UNIVERSITY OF MARYLAND BALTIMORE COUNTY

DEPARTMENT OF COMPUTER SCIENCE

ADVANCED OPERATING SYSTEMS

CMSC 621

PROJECT 2

SUBMITTED BY:

SUSHMITHA MANJUNATHA

ABSTRACT

In this project, I have tried to implement a n node distributed application that consists of following implementations of Berkeley Synchronization, Non-Casual ordered multicast and Casual ordered multicast.

SYSTEM DESIGN

Description of the functions used:

In-built functions used:

argc - It is the number of arguments passed into the program from the command line, including the name of the program.

argv[] - The array of character pointers is the listing of all the arguments provided in the command line.

stderr -Standard error is an output stream typically used by programs to output error messages or diagnostics. It is a stream independent of standard output and can be redirected separately.

gethostbyname() – This function returns a structure of type *hostent* for the given host *name*. Here *name* is either a hostname or an IPv4 address in standard dot notation.

atoi - Convert string to integer.

atof - Convert string to double.

sockaddr_in - The basic structure for all syscalls and functions that deal with internet addresses.

socket(domain,type,protocol) - The function socket() creates an endpoint for communication and returns a file descriptor for the socket. socket() takes three arguments: domain, type and protocol.

AF_INET- Indicates network protocol IPv4 (IPv4-only).

SOCK_STREAM – It is reliable stream-oriented service .

bind() - It assigns a socket to an address. When a socket is created using socket(), it is only given a protocol family, but not assigned an address. This association with an address must be performed with the bind() system call before the socket can accept connections to other hosts. bind() takes three arguments:

sockfd, a descriptor representing the socket to perform the bind on.

my_addr, a pointer to a sockaddr structure representing the address to bind to.

addrlen, a socklen_t field specifying the size of the sockaddr structure.

Bind() returns 0 on success and -1 if an error occurs.

listen() – After a socket has been associated with an address, listen() prepares it for incoming connections. However, this is only necessary for the stream-oriented (connection-oriented) data modes, i.e., for socket types (SOCK_STREAM, SOCK_SEQPACKET). listen() requires two arguments:

sockfd, a valid socket descriptor.

backlog, an integer representing the number of pending connections that can be queued up at any one time. The operating system usually places a cap on this value.

Once a connection is accepted, it is dequeued. On success, 0 is returned. If an error occurs, -1 is returned.

bzero(buff,n) - The bzero() function erases the data in the *n* bytes of the memory starting at the location pointed to by buff, by writing zeroes (bytes containing '\0') to that area.

sprint(char *str ,const char *format) - Composes a string with the same text that would be printed if *format* was used on printf, but instead of being printed, the content is stored as a *C string* in the buffer pointed by *str*.

write(int *fd*, const void **buf*, size_t *count*) - It writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

read(int *fd*, void **buf*, size_t *count*) - attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

sockfd - It is the socket descriptor returned by socket(). serv_addr is pointer to struct sockaddr that contains information on destination IP address and port. addrlen is set to sizeof(struct sockaddr)

sleep() – It makes the calling thread sleep until specified seconds have elapsed or a signal arrives which is not ignored.

accept() - When an application is listening for stream-oriented connections from other hosts, it is notified of such events and must initialize the connection using the accept() function. The accept() function creates a new socket for each connection and removes the connection from the listen queue. It takes the following arguments:

sockfd, the descriptor of the listening socket that has the connection queued.

cliaddr, a pointer to a sockaddr structure to receive the client's address information.

addrlen, a pointer to a socklen_t location that specifies the size of the client address structure passed to accept(). When accept() returns, this location indicates how many bytes of the structure were actually used.

The accept() function returns the new socket descriptor for the accepted connection, or -1 if an error occurs.

pthread_attr_init(&tattr)- This function initializes the thread attributes object pointed to by *attr* with default attribute values.

pthread_attr_setdetachstate((&tattr, int detachstate) – This function sets the detach state attribute of the thread attributes object referred to by t*attr* to the value specified in *detachstate*. The detach state attribute determines whether a thread created using the thread attributes object t*attr* will be created in a joinable or a detached state.

pthread_create() - This function starts a new thread in the calling process. The new thread starts execution by invoking *start_routine*(); *arg* is passed as the sole argument of *start_routine*().

pthread_mutex_init(&mutex,NULL) - This function shall initialize the mutex referenced by *mutex* with attributes specified by*attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the same as passing the address of a default mutex attributes object. Upon successful initialization, the state of the mutex becomes initialized and unlocked.

pthread_mutex_lock(&mutex) - The mutex object referenced by *mutex* is locked by calling *pthread_mutex_lock()*. If the mutex is already locked, the calling thread blocks until the

mutex becomes available. This operation returns with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

pthread_mutex_unlock(&mutex) - This function shall release the mutex object referenced by *mutex*. The manner in which a mutex is released is dependent upon the mutex's type attribute.

Assignment 1:

Design:

• Client-server architecture is used with TCP transport layer protocol for communication in Berkeley algorithm implementation.

• The time daemon first asks all the nodes present in a communication for their current clock value and sends its own clock value which is a random number generated by the time daemon(TimeDaemon.cpp).

• All other processes initially having their logical clock zero generates their logical clock as a random number and send the difference between their clock value and time daemon's received clock to the Time daemon or Server.

• The time daemon then after receiving the expected clocks' differences from all of the processes initiates the procedure of synchronization.

• It takes all the differences and computes the average of it. Then sends back the adjustment time to every node and also synchronizes its own clock value by adding this average to its own clock value.

• This is a single server and multiple clients implementation performing clock synchronization.

How to run the code:

Server side (TimeDaemon.cpp)

1. g++ TimeDaemon.cpp -o DS.out -lm -lpthread
2. ./DS.out <portnumber> <NumberOfProcesses>

3. i.e ./DS.out 9090 3

Client part: (cli.cpp)

1. g++ cli.cpp -o cli.out
2. ./cli <hostname> <portnumber> <Clock_value> <ProcessID>
   Ex: ./cli 127.0.0.1 9090 10 3

   OR

1.chmod 755 clocks.sh

2. ./clocks.sh

OUTPUT: TimeDaemon

```
husmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ clear

husmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ g++ TimeDaemon.cpp -o daemon -lpthread
husmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ ./daemon 8888
aemon process's initial clockValue:28
ifference in Clock value received-25
ifference in Clock value received-26
ifference in Clock value received-22
ifference in Clock value received-21
ifference in Clock value received-24
verage calculated: -4
aemon Process's new clockValue 24
riting to the buffer21
riting to the buffer22
riting to the buffer18
riting to the buffer17
riting to the buffer20
husmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$
```

Clients Output with synchronized clock values

```
shusmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ clear

shusmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ g++ cli.cpp -o client -lpthread
shusmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ chmod 755 clocks.sh
shusmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ ./clocks.sh
shusmitha@shusmitha-VirtualBox:~/AOS/OS-ME/Berkley$ ProcessID:2Initial Clock_Value of client:
2
ProcessID:4Initial Clock_Value of client:
ProcessID:3Initial Clock_Value of client:
7
3
ProcessID :4 Daemon's clock: 28
processID4Sending clock difference to daemon
ProcessID:1Initial Clock_Value of client:
-25
ProcessID :2 Daemon's clock: 28
processID2Sending clock difference to daemon
-26
6
ProcessID :1 Daemon's clock: 28
processID1Sending clock difference to daemon
-22
ProcessID :3 Daemon's clock: 28
processID3Sending clock difference to daemon
-21
ProcessID:0Initial Clock_Value of client:
4
ProcessID :0 Daemon's clock: 28
processID0Sending clock difference to daemon
-24
ProcessID :2 Received difference
22
ProcessID:2Syncronizing the Logical Clock :24
ProcessID :3 Received difference
17
ProcessID:3Syncronizing the Logical Clock :24
ProcessID :4 Received difference
21
ProcessID:4Syncronizing the Logical Clock :24
ProcessID :1 Received difference
18
ProcessID:1Syncronizing the Logical Clock :24
ProcessID :0 Received difference
20
ProcessID:0Syncronizing the Logical Clock :24
```

ASSIGNMENT-2

Without causally ordered multicasting:

- Implemented the non-causal ordered multicasting for the distributed system.
- Created two threads for each process, one for sending the multicast message to other nodes and one for listening to its communication port.
- Once a process delivers a received message to a user, it prints out the message on screen.
- Processes will not fail, join, or leave the distributed system.

How to run the code:

NonCasualOrdering (not_casual.cpp)

1. g++ not_casual.cpp -o not_casual.out -lm -lpthread
2. ./not_casual.out <9094> <9095> <9096>
3. ./not_casual.out <9095> <9094> <9096>
4. ./not_casual.out <9096> <9095> <9094>