CSCI 610  
 Advanced concepts in operating systems

Project 4- Group

Danile Fabres, Site Yin, Sandip Gautam, Swapnil Acharya

StarId: ea4963aw Due: 04/13/2020

Purpose

The objective of this project is to implement several distributed mutual exclusion algorithms and use each to solve a specified synchronization problem

**Organization of this Document**

1. Man Page Documentation
2. Centralized Algorithm Code
3. Distributed Algorithm Code
4. Centralized Algorithm Output
5. Distributed Algorithm Output

**Man Page Documentation**

Centralized algorithm: using centralized algorithm to solve the dining philosophers

problem, a process must used to represent each dining philosopher, so there is must be a total

of six processes. One of the process is elected as coordinator. When a philosopher wants to

access the shared chopsticks, the philosopher sends a request message to the coordinator

asking for the permission. If no other philosopher is using the requested chopsticks, the

coordinator will send back a granting permission and the philosopher can use that after

granting permission arrived. However, if some other philosopher is using the requested

chopsticks, the coordinator will refrain from sending granting permission and the philosopher

will be added to waiting list. When philosopher is finished with using of shared chopsticks, it

will send a message to the coordinator to release its exclusive access and the coordinator will

give the first philosopher in the waiting queue a granting permission.

Distributed algorithm: using distributed algorithm to solve the dining philosophers

problem, a process must used to represent each dining philosopher, so there is must be a total

of five processes. When a philosopher wants to access the shared chopsticks to eat, the

philosopher sends a request message including the process number, chopsticks id and current

time, to the all other processes asking for the permission. When a philosopher receives a

request message, If the philosopher is not using the requested chopsticks and does not want to

access it, the philosopher will send back an available permission. If the philosopher is using the

requested chopsticks, the receiver philosopher will refrain from sending available permission

and the sender philosopher will be added to waiting list. However, if the receiver philosopher

also wants to access the shared chopsticks, it compares the two timestamps. If sender

philosopher has a lower timestamp, the receiver philosopher sends an available message back.

If receiver philosopher has lower timestamp, the sender philosopher will be added to waiting

queue.

After sending a request message, the philosopher waits until everyone gives a

permission. The philosopher can start eating as soon as all permission received. When

philosopher is finished with using of shared chopsticks, it will send a message to all processes in

waiting queue to release its exclusive.

The centralized algorithm is easier to design and implement. With one of the process is

elected as coordinator to update the shared chopstick, request philosopher won’t need to wait

for four permission messages to start eating. The requester only needs to listen to coordinator’s

message. However, the distributed algorithm is more complicated since the more messages.

A cpp file “StartClient.cpp” was created to automate the creation of clients. This file uses fork() and execl() to start 5 client processes.

**To compile and execute Centralized the project, do the following steps:**

1. Open coursework folder for CSCI 610 class and then open student work folder
2. Inside Student work folder, open ea4963aw folder
3. Inside ea4963aw, open Project4 folder
4. Inside ea4963aw, open Centralized folder
5. Clean previous executables using Linux terminal command: make clean
6. Compile the code in a Linux terminal using command: make
7. Start Centralized sever by entering following command in terminal: ./serverCentralized localhost &
8. Start Centralized clients by entering the following command in terminal: ./sc 5
9. Every client/philosopher runs (does eat&think process) for a total of 15 counts and then terminates itself.
10. Once all clients terminate, enter the following command in terminal to see all running processes: ps
11. Use the following command followed by the process id to kill all unterminated processes: kill
12. Clean the executables by using command: make clean

**To compile and execute Distributed the project, do the following steps:**

1. Open coursework folder for CSCI 610 class and then open student work folder
2. Inside Student work folder, open ea4963aw folder
3. Inside ea4963aw, open Project4 folder
4. Inside ea4963aw, open Distributed folder
5. Clean previous executables using Linux terminal command: make clean
6. Compile the code in a Linux terminal using command: make
7. Start Centralized sever by entering following command in terminal: ./distributed\_broadcaster localhost &
8. Start Centralized clients by entering the following command in terminal: ./sc 5
9. Once all clients terminate, enter the following command in terminal to see all running processes: ps
10. Use the following command followed by the process id to kill all unterminated processes: kill
11. Clean the executables by using command: make clean

**Centralized Algorithm Code**

**Makefile**

all: serverCentralized clientCentralized sc

serverCentralized: CentralizedServer.cpp Philosopher.h

g++ -std=c++0x -pthread CentralizedServer.cpp Philosopher.h -o serverCentralized

clientCentralized: CentralizedClient.cpp Philosopher.h

g++ CentralizedClient.cpp Philosopher.h -o clientCentralized

sc: StartClient.cpp

g++ StartClient.cpp -o sc

clean:

rm -f serverCentralized

rm -f clientCentralized

rm -f sc

rm -f output.txt

**Philosopher.cpp**

//This is a class descigner to simulate a Philosopher

//which can store integers like id, client socket descriptor(fd), left adn right chopstick index

#ifndef \_PHILOSOPHER\_H\_

#define \_PHILOSOPHER\_H\_

#include <stdbool.h>

class Philosopher**{**

private**:**

int \_id**;**

int \_fd**;**

int \_leftChopStick**;**

int \_rightChopStick**;**

public**:**

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^constructors^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//

Philosopher**(){**

\_id **=** 0**;**

\_fd **=** 0**;**

**}**

Philosopher**(**int i**,** int fd**){**

\_id **=** i**;**

\_fd **=** fd**;**

**}**

Philosopher**(**int i**,** int fd**,** int totalPhilosophers**){**

\_id **=** i**;**

\_fd **=** fd**;**

\_leftChopStick **=** \_id**;**

\_rightChopStick **=** **(**\_id **+** 1**)** **%** totalPhilosophers**;**

**}**

//destructors

virtual **~**Philosopher**(){**

\_id **=** 0**;**

\_fd **=** 0**;**

**}**

//^^^^^^^^^^^^^^^^^^^^^acessors and mutators^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^///

void setId**(**int i**){**

\_id **=** i**;**

**}**

int getId**(){**

**return** \_id**;**

**}**

void setFd**(**int fd**){**

\_fd **=** fd**;**

**}**

int getFd**(){**

**return** \_fd**;**

**}**

int getLeftChopStick**(){**

**return** \_leftChopStick**;**

**}**

int getRightChopStick**(){**

**return** \_rightChopStick**;**

**}**

/\*

void setLeftChopStick(ChopStick l){

}

void setRightChopStick(ChopStick r){

}

\*/

**};**

#endif

**CentralizedServer.cpp**

//Standarad libraries

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <vector>

#include <queue>

#include <string.h>

#include <string>

#include <iostream>

#include <stdbool.h>

//linux libraries

#include <unistd.h>

#include <errno.h>

#include <time.h>

//socket libraries

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

//pthreads

#include <pthread.h>

//custom library

#include "Philosopher.h" //provides philosopher object

**using** **namespace** std**;**

#define WRITE\_BUF\_SZIE 20 //buffer size for socket communication

#define NUMBER\_OF\_PHILOSOPHERS 5 //totla numberof clients/philosophers

#define GENERAL\_SLEEP\_TIME 2 //sleep duration for threads

#define GENERAL\_USLEEP\_TIME 500 //other sleep duration for threads

//Mutex for locking Queues

pthread\_mutex\_t Mutex1 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex2 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex3 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex4 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex5 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex6 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex7 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex8 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex9 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex10 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex11 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t Mutex12 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

//stuck to hold client socket descriptor for thread safety

**typedef** struct **{**

int \_fd**;**

**}** ClientFd**;**

//vector to indicate a chopstick's avialabilty

//vector<boo> \_chopstickCollection;

bool \_chopstickCollection**[**NUMBER\_OF\_PHILOSOPHERS**];**

//Queue to hold clients/philosophers who are requesting chopsticks to eat

queue**<**Philosopher**>** PendingChopStickRequests**;**

//Queue to hold clients/philosophers who are releaseing chopsticks after done eating

queue**<**Philosopher**>** PendingChopStickReleases**;**

/\*

\* This function is used to lock stdout stream so that only onethread can acess it at a time

\* @param \_dis char buf/char \* to be displayed

\* @return none

\*/

void displayString**(**char **\*** \_dis**){**

pthread\_mutex\_lock**(&**Mutex11**);** //lock mutex

cout**<<**\_dis**<<**endl**;** //print input string

pthread\_mutex\_unlock**(&**Mutex11**);** //unlock mutex

**}**

///^^^^^^^^^^^^^^^^^^^^^^^^^^^^^RELEASE QUEUE ACESSORS WITH MUTEXES FOR THREAD SAFETY ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^///

/\*

\* This function is used to get the top philosopher item ChopStick Release Queue

\* @return \_phil philosopher object that was on the top of the queue

\*/

Philosopher getFrontChopStickReleasesQueue**(){**

Philosopher \_phil**;** //create Philosopher object

pthread\_mutex\_lock**(&**Mutex10**);** //lock mutex

\_phil **=** PendingChopStickReleases**.**front**();** //get the philosopher item in top of the queue

pthread\_mutex\_unlock**(&**Mutex10**);** //unlock mutex

**return** \_phil**;** //return the top philosopher item

**}**

/\*

\* This function is used to dequeue a philospher object from the ChopStick Release Queue

\* @return none

\*/

void dequeuePhilosopherInChopStickReleasesQueue**(){**

pthread\_mutex\_lock**(&**Mutex8**);** //lock mutex

PendingChopStickReleases**.**pop**();** //dequeue phil object

pthread\_mutex\_unlock**(&**Mutex8**);** //unlock item

**}**

/\*

\* This function is used to enqueue a philospher object from the ChopStick Release Queue

\* @param \_phil Philosopher object to be enqueued

\* @return none

\*/

void enqueuePhilosopherInChopStickReleasesQueue**(**Philosopher \_phil**){**

pthread\_mutex\_lock**(&**Mutex6**);** //lock mutex

PendingChopStickReleases**.**push**(**\_phil**);** //enqueue the philosopher item

pthread\_mutex\_unlock**(&**Mutex6**);** //unlock mutex

**}**

/\*

\* This function is used to get size of the ChopStick Release Queue

\* @return size of the queue

\*/

int getLengthChopStickReleasesQueue**(){**

int \_len **=** 0**;**

pthread\_mutex\_lock**(&**Mutex4**);** //lock mutex

\_len **=** PendingChopStickReleases**.**size**();** //get size

pthread\_mutex\_unlock**(&**Mutex4**);** //unlock mutex

**return** \_len**;** //return size

**}**

/\*

\* This function is used to check if the ChopStick Release Queue is empty or not

\* @return queue's empty status

\*/

bool isChopstickReleasesQueueEmpty**(){**

bool \_isEmpty**;**

pthread\_mutex\_lock**(&**Mutex2**);** //lock mutex

\_isEmpty **=** PendingChopStickReleases**.**empty**();** //check if the queue is empty

pthread\_mutex\_unlock**(&**Mutex2**);** //unlock mutex

**return** \_isEmpty**;** //return empty status

**}**

///^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//////////////////

///^^^^^^^^^^^^^^^^^^^^^^^^^^^^^REQUEST QUEUE ACESSORS WITH MUTEXES FOR THREAD SAFETY ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^////

/\*

\* This function is used to get the top philosopher item ChopStick REQUEST Queue

\* @return \_phil philosopher object that was on the top of the queue

\*/

Philosopher getFrontChopStickRequestQueue**(){**

Philosopher \_phil**;** //create Philosopher object

pthread\_mutex\_lock**(&**Mutex9**);**//lock mutex

\_phil **=** PendingChopStickRequests**.**front**();**//get the philosopher item in top of the queue

pthread\_mutex\_unlock**(&**Mutex9**);**//unlock mutex

**return** \_phil**;** //return the top philosopher item

**}**

/\*

\* This function is used to dequeue a philospher object from the ChopStick Request Queue

\* @return none

\*/

void dequeuePhilosopherInChopStickRequestsQueue**(){**

pthread\_mutex\_lock**(&**Mutex7**);** //lock mutex

PendingChopStickRequests**.**pop**();** //dequeue phil object

pthread\_mutex\_unlock**(&**Mutex7**);**//unlock mutex

**}**

/\*

\* This function is used to enqueue a philospher object from the ChopStick request Queue

\* @param \_phil Philosopher object to be enqueued

\* @return none

\*/

void enqueuePhilosopherInChopStickRequestsQueue**(**Philosopher \_phil**){**

pthread\_mutex\_lock**(&**Mutex5**);** //lock mutex

PendingChopStickRequests**.**push**(**\_phil**);** //enqueue the philosopher item

pthread\_mutex\_unlock**(&**Mutex5**);** //unlock mutex

**}**

/\*

\* This function is used to get size of the ChopStick REQUEST Queue

\* @return size of the queue

\*/

int getLengthChopStickRequestsQueue**(){**

int \_len **=** 0**;**

pthread\_mutex\_lock**(&**Mutex3**);** //lock mutex

\_len **=** PendingChopStickRequests**.**size**();** //get size

pthread\_mutex\_unlock**(&**Mutex3**);** //unlock mutex

**return** \_len**;** //return size

**}**

/\*

\* This function is used to check if the ChopStick REQUEST Queue is empty or not

\* @return queue's empty status

\*/

bool isChopStickRequestsQueueEmpty**(){**

bool \_isEmpty**;**

pthread\_mutex\_lock**(&**Mutex1**);** //lock mutex

\_isEmpty **=** PendingChopStickRequests**.**empty**();** //check if the queue is empty

pthread\_mutex\_unlock**(&**Mutex1**);** //unlock mutex

**return** \_isEmpty**;** //return empty status

**}**

///^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^//////////////////

/\*

\* This function handles client/philosopers connections,

\* it commicates with clients using read(recv)/write(send) commands

\* it parses data packets from clients, data apacekts contain client ide and reuest type

\* if the request type is "E" then the client is enqueue in Request queue list else

\* if the reuest type is "F" then the client is enqueue in Release queue

\* @param \_clientSocketFd client socket descriptor

\* @return none

\*/

void handleClientConnection**(**int \_clientSocketFd**){**

**for(;;){** //infine loop for continus connection

//Read Client's/Philosopher's Request or Comannd

char \_recvBuf**[**WRITE\_BUF\_SZIE**];**

int \_err **=** recv**(**\_clientSocketFd**,**\_recvBuf**,**WRITE\_BUF\_SZIE**,**0**);** //read recived data packet

\_recvBuf**[**WRITE\_BUF\_SZIE**]** **=** '\0'**;**

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedServer: Error In Recv"**);**

exit**(**3**);**

**}**

//Parse Id and message from the Message sent by client

int \_uPos **=** **((**string**)**\_recvBuf**).**find**(**"\_"**);**

int \_exPos **=** **((**string**)**\_recvBuf**).**find**(**"!"**);**

string \_readPtr **=** **((**string**)**\_recvBuf**).**substr**(**0**,**\_uPos**);**

int \_philosopherId **=** stoi**(**\_readPtr**);**

\_readPtr **=** **((**string**)**\_recvBuf**).**substr**(**\_uPos**+**1**,(**\_exPos**-**\_uPos**-**1**));**

//create philosophers and set their chopstick index

Philosopher \_philTemp**(**\_philosopherId**,**\_clientSocketFd**,**NUMBER\_OF\_PHILOSOPHERS**);**

//Peform Queue Operations based on Client's Message

**if(**\_readPtr **==** "E"**){** //if the client/philosopher requeted to eat

enqueuePhilosopherInChopStickRequestsQueue**(**\_philTemp**);** //push the philospher/client in Chopstick Request Queue

**}**

**else** **if(**\_readPtr **==** "F"**){** //if the client/philosopher is done eating

enqueuePhilosopherInChopStickReleasesQueue**(**\_philTemp**);** //push the philosopher/Client in ChopStick Release Queue

**}**

sleep**(**GENERAL\_SLEEP\_TIME**);**

**}**

close**(**\_clientSocketFd**);** //close client socket file descriptor

**}**

/\*

\* THis function is a pthread start routine which is used to handle communication with a client/philosopher

\* @param \_args client socket descriptor struct

\* @return none

\*/

void **\*** startPhilosophers**(**void **\*** \_args**){**

//Socket descriptor for client connection

int \_clientSocketFd**;**

//Guarantees that thread resources are deallocated upon return

pthread\_detach**(**pthread\_self**());**

//Extract socket file descriptor from argument

\_clientSocketFd **=** **((** ClientFd **\*)**\_args**)->**\_fd**;** //get client socket descriptor

free**(**\_args**);** //free allocaed memory for client socket struct

handleClientConnection**(**\_clientSocketFd**);**//start communication with client

**}**

/\*

\* This function handles coordinator thread suck as granting/revoking chopstick from/to client/philosopher

\* @return none

\*/

void handleCoordinator**(){**

**for(;;){** //initite loop for continuos connection

//Release unused chopsticks

**if(**isChopstickReleasesQueueEmpty**()** **==** **false){** //if chopstick request queue is not empty

int \_len **=** getLengthChopStickReleasesQueue**();** //get lenngth of chopstick request queue

**for(**int i **=**0**;** i **<** \_len **;** i**++){**

Philosopher \_tempPhil **=** getFrontChopStickReleasesQueue**();** //get the first Philosopher in the release queue

\_chopstickCollection**[**\_tempPhil**.**getLeftChopStick**()]=** **true;** //free left chop stick

\_chopstickCollection**[**\_tempPhil**.**getRightChopStick**()]=** **true;** //free right chop stick

char \_buf**[**200**];**

//sprintf(\_buf,"CentralizedServer: Philosopher(%d) has dropped Chopsticks%d and %d at Time: %d",\_tempPhil.getId(),\_tempPhil.getLeftChopStick(),\_tempPhil.getRightChopStick(),(int)time(NULL));

sprintf**(**\_buf**,**"CentralizedServer: Philosopher(%d) has dropped Chopsticks%d and %d"**,**\_tempPhil**.**getId**(),**\_tempPhil**.**getLeftChopStick**(),**\_tempPhil**.**getRightChopStick**());**

displayString**(**\_buf**);** //print whcihc philosopher dropped which chopsticks

dequeuePhilosopherInChopStickReleasesQueue**();** //pop the philosopherfrom the queue

sleep**(**1**);**

**}**

**}**

usleep**(**GENERAL\_USLEEP\_TIME**);**

//sleep(GENERAL\_SLEEP\_TIME);

//Grant Chopsticks to philosophers waiting in request Queue

**if(**isChopStickRequestsQueueEmpty**()** **==** **false){** //if the request quque

int \_len **=** getLengthChopStickRequestsQueue**();** //get length of chopstick release queue

**for(**int i **=** 0**;** i **<** \_len **;** i**++){**

Philosopher \_tempPhil **=** getFrontChopStickRequestQueue**();** //get the top item in the choptick release queue

bool \_isLeftFree **=** \_chopstickCollection**[**\_tempPhil**.**getLeftChopStick**()];** //chcek to see if the chopstick index is free at the chostick collection

bool \_isRightFree **=** \_chopstickCollection**[**\_tempPhil**.**getRightChopStick**()];**//check another chopstick index

**if((**\_isLeftFree **==** **true)** **&&** **(**\_isRightFree **==** **true)){** //if requested chopsticks are available

\_chopstickCollection**[**\_tempPhil**.**getLeftChopStick**()]=** **false;** //lock left chop stick

\_chopstickCollection**[**\_tempPhil**.**getRightChopStick**()]=** **false;** //lock right chop stick

char \_buf**[**200**];**

//sprintf(\_buf,"CentralizedServer: Philosopher(%d) has pickedup Chopsticks%d and %d at Time: %d",\_tempPhil.getId(),\_tempPhil.getLeftChopStick(),\_tempPhil.getRightChopStick(),(int)time(NULL));

sprintf**(**\_buf**,**"CentralizedServer: Philosopher(%d) has pickedup Chopsticks%d and %d"**,**\_tempPhil**.**getId**(),**\_tempPhil**.**getLeftChopStick**(),**\_tempPhil**.**getRightChopStick**());**

displayString**(**\_buf**);** //display which philosopher picked up which chosticks

//send chopsticks

char \_sendBuf**[**WRITE\_BUF\_SZIE**];**

sprintf**(**\_sendBuf**,**"GRANT"**);**

int \_err **=** send**(**\_tempPhil**.**getFd**(),**\_sendBuf**,**WRITE\_BUF\_SZIE**,**0**);** //indicate chopstick grant to client

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedServer: Error In Send"**);**

exit**(**4**);**

**}**

//sleep(GENERAL\_SLEEP\_TIME);

dequeuePhilosopherInChopStickRequestsQueue**();** //since the chopstick was grant to this clint.philosopher, remove this fro reuest queue

**}**

**else{**

dequeuePhilosopherInChopStickRequestsQueue**();** //pop the philosopher from the reuest queue

enqueuePhilosopherInChopStickRequestsQueue**(**\_tempPhil**);** //enque at the bootom of the quque

//char \_buf[200];

//sprintf(\_buf,"CentralizedServer: Philosopher(%d): waiting in Line. Required Chopsticks Currrently Not Available \n",\_tempPhil.getId());

//displayString(\_buf);

**}**

//sleep(GENERAL\_SLEEP\_TIME);

**}**

**}**

sleep**(**GENERAL\_SLEEP\_TIME**);**

**}**

**}**

/\*

\* This fucntion is the srat routine for coordinator pthread, it handle all coordinator jobs

\* @param \_args NULL

\* @return none

\*/

void **\*** startCoordinator**(**void **\*** \_args**){**

pthread\_detach**(**pthread\_self**());** //Guarantees that thread resources are deallocated upon return

handleCoordinator**();** //communicate with client communication

**return(NULL);**

**}**

/\*

\* This function accepts client connects for given server file descriptor and return clien socke descriptor upon sucess

\* @param \_serverSocket server socket file descripto

\* @return \_clientSocet descriptor uopon sucessful connection

\*/

int acceptClientConnection**(**int \_serverSocket**){**

//acept clinet connections

//int accept(int socketfd,

//struct sockaddr \*restrict address,

// socklent \*restrict address\_len);

int \_clientSocket **=** accept**(**\_serverSocket**,NULL,NULL);** //accept client connection

**if(**\_clientSocket **==** **-**1**){**

perror**(**"CentralizedServer: Could Not Accept Client Connection"**);**

exit**(**4**);**

**}**

**return** \_clientSocket**;** //return client coket descriptor

**}**

/\*

\* This fnction creatoes a TCP server socket and return the server socket file descriptor

\*/

int createTcpServerSocket**(){**

//create socket descriptor

//arg1: Protocol Families : AF\_INET (IPV4 Internet Protocols)

//arg2: SOCK\_STREAM : Connection Oriented socket

//arg3: 0 : protocol: system will select protocol

//based on protocol family (AF\_INET)

int \_serverSocket **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**//create server socket

**if(**\_serverSocket **==** **-**1**){**

perror**(**"CentralizedServer: Could Not Create Socket Descriptor"**);**

exit**(**1**);**

**}**

**else{**

printf**(**"CentralizedServer: Socket File descriptor created\n"**);**

**}**

//assign a name to Socket

struct sockaddr\_in \_serverAddress**;**

\_serverAddress**.**sin\_family **=** AF\_INET**;** //protocl family

\_serverAddress**.**sin\_port **=** htons**(**9112**);** //sever port number

\_serverAddress**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_ANY**);** //server address

//bind socket to specified IP and port

int \_err **=** bind**(**\_serverSocket**,** **(**struct sockaddr**\*)&**\_serverAddress**,sizeof(**\_serverAddress**));** //blind socket descriptor to name

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedServer: Could not Not Bind Socket to Specified Address"**);**

exit**(**2**);**

**}**

**else{**

printf**(**"CentralizedServer: Socket Fd Bind Sucessful\n"**);**

**}**

//listent to client connections

//int listen(int sockfd, int backlog)

//sockfd: socket Descriptor

//backlor: maximum length wo which

//queue of pending connections for

//sockfd may grow

\_err **=** listen**(**\_serverSocket**,**NUMBER\_OF\_PHILOSOPHERS**);** //listen to client connections

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedServer: Could Not Initiate Listen"**);**

exit**(**3**);**

**}**

**else{**

printf**(**"CentralizedServer: Server is now ready to listen\n"**);**

**}**

**return** \_serverSocket**;** //return server socket file descriptor

**}**

/\*

\* This starting point of the program

\*/

int main**(**int \_argc**,** char **\***\_argv**[]){**

//initialize chopsticks

int i **=** 0**;**

**for(**i **=** 0**;** i **<** NUMBER\_OF\_PHILOSOPHERS**;** i**++){**

\_chopstickCollection**[**i**]** **=** **true;** //intiialilly make all chopsticks available

**}**

//create server socket

int \_serverSocketFd **=** createTcpServerSocket**();**

//create pthread type for coordinator

pthread\_t \_coordinator**;**

//create pthread types for philosophers,

pthread\_t \_philosopherList**[**NUMBER\_OF\_PHILOSOPHERS**];**

//start coordinator

int \_err **=** pthread\_create**(&**\_coordinator**,NULL,**startCoordinator**,NULL);**

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedServer: COuld Not create and start coordinator thread"**);**

**}**

int \_philosopherId **=** 0**;**

**for(;;){**

//wait for client connection

int \_clientSocketFd **=** acceptClientConnection**(**\_serverSocketFd**);**

//Create separate memory for client argument

ClientFd**\*** \_cfd **=** **(**ClientFd **\*)**malloc**(sizeof(**ClientFd**));**

\_cfd**->**\_fd **=** \_clientSocketFd**;**

//start seperate thread for this client

\_err **=** pthread\_create**(&**\_philosopherList**[**\_philosopherId**],NULL,**startPhilosophers**,(**void **\*)**\_cfd**);**

//increase count

\_philosopherId**++;**

**if(**\_philosopherId **>** NUMBER\_OF\_PHILOSOPHERS**){**

perror**(**"CentralizedServer: Error!: Client EXCEEDED 5. About to exit"**);**

exit**(**15**);**

**}**

**}**

//join coordinator back to main thread

pthread\_join**(**\_coordinator**,NULL);**

//join all client thread back to main thread

**for(**i **=** 0**;** i **<** \_philosopherId**;** i**++){**

pthread\_join**(**\_philosopherList**[**i**],NULL);**

**}**

**return** 0**;**

**}**

**Centralized Client.cpp**

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <errno.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

//sockets

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include "Philosopher.h" //provides philosopher object

#define INTERVAL\_MAX 7//Maximum time interval for random number generation

#define INTERVAL\_MIN 3//Minimum time interval for random number generation

#define WRITE\_BUF\_SZIE 20 //Buffer size

#define RUN\_COUNT 15 ///this program will eat and think this many times

/\*

\* This function simulates the thinking process for a philospoher with random think time

\* @param \_phil Philosoher object which containg philosopher id which will think

\* @return none

\*/

void philosopherIsThinking**(**Philosopher \_phil**){**

//seed rand

//srand(time(0));

int \_randomInterval **=** **(**random**()** **%** **(**INTERVAL\_MAX **-** INTERVAL\_MIN **+** 1**))** **+** INTERVAL\_MIN**;** //get random interval for sleep

printf**(**"CentralizedClient: Philosopher(%d) has started thinking. Time: %i\n"**,**\_phil**.**getId**(),(**int**)**time**(NULL));**

sleep**(**\_randomInterval**);** //sleep

printf**(**"CentralizedClient: Philosopher(%d) has completed thinking. Time: %i\n"**,**\_phil**.**getId**(),(**int**)**time**(NULL));**

**}**

/\*

\* This function simulates the eating process of a philospher with random eat time

\* @param \_phil philospoher object that container philosopher id tha is used to think

\* @return none

\*/

void philosopherIsEating**(**Philosopher \_phil**){**

//seed rand

//srand(time(0));

//

//reuest server for chosticks

char \_sendBuf**[**WRITE\_BUF\_SZIE**];**

sprintf**(**\_sendBuf**,**"%i\_E!"**,**\_phil**.**getId**());**

\_sendBuf**[**WRITE\_BUF\_SZIE**]** **=** '\0'**;**

int \_err **=** send**(**\_phil**.**getFd**(),**\_sendBuf**,**WRITE\_BUF\_SZIE**,**0**);** //send reuest to server

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedClient: Error While Sending Id to Server"**);**

exit**(**3**);**

**}**

//waitfor server's response to request

char \_recvBuf**[**WRITE\_BUF\_SZIE**];**

\_err **=** recv**(**\_phil**.**getFd**(),**\_recvBuf**,**strlen**(**\_recvBuf**),**0**);** //recevive the request

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedClient: Error While Receving from Server"**);**

exit**(**4**);**

**}**

**else{**

//start eating

printf**(**"CentralizedClient: Philosopher(%d) has started eating. Time: %i \n"**,**\_phil**.**getId**(),(**int**)**time**(NULL));**

int \_randomInterval **=** **(**random**()** **%** **(**INTERVAL\_MAX **-** INTERVAL\_MIN **+** 1**))** **+** INTERVAL\_MIN**;**

sleep**(**\_randomInterval**);**

printf**(**"CentralizedClient: Philosopher(%d) has finished eating. Time: %i \n"**,**\_phil**.**getId**(),(**int**)**time**(NULL));**

//Notifying server that philosopher has completed eating

sprintf**(**\_sendBuf**,**"%i\_F!"**,**\_phil**.**getId**());**

\_err **=** send**(**\_phil**.**getFd**(),**\_sendBuf**,**strlen**(**\_sendBuf**),**0**);**

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedClient: Error While Sending Id to Server"**);**

exit**(**5**);**

**}**

**}**

**}**

/\*

\* this function create a client socket descriptor and return the descriptor

\* return client socket file descritor

\*/

int createClientSocketDescriptorAndConnectToServer**(){**

//create client socket

int \_clientSocket **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**

**if(**\_clientSocket **==** **-**1**){**

perror**(**"CentralizedClient: Creation of Clinet Socket Descriptor failed"**);**

exit**(**1**);**

**}**

//specifiy and address for socket

struct sockaddr\_in \_serverAddress**;**

\_serverAddress**.**sin\_family **=** AF\_INET**;** //IPV4 Family Protocol

\_serverAddress**.**sin\_port **=** htons**(**9002**);**

\_serverAddress**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_ANY**);**

//connect to remote server

int \_err **=** connect**(**\_clientSocket**,** **(**struct sockaddr**\*)&**\_serverAddress**,sizeof(**\_serverAddress**));**

**if(**\_err **==** **-**1**){**

perror**(**"CentralizedClient: Cannot Connect to Server"**);**

exit**(**2**);**

**}**

**return** \_clientSocket**;** //return socket descriptor

**}**

/\*

\* This is tarting point for this program

\* @this main routine accepts client id as argv[1]

\*/

int main**(**int \_argc**,** char **\*** \_argv**[]){**

int \_clientSocketFd **=** createClientSocketDescriptorAndConnectToServer**();**//create client socket descriptor

Philosopher \_phil**(**atoi**(**\_argv**[**1**]),**\_clientSocketFd**);**//get client id(\_argv[1]) and create philospopher object

srand**(**time**(**0**));** //seed random

int \_count **=** 0**;**

**for(;;){** //start continuos client simulation loop

philosopherIsEating**(**\_phil**);** //eat

philosopherIsThinking**(**\_phil**);** //think

\_count **=** \_count **+** 1**;** //incremet activity counts

**if(**\_count **>** RUN\_COUNT**){** //if the activinty coutnreaches a constant time then exit

printf**(**"CentralizedClient: About to Exit. This Client ran % times\n"**,**RUN\_COUNT**);**

exit**(**1**);**

**}**

**}**

**return** 0**;**

**}**

**StartClients.cpp**

//This program is sued to start n number of clients

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/wait.h>

//start point of the program number of clients is to be specified here in argv[1]

int main**(**int \_argc**,** char **\*** \_argv**[]){**

**if(**\_argc **<** 1**){**

perror**(**"StartClient: Need to specifiy Number of clients"**);**

exit**(**5**);**

**}**

int \_totalClient **=** atoi**(**\_argv**[**1**]);**

**if(**\_totalClient **<** 5**){**

perror**(**"StartClient: Need 5 pilosophers "**);**

exit**(**6**);**

**}**

printf**(**"Total Clients: %i\n"**,**\_totalClient**);**

pid\_t \_pid**;**

int i **=** 0**;**

//Start child process process by \_toalClient

**for(**i **=** 0**;** i **<** \_totalClient**;**i**++){**

\_pid **=** fork**();** //creat child process

**if(**\_pid **<** 0**){** //if fork() wasnt sucessful print error

perror**(**"StartClients: Could Not Fork"**);**

exit**(**0**);**

**}**

**else** **if(**\_pid **==** 0**){** //if in child process start a clint

char \_buf**[**10**];**

sprintf**(**\_buf**,**"%i"**,**i**);**

printf**(**"About To Start Client: %i\n"**,**i**);**

int \_err **=** execl**(**"./clientCentralized"**,**"Test"**,**\_buf**,(**char **\*)NULL);**

**if(**\_err **==** **-**1**){**

perror**(**"COuld Not Execute Execl"**);**

exit**(**1**);**

**}**

**}**

**}**

//wait for child process to terminate so that they dont become zombie processes

**for(**i **=** 0**;** i **<** \_totalClient**;** i**++){**

int \_status **=** 0**;**

int \_err **=** waitpid**(-**1**,&**\_status**,**0**);**

**if(**\_err **==** **-**1**){**

perror**(**"StartClient: Could Not perfrom waitpid. About to exit"**);**

exit**(**2**);**

**}**

**}**

**return** 0**;**

**}**

**Distributed Algorithm Code**

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<errno.h>

#include<algorithm>

#include<random>

#include<queue>

#include<time.h>

#include<unistd.h>

#include<vector>

#include<string.h>

#include<stdbool.h>

**using** **namespace** std**;**

#define server\_port 7060

#define MAX\_RANDOM 20

#define NUM\_PROCS 5 //TODO Remember to change the number of clients here as well

std**::**random\_device seeder**;**

std**::**mt19937 rng**(**seeder**());**

std**::**uniform\_int\_distribution**<**int**>** gen**(**4**,** MAX\_RANDOM**);** // Use : gen (rng)

// Client Queue that keeps all the requests from other client for chopstick

std**::**queue**<**char**\*>** myqueue**;**

// Timestamp of the client, when it requests the resource (chopstick)

long requested\_timestamp **=** 0**;**

// Boolean value that signifies whether a client is eating at a particular instance

bool isEating **=** **false;**

bool has\_left\_cp **=false;**

bool has\_right\_cp **=false;**

/\* DESCRIPTION OF MESSAGE

\*

\* CLIENT/PROCESS ID \_ REQ/RES \_ int CHOP1 \_ int CHOP2 \_ [timestamp]\_ [OK]

\*

\* Example of Request : 452\_REQ\_1,2\_22344567 , In case of Request first param will be the source

\*

\* Example of Response : 213\_RES\_CHOPSTICK\_OK , In case of Response first param will be the destination

\*

\* In above message description, anything surrounded by [] is optional and should be used as required

\*

\* Request message does not send the last parameter [OK]

\*

\* Response parameter does not contain the [timestamp] parameter.

\*

\* MESSAGE\_TO\_SEND : OK, If it is not using resource, or has lower timestamp

\* Else Queues the request, to reply at later stage

\*

\* MESSAGE\_TO\_RECEIVE: OK , If permission is granted to use the resource

\* Else waits for permisssion

\*

\*/

void eat\_cycle**(**int myid**,** int cli\_sd**)**

**{**

/\* In this function, the philospher intiates to request for chopstick and eat

\* Input : myid -> client id assigned by the server after connecting

\* cli\_sd -> socket descriptor for communication

\* Returns : None

\*

\*/

int left\_chopstick **=** myid**;**

int right\_chopstick **=** **(**myid **+** 1 **)** **%** NUM\_PROCS**;**

// Request for the chopsticks

char request\_buffer**[**150**];**

char response\_buffer**[**150**];**

char incoming\_buffer**[**150**];**

**if** **(**requested\_timestamp **==** 0**)** // Only one request per cycle, making sure it does not flood o/p console

**{**

requested\_timestamp**=** time**(NULL);**

sprintf**(**request\_buffer**,** "%d\_REQ\_%d\_%d\_%ld"**,** myid**,**

left\_chopstick**,**right\_chopstick**,** requested\_timestamp **);**

printf**(**"\nclient[%d]: Waiting for the chopstick\n"**,**myid**);**

**if(** **(**send**(**cli\_sd**,** request\_buffer**,** 150**,**0 **))** **<** 0**)**

**{**printf**(**"Client[%d]: Failed to send the request to other clients"**,** myid**);}**

**}**

**if** **((**recv**(**cli\_sd**,** incoming\_buffer**,** 150**,** 0 **))** **<** 0**)** // receive one message at a time

**{**printf**(**"Client[%d]: Failed to receive incoming messages from other clients"**,**myid**);}**

**else**

**{**

sleep**(**5**);**

// process the request

printf**(**"\n Incoming buffer : %s\n"**,** incoming\_buffer**);**

vector**<**char**\*>** incoming\_tokens**;**

char**\*** token **=** strtok**(**incoming\_buffer**,**"\_"**);**

**while(**token **!=** **NULL** **)**

**{**

incoming\_tokens**.**push\_back**(**token**);**

token **=**strtok**(NULL,** "\_"**);**

**}**

char**\*** iop**=**incoming\_tokens**[**1**];**

printf**(**"\n%s\n"**,** iop**);**

printf**(**"\n %s ---> %s \n"**,** incoming\_tokens**[**1**],** "REQ"**);**

// If the request of same client is sent back to the client, then omit the request

**if** **(**strncmp**(**iop**,** "REQ"**,**strlen**(**"REQ"**))==**0**)**

**{**

printf**(**"\n parsed requests"**);**

**if** **(** atoi**(**incoming\_tokens**[**0**])** **==**myid**)**

**{**

int asdasdasda **=** 0**;**

**}**

**else** **if** **((**atoi**(**incoming\_tokens**[**2**])** **!=** left\_chopstick **&&**

atoi**(**incoming\_tokens**[**2**])** **!=** right\_chopstick**)** **&&**

**(**atoi**(**incoming\_tokens**[**3**])** **!=** left\_chopstick **&&**

atoi**(**incoming\_tokens**[**3**])** **!=** right\_chopstick**))**

**{**

char temp\_buffer**[**150**];**

sprintf**(**temp\_buffer**,** "%d\_RES\_%d\_%s"**,**atoi**(**incoming\_tokens**[**0**]),**999**,**"OK"**);** //dummy value 999

printf**(**"\nclient[%d]: Sending OK Message to client %d"**,**myid**,** atoi**(**incoming\_tokens**[**0**]));**

//send response

send**(**cli\_sd**,** temp\_buffer**,** 150**,**0**);**

**}**

**else** // It uses the resources i.e., chopstick

**{**

**if(**atol**(**incoming\_tokens**[**4**])** **<** requested\_timestamp**)**

**{**

**if** **(**left\_chopstick **==** atoi**(**incoming\_tokens**[**2**])** **||**

left\_chopstick **==** atoi**(**incoming\_tokens**[**3**])** **)**

**{**

has\_left\_cp **=** **false;**

char temp\_buffer**[**150**];**

sprintf**(**temp\_buffer**,** "%d\_RES\_%d\_%s"**,**atoi**(**incoming\_tokens**[**0**]),**left\_chopstick**,**"OK"**);**

printf**(**"\nclient[%d]: Granting chopstick %d"**,**myid**,** left\_chopstick**);**

//send response

send**(**cli\_sd**,** temp\_buffer**,** 150**,**0**);**

**}**

**else** //TODO mttp ->ctctcelp

**{**

has\_right\_cp **=** **false;**

char temp\_buffer**[**150**];**

sprintf**(**temp\_buffer**,** "%d\_RES\_%d\_%s"**,**atoi**(**incoming\_tokens**[**0**]),**right\_chopstick**,**"OK"**);**

printf**(**"\nclient[%d]: Granting chopstick %d"**,**myid**,** right\_chopstick**);**

//send response

send**(**cli\_sd**,** temp\_buffer**,** 150**,**0**);**

**}**

**}**

**else** // timestamp of request is greater, add in queue

**{**myqueue**.**push**(**incoming\_buffer**);}**

**}**

**}**// end of if

printf**(**"\nNope Its not REQ\n"**);**

**if(**incoming\_tokens**[**1**]** **==** "RES"**)**

**{**

**if** **(**incoming\_tokens**[**3**]** **==** "OK"**)**

**{**

**if** **(**atoi**(**incoming\_tokens**[**2**])** **==** left\_chopstick**)**

**{**has\_left\_cp **=** **true;}**

**else** **if(**atoi**(**incoming\_tokens**[**2**])** **==** right\_chopstick**)**

**{**has\_right\_cp **=** **true;}**

**else{;}**

**}**

**}**

**else{;}**

**}** //outside of else loop, "reads from the receiver socket descriptor"

// If it has both the chopsticks to eat , then eat

**if** **(**has\_left\_cp **&&** has\_right\_cp **)**

**{**

printf**(**"\nclient[%d]: Started Eating with chopsticks %d and %d"**,**myid**,** left\_chopstick**,** right\_chopstick**);**

//sends OK messages to all pending queue elements and empty the queue

sleep**(**5**);**

printf**(**"\nclient[%d]: Finished Eating with chopsticks %d and %d"**,**myid**,** left\_chopstick**,** right\_chopstick**);**

requested\_timestamp **=** 0**;** // This will make the client to send another request to eat, after the rest cycle completes

**for** **(**int p**=**0**;** p**<** myqueue**.**size**();**p**++)**

**{**

vector**<**char**\*>** temp\_tokens**;**

char**\*** t\_token **=** strtok**(**incoming\_buffer**,**"\_"**);**

**while(**t\_token **!=** **NULL** **)**

**{**

temp\_tokens**.**push\_back**(**t\_token**);**

t\_token **=**strtok**(NULL,** "\_"**);**

**}**

char queue\_response\_buffer**[**150**];**

sprintf**(**queue\_response\_buffer**,** "%d\_RES\_999\_%s"**,**atoi**(**temp\_tokens**[**0**]),**"OK"**);** // '999' dummy value, just sending ok message

printf**(**"\nclient[%d]: Sending OK to process %d"**,**myid**,** atoi**(**temp\_tokens**[**0**]));**

//send response

send**(**cli\_sd**,** queue\_response\_buffer**,** 150**,**0**);**

myqueue**.**pop**();** // pop the oldest element from the queue i.e., front element

**}** // end for loop

**}** //end if loop

**}**

void think\_cycle**(**int myid**,** int cli\_sd**)**

**{**

int nap\_time **=** gen**(**rng**);**

printf**(**"\nclient[%d]: Started thinking\n"**,** myid**);**

sleep**(**nap\_time**);**

printf**(**"\nclient[%d]: Finished thinking\n"**,**myid**);**

**}**

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

**{**

// create the client socket

printf **(**"\n\n"**);**

int client\_number **=** atoi**(**argv**[**1**]);**

int serv\_assigned\_cid**;**

int p\_client**,** ernum**;**

**if** **((**p\_client **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**))** **<** 0**)**

**{**

printf**(**"\nProcess [%d]: Failed to create socket for process\n"**,** client\_number **);**

**}**

**else{**printf**(**"\nProcess[%d]: Created client socket\n"**,** client\_number**);}**

struct sockaddr\_in serverAddress**;** // prepare the server address to connect to

serverAddress**.**sin\_family **=**AF\_INET**;**

serverAddress**.**sin\_port **=** htons**(**server\_port**);**

serverAddress**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_ANY**);**

**if((**ernum **=** connect**(**p\_client**,** **(**struct sockaddr**\*)** **&**serverAddress**,** **sizeof(**serverAddress**)))** **<**0**)**

**{**

printf**(**"\nProcess[%d]: Failed to connect to server process\n"**,** client\_number**);**

**}**

**else{**printf**(**"\nProcess[%d]: Successfully connected to server process\n"**,** client\_number**);}**

// Receive the client id that is assigned by the server

char client\_id\_buffer**[**150**];**

**if** **((**ernum **=** recv**(**p\_client**,** client\_id\_buffer**,** 150**,** 0 **))** **<** 0**)**

**{**printf**(**"\nProcess[%d]: Error receiving assigned client id from the broadcaster"**,**client\_number**);}**

**else**

**{**

sscanf**(**client\_id\_buffer**,**"%d"**,** **&**serv\_assigned\_cid**);**

printf**(**"Process[%d]: Received assigned client id as %d "**,**client\_number**,** serv\_assigned\_cid**);**

**}**

// Eat and think for infinite number of times

**for** **(;;)**

**{**

eat\_cycle**(**serv\_assigned\_cid**,** p\_client**);**

// think\_cycle(serv\_assigned\_cid, p\_client);

**}**

**return** 0**;**

**}**

#include<stdio.h>

#include<iostream>

#include<stdlib.h>

#include<vector>

#include<errno.h>

#include<string.h>

#include<map>

#include<errno.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netdb.h>

#include<queue>

#include<pthread.h>

#include<time.h>

#define MAX\_PROC 20

#define NUM\_PROC 3

**using** **namespace** std**;**

#define server\_port 7060

/\* There will be

\* 1 master socket, which broadcasts or passes message to all client sockets

\* 'n' client sockets, which will request for the chopstick\_acquire or chopstick\_release

\* All of 'n' sockets will pass message to each other with the help of broadcaster socket

\*/

**typedef** struct**{**

int sd\_client**;**

int id\_client**;**

**}**clientInfo**;**

// Mutex to protect the shared variable

pthread\_mutex\_t message\_queue\_lock **=** PTHREAD\_MUTEX\_INITIALIZER**;**

pthread\_mutex\_t message\_queue\_lock2 **=** PTHREAD\_MUTEX\_INITIALIZER**;**

//pthread\_mutex\_t mutex1 = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t queue\_condn **=** PTHREAD\_COND\_INITIALIZER**;**

// Global queue that tracks all the requests and responses sent by the clients

std**::**vector**<**int**>** \_clientsWhoRead**;**

std**::**queue**<**char**\*>** message\_list**;**

int queue\_flag**;**

int count **=** 0**;**

void**\*** serveClient**(**void**\*** params**)**

**{**

/\* This thread routine is called everytime a thread is created by this (broadcaster) program

\* Input: param of type null pointer, that contains the clientInfo Structure

\* Returns: #TODO Need to fill this out later

\*/

clientInfo c\_ **=** **\*((**clientInfo **\*)** params**);**

pthread\_detach**(**pthread\_self**());**

free**(**params**);**

std**::**queue**<**char**\*>** local\_messages**;**

char send\_from**[**150**];**

char receive\_from**[**150**];**

//printf("\n Socket descriptior of client: %d\n",c\_.sd\_client);

//printf("\n Id of the client is: %d\n",c\_.id\_client);

// First send the server assigned client id for the client

sprintf**(**send\_from**,** "%d"**,** c\_**.**id\_client**);**

**if** **((**send**(**c\_**.**sd\_client**,** send\_from**,** 150**,** 0**))** **<** 0 **)**

**{**

printf**(**"\nProcess [BROADCASTER]: Cannot send client id\n"**);**

**}**

// receive the requests, and push it into the queue by obtaining mutex lock

**for(;;)**

**{**

**if((**recv**(**c\_**.**sd\_client**,** receive\_from**,** 150 **,** 0**))** **<**0**){**

printf**(**"\nProcess [BROADCASTER]: Error receiving request from clients"**);**

**}**

**else**

**{** // pack the values to be pushed into the queue

// If v1, is of length 3, that means all the processes has taken the requests with them, and the request can be safely removed from queue

pthread\_mutex\_lock**(&**message\_queue\_lock**);**

message\_list**.**push**(**receive\_from**);**

pthread\_mutex\_unlock**(&**message\_queue\_lock**);**

// queue : I1 I2 I3

//thread: 1 2 3

pthread\_mutex\_lock**(&**message\_queue\_lock**);**

char**\*** individual\_messages **=** message\_list**.**front**();**

printf**(**"\n%s\n"**,** individual\_messages**);**

count **++;**

pthread\_mutex\_unlock**(&**message\_queue\_lock**);**

vector**<**char**\*>** msg\_tokens**;**

char temp**[**150**];**

strcpy**(**temp**,** individual\_messages**);**

char**\*** token **=** strtok**(**individual\_messages**,** "\_"**);**

printf**(**"\nCopied value: %s\n"**,** temp**);**

**while(**token **!=NULL)**

**{**

msg\_tokens**.**push\_back**(**token**);**

token**=**strtok**(NULL,**"\_"**);**

**}**

**if** **(**msg\_tokens**[**1**]** **==** "REQ" **)**

**{**

//if request token pass into the this thread's client

//

pthread\_mutex\_lock**(&**message\_queue\_lock2**);**

printf**(**"\n Before sending : %s\n"**,**individual\_messages**);**

**if** **((**send**(**c\_**.**sd\_client**,**temp**,** 150**,** 0**))** **<** 0 **)**

**{**printf**(**"\nProcess [BROADCASTER]: Cannot send client id\n"**);**

**}**

**else{**printf**(**"\n Sending buffer: %s"**,** individual\_messages**);}**

pthread\_mutex\_unlock**(&**message\_queue\_lock2**);**

**}**

**else** // if not request then , response message

**{** // this is only to be accessed by destination client and no one else

**if** **(**c\_**.**id\_client **==** atoi**(**msg\_tokens**[**0**]))**

**{**

// if the message is for this server's client

**if** **((**send**(**c\_**.**sd\_client**,**temp**,** 150**,** 0**))** **<** 0 **)**

**{**printf**(**"\nProcess [BROADCASTER]: Cannot send client id\n"**);}**

**else{**printf**(**"\n Sending buffer: %s\n"**,** individual\_messages**);}**

message\_list**.**pop**();** // Remove client specific message from global queue

**}**

**}**

printf**(**"\n Before mutex lock\n"**);**

pthread\_mutex\_lock**(&**message\_queue\_lock2**);**

printf**(**"\n AFter mutex lock"**);**

**if(**count **==** NUM\_PROC**){**

printf**(**"\nAbout to bcast\n"**);**

pthread\_cond\_broadcast**(&**queue\_condn**);**

count **=**0**;**

**}**

**else{** //if count != NUM\_PROCs

printf**(**"\n Waiting \n"**);**

pthread\_cond\_wait**(&**queue\_condn**,&**message\_queue\_lock2**);**

**}**

pthread\_mutex\_unlock**(&**message\_queue\_lock2**);**

//pthread\_mutex\_unlock(&message\_queue\_lo2ck2);

**}**

**}**

**}**

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

**{**

int p\_server**,** p\_port**,** ernum**;**

int client\_tracker **=** 0**;**

p\_port **=** server\_port**;**

pthread\_t thread\_id**[**NUM\_PROC**];**

//create process specific server socket

**if** **((**p\_server **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**))** **<** 0**)**

**{**

printf **(**"\nProcess [BROADCASTER] : Creation of server socket failed\n"**);**

**}**

**else** **{**printf**(**"\nProcess [BROACASTER] : Creation of server socket successful\n"**);}**

// Initialize the socket address structure and zero out structure

struct sockaddr\_in serverAddress**;**

memset**(&**serverAddress**,** 0**,** **sizeof(**serverAddress**));**

// Assign only the needed information to socket structure

serverAddress**.**sin\_family **=** AF\_INET**;**

serverAddress**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_ANY**);**

serverAddress**.**sin\_port**=** htons**(**p\_port**);**

//bind the socket to the local address

**if** **((**ernum **=** bind**(**p\_server**,** **(**struct sockaddr**\*)&**serverAddress**,** **sizeof(**serverAddress**))** **<** 0**))**

**{**

printf**(**"\nProcess [BROADCASTER] : Binding failed for the process server socket: %d\n"**,** ernum**);**

**}**

**else** **{**printf**(**"\nProcess [BROADCASTER] : Binding successful\n"**);}**

// Listen for incoming connection to the process server socket

**if** **(**listen**(**p\_server**,** MAX\_PROC **)** **<**0 **)**

**{**

printf **(**"\n Process [BROADCASTER]: Listening failed for processe's server socket\n"**);**

**}**

**else** **{**printf**(**"\nProcess [BROADCASTER] : Server socket now listining for incoming connection\n"**);}**

// wait for incoming request of the client

**for** **(;;)**

**{**

sleep**(**4**);** // This is necessary here, because the loop runs too fast, therefore

// multiple threads read same client socket descriptor value

int c\_sd**;**

//clientInfo temp;

**if** **((**c\_sd **=** accept**(**p\_server**,** **NULL,** **NULL))** **<** 0**)**

**{**

printf**(**"\nProcess [BROADCASTER]: Failed client connection\n"**);**

**}**

**else**

**{**

printf**(**"\nProcess [BROADCASTER]: Accepted client connection\n"**);**

//client\_sd.push(client\_tracker,c\_sd);

clientInfo**\*** temp **=** **(**clientInfo**\*)**malloc**(sizeof(**clientInfo**));**

temp**->**sd\_client**=** c\_sd**;** temp**->**id\_client **=** client\_tracker**;**

ernum**=** pthread\_create**(&**thread\_id**[**client\_tracker**],** **NULL,** serveClient**,(**void**\*)** temp**);**

client\_tracker**++;**

**}**

**}**

// wait for all the threads to complete

//

**for** **(**int k**=**0**;** k**<** client\_tracker**;** k**++)**

**{**

pthread\_join**(**thread\_id**[**k**],** **NULL);**

**}**

**return** 0**;**

**}**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/wait.h>

int main**(**int \_argc**,** char **\*** \_argv**[]){**

int \_totalClient **=** atoi**(**\_argv**[**1**]);**

printf**(**"Total Clients: %i\n"**,**\_totalClient**);**

pid\_t \_pid**;**

int i **=** 0**;**

**for(**i **=** 0**;** i **<** \_totalClient**;**i**++){**

\_pid **=** fork**();**

**if(**\_pid **<** 0**){**

perror**(**"StartClients: Could Not Fork"**);**

exit**(**0**);**

**}**

**else** **if(**\_pid **==** 0**){**

char \_buf**[**10**];**

sprintf**(**\_buf**,**"%i"**,**i**);**

printf**(**"Starting Client: %d\n"**,** i**);**

int \_err **=** execl**(**"./pclient"**,**"Test2"**,**\_buf**,(**char **\*)NULL);**

**if(**\_err **==** **-**1**){**

perror**(**"COuld Not Execute Execl"**);**

exit**(**1**);**

**}**

**}**

**}**

**for(**i **=** 0**;** i **<** \_totalClient**;** i**++){**

int \_status **=** 0**;**

waitpid**(-**1**,&**\_status**,**0**);**

**}**

**return** 0**;**

**}**

**Centralized Algorithm Code Output**

Higlighted portion show more than 1 philosophers eating at the same time

Script started on Mon Apr 13 13:37:38 2020

]0;ea4963aw@ahscentos:~/CSCI\_610/project4/Centralized[?1034h[ea4963aw@ahscentos Centralized]$ make

g++ -std=c++0x -pthread CentralizedServer.cpp Philosopher.h -o serverCentralized

g++ CentralizedClient.cpp Philosopher.h -o clientCentralized

g++ StartClient.cpp -o sc

]0;ea4963aw@ahscentos:~/CSCI\_610/project4/Centralized[ea4963aw@ahscentos Centralized]$ ./serverCentralized &

[1] 43929

]0;ea4963aw@ahscentos:~/CSCI\_610/project4/Centralized[ea4963aw@ahscentos Centralized]$ CentralizedServer: Socket File descriptor created

CentralizedServer: Socket Fd Bind Sucessful

CentralizedServer: Server is now ready to listen

^C

]0;ea4963aw@ahscentos:~/CSCI\_610/project4/Centralized[ea4963aw@ahscentos Centralized]$ ./c sc 5

Total Clients: 5

About To Start Client: 0

About To Start Client: 1

About To Start Client: 2

About To Start Client: 3

About To Start Client: 4

CentralizedServer: Philosopher(3) has pickedup Chopsticks3 and 4

CentralizedServer: Philosopher(1) has pickedup Chopsticks1 and 2

CentralizedClient: Philosopher(3) has started eating. Time: 1586803074

CentralizedClient: Philosopher(1) has started eating. Time: 1586803074

CentralizedClient: Philosopher(3) has finished eating. Time: 1586803080

CentralizedClient: Philosopher(1) has finished eating. Time: 1586803080

CentralizedClient: Philosopher(1) has started thinking. Time: 1586803080

CentralizedClient: Philosopher(3) has started thinking. Time: 1586803080

CentralizedServer: Philosopher(1) has dropped Chopsticks1 and 2

CentralizedServer: Philosopher(3) has dropped Chopsticks3 and 4

CentralizedServer: Philosopher(2) has pickedup Chopsticks2 and 3

CentralizedServer: Philosopher(4) has pickedup Chopsticks4 and 0

CentralizedClient: Philosopher(2) has started eating. Time: 1586803082

CentralizedClient: Philosopher(4) has started eating. Time: 1586803082

CentralizedClient: Philosopher(1) has completed thinking. Time: 1586803084

CentralizedClient: Philosopher(3) has completed thinking. Time: 1586803084

CentralizedClient: Philosopher(1) has started eating. Time: 1586803084

CentralizedClient: Philosopher(3) has started eating. Time: 1586803084

CentralizedClient: Philosopher(3) has finished eating. Time: 1586803088

CentralizedClient: Philosopher(1) has finished eating. Time: 1586803088

CentralizedClient: Philosopher(1) has started thinking. Time: 1586803088

CentralizedClient: Philosopher(3) has started thinking. Time: 1586803088

CentralizedClient: Philosopher(2) has finished eating. Time: 1586803088

CentralizedClient: Philosopher(4) has finished eating. Time: 1586803088

CentralizedClient: Philosopher(2) has started thinking. Time: 1586803088

CentralizedClient: Philosopher(4) has started thinking. Time: 1586803088

CentralizedServer: Philosopher(1) has dropped Chopsticks1 and 2

CentralizedServer: Philosopher(3) has dropped Chopsticks3 and 4

CentralizedServer: Philosopher(4) has dropped Chopsticks4 and 0

CentralizedServer: Philosopher(2) has dropped Chopsticks2 and 3

CentralizedClient: Philosopher(2) has completed thinking. Time: 1586803092

CentralizedClient: Philosopher(4) has completed thinking. Time: 1586803092

CentralizedClient: Philosopher(4) has started eating. Time: 1586803092

CentralizedClient: Philosopher(2) has started eating. Time: 1586803092

CentralizedServer: Philosopher(0) has pickedup Chopsticks0 and 1

CentralizedServer: Philosopher(3) has pickedup Chopsticks3 and 4

CentralizedClient: Philosopher(0) has started eating. Time: 1586803092

CentralizedClient: Philosopher(3) has completed thinking. Time: 1586803095

CentralizedClient: Philosopher(1) has completed thinking. Time: 1586803095

CentralizedClient: Philosopher(1) has started eating. Time: 1586803095

CentralizedClient: Philosopher(3) has started eating. Time: 1586803095

CentralizedClient: Philosopher(4) has finished eating. Time: 1586803096

CentralizedClient: Philosopher(2) has finished eating. Time: 1586803096

CentralizedClient: Philosopher(2) has started thinking. Time: 1586803096

CentralizedClient: Philosopher(4) has started thinking. Time: 1586803096

CentralizedServer: Philosopher(2) has dropped Chopsticks2 and 3

CentralizedServer: Philosopher(4) has dropped Chopsticks4 and 0

CentralizedServer: Philosopher(4) has pickedup Chopsticks4 and 0

CentralizedServer: Philosopher(2) has pickedup Chopsticks2 and 3

CentralizedClient: Philosopher(0) has finished eating. Time: 1586803098

CentralizedClient: Philosopher(0) has started thinking. Time: 1586803098

CentralizedClient: Philosopher(1) has finished eating. Time: 1586803099

CentralizedClient: Philosopher(3) has finished eating. Time: 1586803099

CentralizedClient: Philosopher(1) has started thinking. Time: 1586803099

CentralizedClient: Philosopher(3) has started thinking. Time: 1586803099

CentralizedServer: Philosopher(0) has dropped Chopsticks0 and 1

CentralizedServer: Philosopher(1) has dropped Chopsticks1 and 2

CentralizedServer: Philosopher(3) has dropped Chopsticks3 and 4

CentralizedClient: Philosopher(0) has completed thinking. Time: 1586803102

CentralizedClient: Philosopher(0) has started eating. Time: 1586803102

CentralizedClient: Philosopher(2) has completed thinking. Time: 1586803103

CentralizedClient: Philosopher(4) has completed thinking. Time: 1586803103

CentralizedClient: Philosopher(2) has started eating. Time: 1586803103

CentralizedClient: Philosopher(4) has started eating. Time: 1586803103

CentralizedServer: Philosopher(1) has pickedup Chopsticks1 and 2

CentralizedServer: Philosopher(3) has pickedup Chopsticks3 and 4

CentralizedClient: Philosopher(3) has completed thinking. Time: 1586803105

CentralizedClient: Philosopher(1) has completed thinking. Time: 1586803105

CentralizedClient: Philosopher(1) has started eating. Time: 1586803105

CentralizedClient: Philosopher(3) has started eating. Time: 1586803105

CentralizedClient: Philosopher(0) has finished eating. Time: 1586803106

CentralizedClient: Philosopher(0) has started thinking. Time: 1586803106

CentralizedClient: Philosopher(2) has finished eating. Time: 1586803107

CentralizedClient: Philosopher(4) has finished eating. Time: 1586803107

CentralizedClient: Philosopher(2) has started thinking. Time: 1586803107

CentralizedClient: Philosopher(4) has started thinking. Time: 1586803107

CentralizedServer: Philosopher(0) has dropped Chopsticks0 and 1

CentralizedClient: Philosopher(1) has finished eating. Time: 1586803108

CentralizedClient: Philosopher(3) has finished eating. Time: 1586803108

CentralizedClient: Philosopher(1) has started thinking. Time: 1586803108

CentralizedClient: Philosopher(3) has started thinking. Time: 1586803108

CentralizedServer: Philosopher(4) has dropped Chopsticks4 and 0

CentralizedServer: Philosopher(2) has dropped Chopsticks2 and 3

CentralizedServer: Philosopher(1) has pickedup Chopsticks1 and 2

CentralizedServer: Philosopher(4) has pickedup Chopsticks4 and 0

CentralizedClient: Philosopher(1) has completed thinking. Time: 1586803112

CentralizedClient: Philosopher(3) has completed thinking. Time: 1586803112

CentralizedClient: Philosopher(3) has started eating. Time: 1586803112

CentralizedClient: Philosopher(1) has started eating. Time: 1586803112

CentralizedServer: Philosopher(3) has dropped Chopsticks3 and 4

CentralizedClient: Philosopher(4) has completed thinking. Time: 1586803113

CentralizedClient: Philosopher(2) has completed thinking. Time: 1586803113

CentralizedClient: Philosopher(4) has started eating. Time: 1586803113

CentralizedClient: Philosopher(2) has started eating. Time: 1586803113

CentralizedClient: Philosopher(0) has completed thinking. Time: 1586803113

CentralizedClient: Philosopher(0) has started eating. Time: 1586803113

CentralizedServer: Philosopher(1) has dropped Chopsticks1 and 2

CentralizedServer: Philosopher(2) has pickedup Chopsticks2 and 3

**Distributed Algorithm Code Output**

Script started on 2020-04-13 18:33:07-0500

]0;swapnil@Swapnil\_Acharya: /mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/code/distributed[01;32mswapnil@Swapnil\_Acharya[00m:[01;34m/mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/co

ode/distributed[00m$ ps

PID TTY TIME CMD

84 pts/0 00:00:00 bash

92 pts/0 00:00:00 ps

]0;swapnil@Swapnil\_Acharya: /mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/code/distributed[01;32mswapnil@Swapnil\_Acharya[00m:[01;34m/mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/co

ode/distributed[00m$ ./distributed\_broadcaster &

[1] 93

]0;swapnil@Swapnil\_Acharya: /mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/code/distributed[01;32mswapnil@Swapnil\_Acharya[00m:[01;34m/mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/co

ode/distributed[00m$

Process [BROACASTER] : Creation of server socket successful

Process [BROADCASTER] : Binding successful

Process [BROADCASTER] : Server socket now listining for incoming connection

^C

]0;swapnil@Swapnil\_Acharya: /mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/code/distributed[01;32mswapnil@Swapnil\_Acharya[00m:[01;34m/mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/co

ode/distributed[00m$ ./sc 5

Total Clients: 5

Starting Client: 0

Starting Client: 1

Starting Client: 2

Starting Client: 3

Starting Client: 4

Process[0]: Created client socket

Process[0]: Successfully connected to server process

Process[1]: Created client socket

Process[1]: Successfully connected to server process

Process[2]: Created client socket

Process[2]: Successfully connected to server process

Process[3]: Created client socket

Process[3]: Successfully connected to server process

Process[4]: Created client socket

Process[4]: Successfully connected to server process

Process [BROADCASTER]: Accepted client connection

Process[0]: Received assigned client id as 0

client[0]: Waiting for the chopstick

0\_REQ\_0\_1\_1586820798

Copied value: 0\_REQ\_0\_1\_1586820798

Sending buffer: 0

Before mutex lock

AFter mutex lock

Waiting

Process [BROADCASTER]: Accepted client connection

Process[1]: Received assigned client id as 1

client[1]: Waiting for the chopstick

1\_REQ\_1\_2\_1586820802

Copied value: 1\_REQ\_1\_2\_1586820802

Sending buffer: 1

Before mutex lock

AFter mutex lock

Waiting

Incoming buffer : 0\_REQ\_0\_1\_1586820798

REQ

REQ ---> REQ

parsed requests

Nope Its not REQ

Process [BROADCASTER]: Accepted client connection

Process[2]: Received assigned client id as 2

client[2]: Waiting for the chopstick

2\_REQ\_2\_3\_1586820806

Copied value: 2\_REQ\_2\_3\_1586820806

Sending buffer: 2

Before mutex lock

AFter mutex lock

About to bcast

Incoming buffer : 1\_REQ\_1\_2\_1586820802

REQ

REQ ---> REQ

parsed requests

Nope Its not REQ

Process [BROADCASTER]: Accepted client connection

Process[3]: Received assigned client id as 3

client[3]: Waiting for the chopstick

3\_REQ\_3\_4\_1586820810

Copied value: 3\_REQ\_3\_4\_1586820810

Sending buffer: 3

Before mutex lock

AFter mutex lock

Waiting

Incoming buffer : 2\_REQ\_2\_3\_1586820806

REQ

REQ ---> REQ

parsed requests

Nope Its not REQ

Process [BROADCASTER]: Accepted client connection

Process[4]: Received assigned client id as 4

client[4]: Waiting for the chopstick

4\_REQ\_4\_0\_1586820814

Copied value: 4\_REQ\_4\_0\_1586820814

Sending buffer: 4

Before mutex lock

AFter mutex lock

Waiting

Incoming buffer : 3\_REQ\_3\_4\_1586820810

REQ

REQ ---> REQ

parsed requests

Nope Its not REQ

Incoming buffer : 4\_REQ\_4\_0\_1586820814

REQ

REQ ---> REQ

parsed requests

Nope Its not REQ

^C

2

Copied value: 2

2

Copied value: 2

Before mutex lock

AFter mutex lock

Waiting

2

Copied value: 2

Before mutex lock

AFter mutex lock

Waiting

Sending buffer: 2

Before mutex lock

AFter mutex lock

Waiting

]0;swapnil@Swapnil\_Acharya: /mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems/Finalized\_Projects/project4/code/distributed[01;32mswapnil@Swapnil\_Acharya[00m:[01;34m/mnt/d/OneDrive/Onedurive/Ms\_Computer\_Science\_Stuff/School\_Stuff/CSCI\_610\_Advanced\_Concepts\_In\_Operating\_Systems

s/Finalized\_Projects/project4/code/distributed[00m$ exit

exit

Script done on 2020-04-13 18:34:21-0500