**Design of Assignment1**

Process\_sync

Angel Jimenez

Tyler James McConnell

Miguel Velazquez

March 18, 2016

Version 1.0

# 

# **Table of Contents**

Introduction 2

Design Overview 2

System Architecture 3

Interaction 4

**Introduction**

The purpose of this document is to describe the implementation of “process\_sync” as described by the Programming Assignment 1 handout. This involves two processes, sender and recv, which synchronously transfer files between each other.

**Design Overview**

Description

Use shared memory and message queues to share data between two processes.

sender.cpp

Implementation of the process that sends to the receiver process:

1. Attach to shared memory
2. Connect to message queue
3. Read from message queue
4. Send to receiver a message indicating how many bytes were read
5. Wait for receiver to receive message and confirm
6. Repeat steps 3 to 5 until the whole message is read
7. When the file is reached, tell receiver there is no more to send
8. Close file and detach shared memory

recv.cpp

Implementation of the process that receives from the sender process:

1. Attach to shared memory
2. Connect to message queue
3. Wait in message queue to receive a message from sender.cpp
4. If the size of the message is not equal to zero, the receiver reads the size number of bytes from the shared memory and saves it to “recvfile”
5. Send a message to sender letting it know it received a message successfully
6. Repeat steps 3 to 5 until the size of the incoming message is 0
7. Before closing the file, detach and de-allocate the shared memory and message queue

msg.h

Implementation of the structure to be used and shared between both sender and recv:

1. Contains macros SENDER\_DATA\_TYPE 1 & RECV\_DONE\_TYPE 2
2. Structure will contain a long mtype to distinguish it as a send or receive message
3. Structure will contain an int size to hold the size of bytes in the message
4. Function to print the structure will also be included

**System Architecture**

The following is a Data Flow Diagram (DFD) indicating the movement of data throughout the system:

**sender.cpp:**

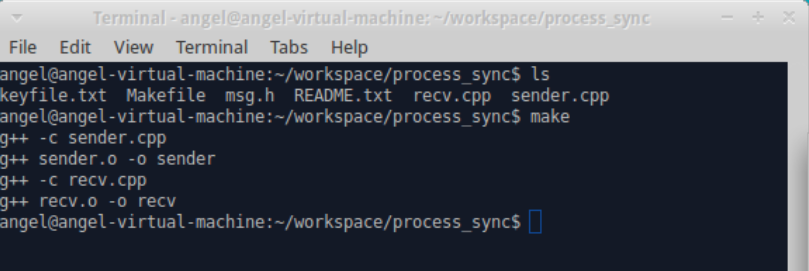


**recv.cpp:**

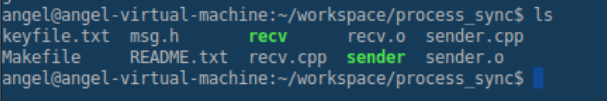
**Interaction**

“process\_sync” uses the Command Line Interface (CLI) to interact with the user. After using “make” to compile the files into executables on a Unix based/ Linux Operating system, the user can run the file “sender” with the parameter “keyfile.txt” to execute the sender process. Likewise, the user can execute the “recv” file in a separate terminal window. The CLI will display prompts to the user indicating the status of each processes execution. Below are some screen shots of an example:

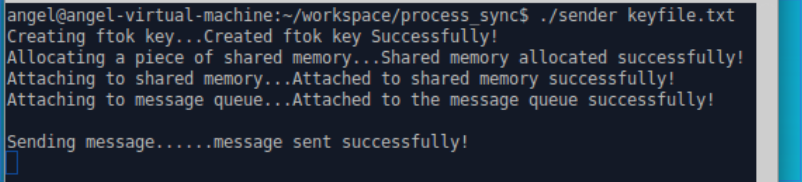
After extracting the .tar file, use “make” to compile the code:



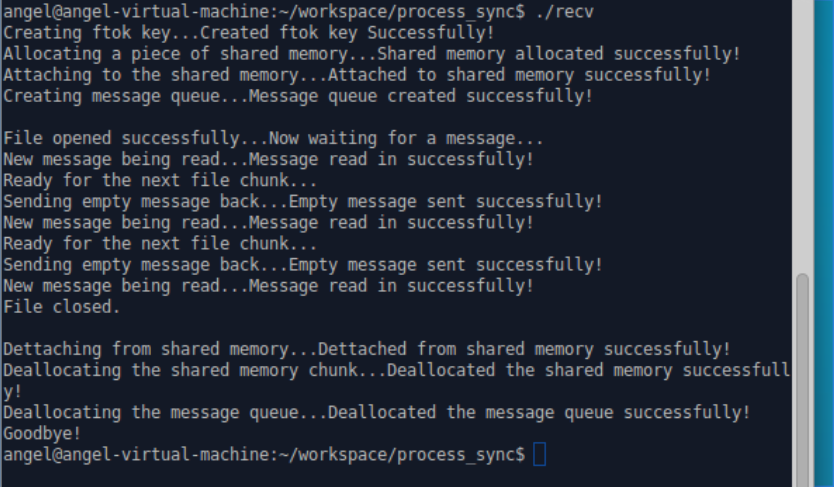
“ls” will let you see if the executables are available:



“./sender keyfile.txt” will run the sender process and wait for “recv” to respond:



“./recv” will run the recv process and respond to sender process:



sender process will finish reading and sending the message and notify recv that it is finished to end the processes:

