

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINNERING BUNDELKHAND INSTITUTE OF ENGINEERING AND TECHNOLOGY, JHANSI

LAB FILE DISTRIBUTED SYSTEMS

**SUBMITTED BY:** AVINASH GUPTA B.TECH CSE 4TH YEAR 7TH SEMESTER

ROLL NO: 1704310018

# SUBMITTED TO:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Contents** | **Date** | **Signature** | **Remarks** |
| 1 | Simulate the functioning of Lamport’s Logical Clock in C. |  |  |  |
| 2 | Simulate the Distributed Mutual Exclusion in C. |  |  |  |
| 3 | Implement a Distributed Chat Server using  TCP Sockets in C. |  |  |  |
| 4 | Implement RPC mechanism for a file transfer across a network in C |  |  |  |
| 5 | Implement Java RMI mechanism for accessing methods of remote systems. |  |  |  |
| 6 | Simulate Balanced Sliding Window Protocol in C. |  |  |  |
| 7 | Implement CORBA mechanism by using C+  + program at one end and Java program  on the other |  |  |  |

**EXPERIMENT: 1**

**Object:** Simulate the functioning of Lamport’s Logical Clock in C.

# Aim: Simulate the functioning of Lamport’s Logical Clock in ‘C’.

**Theory: Logical Clocks** refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual time span. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

# Program:

#include<stdio.h> #include<conio.h> int max(int a,int b); int main()

{

int i,j,k,p1[20],p2[20],e1,e2,dep[20][20]; printf("\*\*\* Lamport's Logical Clock \*\*\*\ n"); printf("Enter the events : "); scanf("%d

%d",&e1,&e2); for(i=0;i<e1;i++)

p1[i]=i+1; for(i=0;i<e2;i+

+)

p2[i]=i+1;

printf("Enter the Dependency matrix:\n");

printf("\nEnter 1 if E1->E2 \nEnter -1, if E2->E1 \nElse Enter 0 \n\ n"); printf(" ");

for(i=0;i<e2;i++) printf(" e2%d",i+1);

for(i=0;i<e1;i++)

{ printf("\ne1%d ",i+1);

for(j=0;j<e2;j++){ scanf("%d",&dep[i][j]);

}

}

for(i=0;i<e1;i++)

{ for(j=0;j<e2;j+

+){

//change the Time stamp if dependency exist if(dep[i][j]==1){

p2[j]=max(p2[j],p1[i]

+1); for(k=j;k<e2;k++) p2[k+1]=p2[k]+1;}

//change the Time stamp if dependency exist if(dep[i][j]==-1){

p1[i]=max(p1[i],p2[j]

+1); for(k=i;k<e1;k++) p2[k+1]=p1[k]+1;

}

}

}

//to print the outcome of Lamport Logical Clock printf("\nP1 : ");

for(i=0;i<e1;i++)

{ printf("%d",p1[i

]);

}

printf("\nP2 : "); for(j=0;j<e2;j++) printf("%d",p2[j])

; getch(); return 0 ;

}

//to find the maximum timestamp between two events int max(int a, int b)

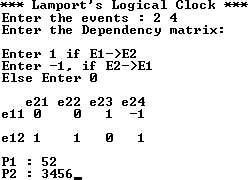
{

if (a>b) return a;

else return b;

}

# OUTPUT:



**EXPERIMENT: 2**

**Object:** Simulate the Distributed Mutual Exclusion in C.

**Theory: Distributed Mutual Exclusion Algorithm** is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems. In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.

In this algorithm:

* Three types of messages (REQUEST, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
* A site sends a **REQUEST** message to all other site to get their permission to enter critical section.
* A site sends a **REPLY** message to requesting site to give its permission to enter the critical section.
* A site sends a **RELEASE** message to all other site upon exiting the critical section.
* Every site Si, keeps a queue to store critical section requests ordered by their timestamps.

**request\_queuei** denotes the queue of site Si

* A timestamp is given to each critical section request using Lamport’s logical clock.
* Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

# Algorithm:

* **To enter Critical section:**
  + When a site Si wants to enter the critical section, it sends a request message **Request (tsi, i)** to all other sites and places the request on **request\_queuei**. Here, Tsi denotes the timestamp of Site Si.
  + When a site Sj receives the request message **REQUEST (tsi, i)** from site Si, it returns a times tamped REPLY message to site Si and places the request of site

Si on **request\_queuej.**

.

# To execute the critical section:

* + A site Si can enter the critical section if it has received the message with timestamp larger than **(tsi, i)** from all other sites and its own request is at the top

# of request\_queuei.

* **To release the critical section:**
  + When a site Si exits the critical section, it removes its own request from the top of its request queue and sends a times tamped **RELEASE** message to all other sites.
  + When a site Sj receives the times tamped **RELEASE** message from site Si, it removes the request of Si from its request queue.

# Program:

#include<stdio.h> #include<conio.h

> #include<dos.h> #include<time.h> void main()

{

int cs=0,pro=0; double run=5; char key='a'; time\_t t1,t2; clrscr();

printf("Press a key(except q) to enter a proc ess into critical section.");

printf("

\

nPress q at any time to exit."); t1 = time(NULL)

-

5;

while(key!='q')

{

while(! kbhit()) if(cs!

=0)

{

t2 = time(NULL); if(t2

-

t1 > run)

{

printf("Process%d ",pro

- 1);

printf(" exits crit ical section.

\n"); cs=0;

}

}

key = getch(); if(key!='q')

{

if(cs!=0)

printf("Error: Another process is currently executing critical section Please wait till its execution is over.

\ n");

else

{

printf("Proce ss %d ",pro);

printf(" entered critical section

\ n");

cs=1; pro+

+;

t1 = time(NULL);

}

}

}

}

# Output:

Press a key(except q) to enter a process into critical section. Press q at any time to exit.

Process 0 entered critical section.

Error: Another process is currently executing critical section. Please wait till its execution is over.

Process 0 exits critical section. Process 1 entered critical section. Process 1 exits critical section.

Process 2 entered critical section.

Error: Another process is currently executing critical section. Please wait till its execution is over.

Process 2 exits critical section

# EXPERIMENT: 3

**Object:** Implement a Distributed Chat Server using TCP Sockets in C.

**Theory:** TCP is a connection-oriented protocol that provides a reliable. Flow of data between two computers.

Example applications that. Use such services are HTTP, FTP, and Telnet.

# Program:

event.c/

#include <sys/time.h> #include <string.h> #include <stdio.h> #include "event.h"

void init\_fdvec(fdvec \*e)

{

FD\_ZERO(&e->fds);

memset(&e->f, '\0', sizeof(e-

>f)); e->size = 0;

}

void init\_eventset(eventset \*e)

{

init\_fdvec(&e->read

);

init\_fdvec(&e->write);

}

void on\_event(fdvec \*e, int fd, void (\*f)(int fd))

{

FD\_SET(fd,&e-

>fds); e->f[fd] = f;

if (fd >= e->size) e->size = fd + 1;

}

void on\_event\_nop(fdvec \*e, int fd)

{

int i; FD\_CLR(fd,&e-

>fds); e->f[fd] = NULL;

if (fd == e->size-1)

{ e->size = 0;

for (i = 0; i != fd; i++) {

if (FD\_ISSET(i, &e->fds)) e->size = i + 1;

}

}

}

void handle\_events(eventset \*e)

{

fd\_set readfds, writefds;int maxfd; int i;

int n othing\_to\_write = 1;

readfds = e->read.fds; writefds = e->write.fds;

maxfd = (e->read.size > e->write.size) ? e->read.size : e-

>write.size; select(maxfd, &readfds, &writefds, 0, 0);

for (i = 0; i != maxfd; i+

+) { if (FD\_ISSET(i, &

writefds) && FD\_ISSET(i, &e->write.fds)) {

/\* fprintf(stderr, "%d writable\n", i); \*/ e->write.f[i](i);

nothing\_to\_write = 0;

}

}

if (nothing\_to\_write) {

for (i = 0; i != maxfd; i++) {

if (FD\_ISSET(i, &readfds) && FD\_ISS ET(i, &e->read.fds)) {

/\* fprintf(stderr, "%d readable\n", i); \*/ e->read.f[i](i);

}

}

}

}

event.h/ typedef struct

{

fd\_set fds;

void (\*f[FD\_SETSIZE])(int fd); int size;

} fdvec; typedef struct

{

fdvec read;

fdvec write;

} eventset;

void init\_eventset(eventset \*e);

void on\_event(fdvec \*e, int fd, void (\*f)(int fd)); void on\_event\_nop(fdvec \*e, int fd);

void handle\_events(eventset \*e); die.c/

#include <stdio

.h>

#include <string.h>#include <errno.h>

void die\_if\_func(int whether, char \*cond, char \*file, int line, char \*msg)

{

if (whether)

{ char \*s;

for (s = msg; \*s; ++s)

{ if (\*s != '%') {

putc(\*s, stderr);

} else {

++s;

switch(\*s)

{

case '\0':

fprintf(stderr, "(Unterminated %% sequence in error string)\n"); goto done\_with\_msg;

case '%':

putc('%', stderr); break;

case 'f':

fprintf(stderr, "%s", file); break;

case 'l':

fprintf(stderr, "%d", line); break;

case 'c':

fprintf(stderr, "%s", cond); break;

case 'e':

fprintf(stderr, "%s", strerror(errno)); break;

default:

fprintf(stderr, "(invalid %% sequence %%%c in error string)\n",

\*s); break;

}

}

}

done\_with\_msg:

putc('\n', stderr); fflush(stderr); exit(1);

}

}

char \*out\_of\_memory = "Out of memory at %f:%l (says %c) (error

%e)"; die\_test.c/ #include "die.h" int main()

{die\_if(1, out\_of\_memory); return 0;

}

die.h/ `

#define die\_if(cond,msg) (die\_if\_func(cond,#cond, FILE , LINE

,msg))

void die\_if\_func(int whether, char \*cond, char \*file, int line, char \*msg); char \*out\_of\_memory;

kstr.c/

#include <string.h> #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include "die.h"

#include "kstr.h" #include "talloc.h" void kstr\_new(kstr

\*k)

{

\*k = talloc(sizeof(\*\*k)); die\_if(!\*k, out\_of\_memory); (\*k)->start = 0;

(\*k)->length = 0;

(\*k)->allocated\_length = 0;

}

void kstr\_del(kstr k)

{

tfree(k->start); tfree(k);

}

void kstr\_growto(kstr k, int len)

{

if (len > k->allocated\_length)

{ int nal = ((len | 7) + 1) \* 2; char \*nstart = talloc(nal); die\_if(!nstart, out\_of\_memory); memset(nstart, 'Y', nal);

memcpy(nstart, k->start, k->length); tfree(k->start);

k->start = nstart;

k->allocated\_length = nal;

}

}

void kstr\_growby(kstr k, int len)

{

kstr\_growto(k, len + k->length);

}void kstr\_getline(kstr k, FILE \*f)

{

int l = 80;

k->length = 0; for (;;) {

char \*rv; kstr\_growby(k, l); clearerr(f);

rv = fgets(k->start + k->length, l, f); if (!rv) {

return;

}

k->length += strlen(k->start + k-

>length); if (k->start[k->length-1] == '\')

{

/\* end of line \*/

k->start[k->length] = 'X'; return;

}

l \*= 2;

}

}

int kstr\_read(kstr k, int fd, int maxlen)

{

int rv;

kstr\_growto(k, maxlen); rv = read(fd, k->start, maxlen); if (rv <= 0) {

k->length = 0; return rv;

} else {

k->length = rv; return rv;

}

}

void kstr\_append(kstr k, char \*s, int len)

{

kstr\_growby(k, len);

memcpy(k->start + k->length, s, len); k->length += len;

}

kstr.h/ typedef struct

{

char \*start; int length;

int allocated\_length;

} \*kstr;void kstr\_new(kstr \* k); void kstr\_del(kstr k);

void kstr\_growto(kstr k, int len);

void kstr\_growby(kstr k, int len); void kstr\_getline(kstr k, FILE \*f);

int kstr\_read(kstr k, int fd, int maxlen); void kstr\_append(kstr k, char \*s, int len); kstr\_test.c/

#include <stdio.h> #include "kstr.h" #include "die.h"

char \*input\_error = "input error at %f:%l:

%e"; int main()

{

kstr s; kstr\_new(&s);

while (!feof(stdin))

{ kstr\_getline(s, stdin); die\_if(ferror(stdin), input\_error); fwrite(s->start, s->length, 1, stdout);

}

kstr\_del(s); return 0;

}

talloc.c/

#include

<stdlib.h> #include <stdio.h> #include "talloc.h"

/\* to turn on tracing:

#define tracing /\* \*/ void \*talloc(int n)

{

void \*rv = malloc(n); #ifdef tracing

fprintf(stderr, "0x%08x: %d bytes\n", (unsigned)rv, n); #endif

return rv;

}

void tfree(void \*p)

{

#ifdef tracing

fprintf(stderr, "0x%08x: freed\n", (unsigned)p); #endif

free(p);

}

talloc.h/void \*talloc(int n); void tfree(void \*p); mem-used/#!/var/u/sittler/bin/perl-w

use strict;

# analyze memory usage trace from talloc. my %blocks;

my $total = 0; while (<>) {

printf "%9d %s", $total, $\_; if (/^(0x[0-9a-f]+):

(\d+) bytes$/) {

if (exists $blocks{$1}) {

warn "Uh-oh: $1 allocated twice without intervening free\n";

} else {

$blocks{$1} = $2;

$total += $2;

}

} elsif (/^(0x[0-9a-f]

+): freed$/) {

next if $1 eq '0x00000000';

$total -=

$blocks{$1}; delete

$blocks{$1};

}

}

Print "Final: $total\n"; chat-serve

r.c/

#include <sys/types.h> #include <sys/socket.h> #include <errno.h> #include <stdio.h> #include <netinet/in.h> #include <arpa/inet.h> #include <unistd.h> #include <fcntl.h> #include <signal.h> #include "event.h" #include "kstr.h" #include "die.h"

eve ntset e;

kstr client\_list;

typedef struct client\_info

{

int connected;

struct sockaddr\_in sin; kstr outbuf;

int outbufp;

} client\_info;

client\_info \*get\_cip(int fd)

{

return ((client\_info\*)client\_list->start) + fd;

}

void handle\_disconnectio n(int fd)

{

client\_info \*cip = get\_cip(fd); cip->connected

= 0; kstr\_del(cip->outbuf); on\_event\_nop(&e.read, fd); on\_event\_nop(&e.write, fd); close(fd);

}

void write\_queued\_data(int fd)

{

client\_info \*cip = get\_cip(fd); int rv;

die\_if(!cip->connected, "Damn event handler called on disconnected client"); die\_if(cip->outbufp > cip->outbuf->length, "outbufp out of range (%c)");

rv = write(fd, cip->outbuf->start + cip->outbufp, cip->outbuf->length -cip-

>outbufp); if (rv < 0) {

fprintf(stderr, "error writing to client %d (%s): ", fd, net\_ntoa(cip->sin.sin\_addr)); perror("closing connection");

handle\_disconnection(fd);

} else {

cip->outbufp += rv;

if (cip->outbufp == cip->outbuf->length)

{

cip->outbufp = 0;

cip->outbuf->length = 0; on\_event\_nop(&e.write, fd);

} else {

if (cip->outbufp > 15\*cip->outbuf->length/16) {

/\* time to move it back to the beginning of the buffer \*/

memcpy(cip->outbuf->start, cip->outbuf->start+cip->outbufp, cip->outbuf->length -cip-

>outbufp);

cip->outbuf->length -= cip-

>outbufp; cip->outbufp = 0;

}

}

}

}

char lostmsg[] = "(Lost messages)\r\n"; int queuelimit = 50 \* 1024;void queue\_data(int fd, char \*s, int len)

{

client\_info \*cip = get\_cip(fd);

die\_if(!cip->connected, "Attempt to send to disconnected client"); if (cip->outbuf->length + len > queuelimit) {

if (cip->outbuf->length < queuelimit)

{ kstr\_append(cip->outbuf, lostmsg, sizeof(lostmsg)- 1);

} else {

}

} else {

kstr\_append(cip->outbuf, s, len);

}

on\_event(&e.write, fd, write\_queued\_data);

}

void queue\_string(int fd, char \*s)

{

queue\_data(fd, s, strlen(s));

}

kstr rbuf;

void handle\_client\_data(int fd)

{

int rv;

rv = kstr\_read(rbuf, fd, 8192); if (rv < 0) {

fprintf(stderr, "client fd %d:", fd); perror("read error");

} else if (rv == 0)

{ handle\_disconnection(fd

);

} else

{ int i;

client\_info \*cip = get\_cip(0); for (i = 0; i != e.

read.size; i++) {

if (cip[i].connected)

{ queue\_string(i, "From ");

queue\_string(i, inet\_ntoa(cip[fd].sin.sin\_addr)); queue\_string(i, ": ");

queue\_data(i, rbuf->start, rbuf->length);

}

}

}

}

void new\_client\_conn(int listenfd)

{

struct sockaddr\_in addr; socklen\_t socklen = sizeof(addr); client\_info \*cip;int space\_to\_allocate;

int nc = accept(listenfd, (struct sockaddr\*)&addr, &socklen); fcntl(nc, F\_SETFL, fcntl(nc, F\_GETFL, 0) | O\_NDELAY); kstr\_growto(client\_list, (nc+1) \* sizeof(struct client\_info));

space\_to\_allocate = (nc+1) \* sizeof(struct client\_info) -client\_list->length; memset(client\_list->start + client\_list->length, '\0', space\_to\_allocate); client\_list->length += space\_to\_allocate;

cip = ((client\_info\*)client\_list->start) + nc; cip->connected = 1;

cip->sin = addr; kstr\_new(&cip-

>outbuf); cip->outbufp

= 0;

on\_event(&e.read, nc, handle\_client\_data); queue\_string(nc, "Hello there "); queue\_string(nc, inet\_ntoa(addr.sin\_addr)); queue\_string(nc, "\n");

}

int open\_server\_socket()

{

int fd = socket(PF\_INET, SOCK\_STREAM, 0); int rv;

int one = 1;

struct sockaddr\_in addr;

setsockopt(fd, SOL\_SOCKET, SO\_REUSEADDR, &one, sizeof one); memset((char\*)&addr, '\0', sizeof(addr));

addr.sin\_family = AF\_INET; addr.sin\_port = htons(17224); addr.sin\_addr.s\_addr = INADDR\_ANY;

rv = bind(fd, (struct sockaddr\*)&addr, sizeof(addr)); die\_if(rv<0, "bind failed: %e");

rv = listen(fd, 5); die\_if(rv<0, "listen failed:

%e"); return fd;

}

void end\_server(int fd)

{

kstr\_del(client\_list); kstr\_del(rbuf); exit(0);

}

int main()

{

int s = open\_server\_socket(); kstr\_new(&client\_list); kstr\_new(&rbuf); sigignore(SIGPIPE); init\_eventset(&e);on\_even

t(&e.read, s, new\_client\_conn); on\_event(&e.read, 0, end\_server); for (;;) {

handle\_events(&e);

}

die\_if(1 + 1 == 2, "Can't happen at %f:

%l"); return 0;

}

# EXPERIMENT: 4

**Object:** Implement RPC mechanism for a file transfer across a network in C.

**Theory: RPC** is a request–response protocol. An **RPC** is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process.

**Theory: RPC** is a request–response protocol. An **RPC** is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process.

# Program:

client.java import java.io.\*;

import java.net.\*; class client{

public static void main(String args[])

{ try{

Socket sock=new Socket (args[0],8081); FileInputStream is=new FileInputStream("client.class"); OutputStream os=sock.getOutput

Stream(); int ch=0; ch=is.read()

; do{ os.write(ch);

ch=is.read();

}while(ch!=- 1); os.flush();

os.close();

sock.close();

}

catch(Exception e){System.out.println(e);}

}

}

server.java import java.io.\*; import java.net.\*; class

server { public static voi

d main(String args[])

{ new server().go();

}

public void go(){while(true){ try{

ServerSocket server=new ServerSocket(8081); Socket socket=server.accept();

new Thread(new thread(socket)).start();

}

catch(Exception e){

}

}

}

class thread implements Runnable{

Socket s; thread(Socket s){ this.s=s;

}

public void run()

{ try{

InputStream is=s.getInputStream();

FileOutputStream out =new FileOutputStream(new File("clientcopy.class")); int ch=0;

ch=is.read(); do{ out.write( ch); ch=is.read();

}while(ch!=- 1); out.flush();

System.out.println("File (client.class) Copied to server as (clientcopy.class)");

out.close();

s.close();

}

catch(Exception e)

{ System.out.println(e

);

}

}

}

}

# EXPERIMENT 5

**Object**: Implement Java RMI mechanism for accessing methods of remote systems.

# Theory:

Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side).

Working of RMI:

The communication between client and server is handled by using two intermediate objects: S tub object (on client side) and Skeleton object (on server side).

# Program:

AddClient.java import java.rmi.\*; public class AddClient

{

public static void main(String args[])

{

try

{

String addServerURL="rmi://"+ args[0] + "/AddServer"; AddServerIntf addServerIntf = (AddServerIntf)Naming.lookup(addServerURL); System.out.println("the first no is:" + args[1]); double d1=Double.valueOf(args[1]).doubleValue(); System.out.println("the second no is:" + args[2]); double d2=Double.valueOf(args[2]).doubleValue();

System.out.println("Sum = " + addServerIntf.add(d1,d2));

}

catch(Exception e)

{

System.out.println("Exception:" +e);

}

}

}

AddServer.java import java.net.\*; import java.rmi.\*; public class AddServer

{

public static void main(String args[]){try

{

AddServerImpl addServerImpl = new AddServerImpl(); Naming.rebind("AddServer", addServerImpl);

}

catch(Exception e)

{

System.out.println("Exception:" +e);

}

}

}

AddServerImpl.jav a import java.rmi.\*; import java.rmi. server.\*;

public class AddServerImpl extends UnicastRemoteObject implements AddServerIntf

{

public AddServerImpl() throws RemoteException

{

}

public double add(double d1,double d2) throws RemoteException

{

return d1+d2;

}

}

AddServerIntf.jav a import java.rmi.\*;

public interface AddServerIntf extends Remote

{

double add(double d1, double d2) throws RemoteException;

}

# Output:

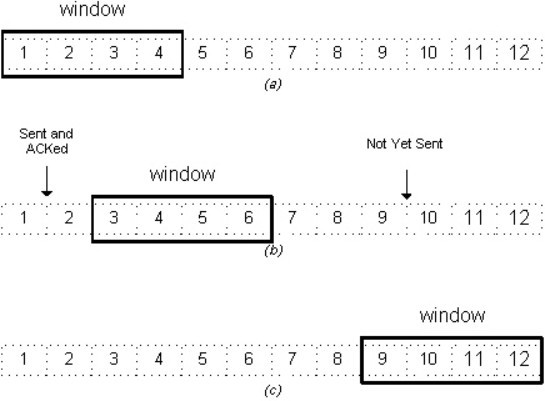
/ when arguments are passed as 35 and 16 Sum = 51

# EXPERIMENT: 6

**Object:** Simulate Balanced Sliding Window Protocol in C

**Theory: I**n computer networks sliding window protocol is a method to transmit data on a network. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames. In Networking, Window simply means a buffer which has data frames that needs to be transmitted.

As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received, for eg:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.



# Efficiency of Sliding Window Protocol

η = (W\*tx)/(tx+2tp)

W = Window Size

tx = Transmission time tp = Propagation delay

Sliding window works in full duplex mode

# Program:

#include<stdio.h>

int main()

{

int w,i,f,frames[50];

printf("Enter window size:

"); scanf("%d",&w);

printf("\nEnter number of frames to transmit: "); scanf("%d",&f);

printf("\nEnter %d frames: ",f); for(i=1;i<=f;i++)

scanf("%d",&frames[i]);

printf("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n\n");

printf("After sending %d frames at each stage sender waits for acknowledgement sent by the receiver\n\n",w);

for(i=1;i<=f;i++)

{

if(i%w==0)

{

printf("%d\n",frames[i]);

printf("Acknowledgement of above frames sent is received by sender\n\n");

}

else

printf("%d ",frames[i]);

}

if(f%w!=0)

printf("\nAcknowledgement of above frames sent is received by sender\n");

return 0;

}

# Output:

Enter window size: 3

Enter number of frames to transmit: 5 Enter 5 frames: 12 5 89 4 6

With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver

12 5 89

Acknowledgement of above frames sent is received by sender 4 6

Acknowledgement of above frames sent is received by sender

# EXPERIMENT: 7

**Object:** Implement CORBA mechanism by using C++ program at one end and Java program on the other

# Program:

Server programs: #ifndef hello\_skel\_h

#define

hello\_skel\_h

#include

<hello.h>

class Hello\_skel : virtual public Hello, virtual public CORBA\_Object\_skel

{

static CORBA\_ULong \_ob\_num\_; Hello\_skel(const Hello\_skel&); void operator=(const Hello\_skel&); protected:

Hello\_skel() { } Hello\_skel(const char\*); public:

Hello\_ptr \_this() { return

Hello::\_duplicate(this); } virtual CORBA\_ULong

\_OB\_incNumber() const;

virtual OBDispatchStatus \_OB\_dispatch(const char\*, OBFixSeq< CORBA\_Octet

>&, bool, CORBA

\_ULong, CORBA\_ULong);

};

#endif

#include

<OB/CORBA.h>

#include <hello\_skel.h>

CORBA\_ULong Hello\_skel::\_ob\_num\_ = 0; Hello\_skel::Hello\_skel(const char\* name)

{

assert\_nca(name, OBNCANullString); try

{

\_OB\_cre ateObjectKeyWithName(name);

}

catch(...)

{

\_OB\_setRef(0)

; throw;}

}

CORBA\_ULong Hello\_skel::\_OB\_incNumber() const

{

return Hello\_skel::\_ob\_num\_++;

}

OBDispatchStatus Hello\_skel::\_OB\_dispatch(const char\*

\_ob\_op, OBFixSeq< CORBA\_Octet >&

\_ob\_seq, bool \_ob\_sw,

CORBA\_ULong

\_ob\_offIn, CORBA\_ULong

\_ob\_offOut)

{

if(strcmp(\_ob\_op, "hello") == 0)

{

hello();

CORBA\_ULong \_ob\_cnt = \_ob\_offOut;

\_ob\_seq.length(0);

\_ob\_seq.length(\_ob\_cnt); #ifdef OB\_CLEAR\_MEM

memset(\_ob\_seq.data(), 0, \_ob\_seq.length()); #endif

return OBDispatchStatusOK;

}

else

return CORBA\_Objec t\_skel::\_OB\_dispatch(\_ob\_op, \_ob\_seq, \_ob\_sw,

\_ob\_offIn, \_ob\_offOut);

}

#ifndef hello\_h

#define hello\_h

class Hello;

typedef Hello\* Hello\_ptr; typedef Hello\* HelloRef;

typedef OBObjVar< Hello > Hello\_var; class Hello : virtual public CORBA\_Object

{

Hello(const Hello&);

void operator=(const Hello&);protected:

Hello() {

} public:

static inline Hello\_ptr

\_duplicate(Hello\_ptr p)

{

CORBA\_Object::\_duplicate(p

); return p;

}

static inline Hello\_ptr

\_nil()

{

return 0;

}

static Hello\_ptr \_narrow(CORBA\_Object\_ptr); virtual void\* \_OB\_narrowHelp(const char\*) const; virtual const char\* \_OB\_typeId() const; friend void OBUnmar

shal(Hello\_ptr&, const CORBA\_Octet\*&, bool);

friend CORBA\_Boolean operator>>=(const CORBA\_Any&, Hello\_ptr&); virtual void hello();

};

extern const OBTypeCodeConst \_tc\_Hello;

inline void

CORBA\_release(Hello\_ptr p)

{

CORBA\_release((CORBA\_Object\_ptr)p);

}

inline CORBA\_Boolean CORBA\_is\_nil(Hello\_ptr p)

{

return p == 0;

}

inline void

OBMarshal(Hello\_ptr p, CORBA\_Octet\*& oct)

{

OBMarshal((CORBA\_Object\_ptr)p, oct);

}

inline v oidOB Marsh alCoun t(Hello

\_ptr p, CORB A\_UL

ong& count)

{

OBMarshalCount((CORBA\_Object\_ptr)p, count);

}

void OBUnmarshal(Hello\_ptr&, const CORBA\_Octet\*&, bool); void operator<<=(CORBA\_Any&, Hello\_ptr);

void operator<<=(CORBA\_Any&, Hello\_ptr\*);

CORBA\_Bool

ean operator>>=(const CORBA\_Any&, Hello\_ptr&); inline void

operator<<=(CORBA\_Any\_var& any, Hello\_ptr val)

{

any.inout() <<= val;

}

inline void

operator<<=(CORBA\_Any\_var& any, Hello\_ptr\* val)

{

any.inout() <<= val;

}

inline CORBA\_Boolean

operator>>=(const CORBA\_Any\_var& any, Hello\_ptr& val)

{

return any.in() >>= val;

}

#endif

#include <OB/CORBA.h> #include

<OB/TemplateI.h> #include <hello.h>

#ifndef HAVE\_NO\_EXPLICIT\_TEMPLATES

template class OBObjVar< Hello

>; templa

te class OBObjForSeq< Hello

>; #endif Hello\_ptr

Hello::\_narrow(CORBA\_Object\_ptr p)

{

if(!CORBA\_is\_nil(p))

{

void\* v = p ->

\_OB\_narrowHelp("IDL:Hello:1.0"); if(v) return \_duplicate((Hello\_ptr)v);

if(p -> \_OB\_remoteIsA("IDL:Hello:1.0"))

{

Hello\_ptr val = new Hello;val -> \_OB\_copyFrom(p); return val;

}

}

return \_nil();

}

void\*

Hello::\_OB\_narrowHelp(const char\* \_ob\_id) const

{

if(strcmp("IDL:Hello:1

.0", \_ob\_id) == 0) return (void\*)this; else

return CORBA\_Object::\_OB\_narrowHelp(\_ob\_id);

}

const char\* Hello::\_OB\_typeId() const

{

return "IDL:Hello:1.0";

}

void

OBUnmarshal(Hello\_ptr& val, const CORBA\_Octet\*& coct, bool swap)

{

Hello\_var old = val; CORBA\_Object\_var p; OBUnmarshal(p.inout(), coct, swap); if(!CORBA\_is\_nil(p))

{

void\* v = p->

\_OB\_narrowHelp("IDL:Hello:1.0"); if(v)

val = Hello::\_duplicate((Hello\_ptr)v); else

{

assert\_nca(!(p -> \_is\_local() && p ->

\_is\_dynamic()), OBNCADynamicAsStatic);

assert(!p ->

\_is\_local()); val = new Hello;

val -> \_OB\_copyFrom(p);

}

}

else

val = Hello::\_nil();

}

const OBTypeCodeConst

\_tc\_Hello( "010000000E00000022000000010000000E00000049444C3A48656C6C6F3A312E 300000000 6000""00048656C6C6F00");

void

operator<<=(CORBA\_Any& any, Hello\_ptr val)

{

OBObjAny\* o = new OBObjAny;

* -> b = CORBA\_Object::\_duplicate(val);
* -> d = CORBA\_Object::\_duplicate(val); any.replace(\_tc\_Hello, o, true);

}

void

operator<<=(CORBA\_Any& any, Hello\_ptr\* val)

{

OBObjAny\* o = new OBObjAny;

* -> b = \*val;
* -> d = CORBA\_Object::\_duplicate(\*val); any.replace(\_tc\_Hello, o, true);

}

CORBA\_Boolean

operator>>=(const CORBA\_Any& any, Hello\_ptr& val)

{

if(any.check\_type(\_tc\_Hello))

{

OBObjAny\* o = (OBObjAny\*)any.value(); assert(o);

if(!CORBA\_is\_nil(o -> d)){ void\*v = o -> d ->

\_OB\_narrowHelp("IDL:Hello:1.0"); if(v)

val = (Hello\_ptr)v; else

{

assert\_nca(!(o -> d -> \_is\_local() && o -> d -> \_is\_dynamic()), OBNCADynamicAsStatic); assert(!o -> d -> \_is\_local());

val = new Hello;

val -> \_OB\_copyFrom(o -> d); OBObjAny\* no = new OBObjAny;

no -> b = CORBA\_Object::\_duplicate(o -> b); no -> d = val;

((CORBA\_Any&)any).replace(\_tc\_Hello, no, true);

}

}

else

val = Hello::\_nil(); return true;

}

else

return false;

}

void Hello::hello(

)

{

if(CORBA\_is\_nil(\_ob\_con

\_)) throw CORBA\_N O\_IMPLEMENT();

CORBA\_ULong \_ob\_off = \_ob\_con\_ -> offset(this, "hello"); CORBA\_ULong \_ob\_cnt = \_ob\_off;

OBFixSeq< CORBA\_Octet > \_ob\_seq(\_ob\_cnt);

\_ob\_seq.length(\_ob\_cnt); #ifdef OB\_CLEAR\_MEM

memset(\_ob\_seq.data(), 0, \_ob\_seq.length()); #endif

bool \_ob\_sw, \_ob\_ex, \_ob\_fo;

\_ob\_off = \_ob\_con\_ -> request(this, "hello", \_ob\_seq, \_ob\_sw, \_ob\_ex, \_ob\_fo, \_ob\_tout\_); if(\_ob\_fo)

{

const CORBA\_Octet\* \_ob\_co = \_ob\_seq.data() + \_ob\_off;

\_OB\_forward(\_ob\_co,

\_ob\_sw); hello(); return;

}

if(\_ob\_ex)

throw CORBA\_UNKNOWN();

}

#include <hello\_skel.h>

class Hello\_impl : public Hello\_skel

{

public:

Hello\_impl(); virtual void hello();

};

#include <CORBA.h> #include <hello\_impl.h> Hello\_impl::Hello\_impl ()

{

}

void Hello\_impl::hello(

)

{

cout << "Hello World!" << endl;

}

#include <CORBA.h> #include

<hello\_impl.h> #include <fstream.h> int

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

{

CORBA\_ORB\_var orb = CORBA\_ORB\_init(argc, argv);CORBA\_BOA\_var boa = orb -> BOA\_init(argc, argv); Hello\_var p = new Hello\_impl;

CORBA\_String\_var s = orb -> object\_to\_string(p); const char\* refFile = "Hello.ref";

ofstream out(refFile); out << s << endl; out.close();

boa -> impl\_is\_ready(CORBA\_ImplementationDef::\_nil());

}

Client programs:

public interface Hello extends org.omg.CORBA.Object

{

void hello();

public void hello(); }

abstract public class \_sk\_Hello extends org.omg.CORBA.portable.Skeleton implements Hello

{ protected \_sk\_Hello(java.lang.String name)

{

super(name); }

protected \_sk\_Hello() { super(); }

public java.lang.String[] \_ids() { return ids; }

private static java.lang.String[] ids = { "IDL:Hello:1.0" }; public org.omg.CORBA.portable.

MethodPointer[] \_methods()

{ org.omg.CORBA.portable.MethodPointer[] methods =

{ new org.omg.CORBA.portable.MethodPointer("hello", 0, 0), }; return methods; }

public boolean \_execute(org.omg.CORBA.portable.MethodPointer method, org.omg.CORBA.portable.InputStream input, org.omg.CORBA.portable.OutputStream output) { switch(method.interface\_id)

{

case 0:

{

return \_sk\_Hello.\_execute(this, method.method\_id, input, output);

}

}

throw new org.omg.CORBA.MARSHAL(); } public static boolean \_execute(Hello \_s elf, int

\_method\_id,

org.omg.CORBA.portable.InputStream \_input, org.omg.CORBA.portable.OutputStream

\_output)

{

switch(\_method\_id) { case 0: { \_self.hello(); return false; } } throw new org.omg.CORBA.MARSHAL(); } }

class hello\_client

{ public static vo

id main( String args[] ) {try{ System.out.println( "Initializing the orb."); org.omg.CORBA.ORB orb =

org.omg.CORBA.ORB.init(); IORHolder ior\_holder = new IORHolder();

String iorString = ior\_holder.readIORFile( "Hello.ref"

);

org.omg.CORBA.Object object = orb.string\_to\_object( iorString ); Hello hello = HelloHelper.narrow( object );

hello.hello();

} catch ( org.omg.CORBA.SystemException e ) {

System.err.println( "System Exception ");

System.err.println( e );

}

}}