ‘shmget’. With the id and shared memory segment, it attaches to the shared memory using ‘shmget’. init then attaches to the message queue using ‘shmat’. Lastly, it stores the IDs and the pointer to the shared memory region in the corresponding function parameters.

Sub-Module

perror(“msgget”)

init(int& shmid, int& msqid, void\*& sharedMemPtr)

perror(“shmget”)

fclose(keyfile)

perror(“ftok”)

perror(“fopen”)

fopen(“keyfile.txt”)

exit(-1)

perror(“shmat”)

exit(-1)

exit(-1)

exit(-1)

exit(-1)

sendFileName is used to send the name of the file to the receiver. After making sure the file name does not exceed the maximum buffer size, it creates an instance of the struct representing the message containing the name of the file. It then sends the message and confirms we ‘msgsnd’ was used.

Sub-Module

sendFileName(const char\* fileName)

strncpy(msg.fileName, filename, MAX\_FILE\_NAME\_SIZE)

perror(“msgsnd”)

exit(-1)

The sendFile is the main send function. It opens the file using “fopen(filename, ‘r’)” and confirms the file was opened using ‘perror(“fopen”)’. While reading through the file, if the size of the shared memory chunk is successfully read and stored, it confirms using ‘perror(“fread”)’. If the message was successfully sent, it is confirmed with ‘perror(“msgsnd”)’. If the message was successfully received and acknowledged, it is confirmed using ‘perror(“msgrec”)’. If the file has been read completely, it sends a message saying there is nothing left to send and confirms using ‘perror(“msgsnd”)’ and closes the file fclose(fp).

Sub-Module

fclose(fp)

sendFile(const char\* fileName)

perror(“msgsnd”)

perror(“fopen”)

fopen(fileName, “r”)

exit(-1)

exit(-1)

perror(“msgsnd”)

perror(“fread”)

exit(-1)

perror(“msgrcv”)

exit(-1)

exit(-1)

The cleanUp function clears up the parameters by detaching from the shared memory and deallocating the shared memory and message queue. It then confirms using ‘perror(“shmdt”)’, ‘perror(“shmctl”)’ and ‘perror(“msgctl”)’.

Sub-Module

perror(“shmdt”)

exit(-1)

cleanUp(const int& shmid, const int& msqid, void\*& sharedMemPtr)

perror(“msgctl”)

exit(-1)

perror(“shmctl”)

exit(-1)

Recv.cpp

The Main function in the recv file connects to the shared memory and the message queue using the init function. It then receives the file name from the sender using ‘recvFileName()’. It then goes to the main loop using ‘mainLoop(file.c\_str())’. Lastly, it cleans up the parameters using ‘cleanUp(shmid, msqid, sharedMemPtr)’.

Module

Main(int argc, char\*\* argv)

mainLoop(fileName.c\_str())

cleanUp( shmid, msqid, sharedMemPtr)

recvFileName()

init(shmid, msqid, sharedMemPtr)

The init creates a file called “keyfile.txt” containing the string "Hello world" using ofstream and closes it with ‘outfile.close()’. It then uses ftok("keyfile.txt", 'a') to generates a key. Using the same key for the queue and the shared memory segment it allocates a shared memory segment using “shmget(‘a’, SHARED\_MEMORY\_CHUNK\_SIZE, S\_IRUSR | S\_IWUSR)”. It then creates a message queue with the id and shared memory segment using “msgget(‘a’, S\_IRUSR | S\_IWUSR | IPC\_CREAT)”. Lastly, it stores the IDs and the pointer to the shared memory region in the corresponding function parameters using ‘msgrcv(msqid, & message\_buffer, sizeof(message\_buffer) – sizeof(long), 2, 0)’.

Sub-Module

Msgrcv(msqid, &message\_buffer, sizeof(message\_buffer ) – sizeof(long), 2, 0)

ofstream outfile (“keyfile.txt”)

init(int& shmid, int& msqid, void\*& sharedMemPtr)

outfile.close()

ftok(“keyfile.txt”, ‘a’)

Shmat(‘a’, S\_IRUSR | SIWUSR | IPC\_CREAT)

shmget(‘a’, SHARED\_MEMORY\_CHUNK\_SIZE, S\_IRUSR | S\_IWUSR)

The recvFileName receives the file name using “msgget(‘a’, S\_IRUSR | S\_IWUSR)”.

Sub-Module

recvFileName()

msgget(‘a’, S\_IRUSR | S\_IWUSR)

The mainLoop function opens the file for writing with “\_recv” appended to the end of the filename using the ‘fopen(recvFileNameStr.c\_str(), “w”)’. If it opens mainLoop notifies the user using ‘perror(“fopen”)’. If the size field of the message is not 0, it copies that many bytes from the shared memory segment to the file. Otherwise, if 0, then we close the file and exit. If the sender is not done sending the message, save the shared memory to file and confirm with ‘perror(“fwrite”)’, and if it is ready for the next set of bytes, confirm with ‘perror(“msgsnd”)’.

Sub-Module

fclose(fp)

mainLoop(const char\* fileName)

fopen(recvFileNameStr.c\_str(), “w”)

exit(-1)

perror(“msgsnd”)

perror(“fwrite”)

exit(-1)

exit(-1)

perror(“fopen”)

The cleanUp function clears up the parameters by detaching from the shared memory and deallocating the shared memory and message queue. It then confirms using ‘perror(“shmdt”)’, ‘perror(“shmctl”)’ and ‘perror(“msgctl”)’.

Sub-Module

cleanUp(const int& shmid, const int& msqid, void\* sharedMemPtr)

perror(“msgctl”)

exit(-1)

perror(“shmctl”)

exit(-1)

perror(“shmdt”)

exit(-1)