

INDEX

<u>Sr. No.</u>	<u>Topic</u>	<u>Date</u>	<u>Signature</u>
1	<i>Write a program to simulate the functioning of Lamport's logical clock in 'C'.</i>		
2	<i>Write a program to simulate the Distributed Mutual Exclusion in 'C'.</i>		
3	<i>Write a program to implement a Distributed chat server using TCP sockets in 'C'.</i>		
4	<i>Implement RPC mechanism for a file transfer across a network in 'C'.</i>		
5	<i>Write a JAVA code to implement 'Java RMI' mechanism for accessing methods of remote systems.</i>		
6	<i>Write a code in 'C' to implement sliding window protocol</i>		
7	<i>Implement corba mechanism by using c++ program at one end and java program at the other</i>		

Experiment– 1

OBJECTIVE: Write a program to simulate the functioning of Lamport's logical clock in 'C'.

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

void main()
{
    int i,j,k;
    int x=0;
    char a[10][10];
    int n,num[10],b[10][10];
    clrscr();

    printf("Enter the no. of physical clocks: ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("\nNo. of nodes for physical clock %d",i+1);
        scanf("%d",&num[i]);

        x=0;
        for(j=0;j<num[i];j++)
        {
            printf("\nEnter the name of process: ");
            scanf("%s",&a[i][j]);
            b[i][j]=x + rand() % 10;
            x=b[i][j]+1;
        }
    }

    printf("\nPress a key for watching timestamp of physical clocks");
    getch();

    clrscr();
    for(i=0;i<n;i++)
    {
        printf("Physical Clock %d",i+1);
        for(j=0;j<num[i];j++)
        {
            printf("\nProcess %c",a[i][j]);
            printf(" has P.T. :%d ",b[i][j]);
            printf("\n");
        }
    }

    printf("Press a key for watching timestamp of logical clocks");
    getch();

    clrscr();
    x=0;
    for(i=0;i<10;i++)
```

```
    for(j=0;j<n;j++)
        for(k=0;k<num[j];k++)
            if(b[j][k]==i)
            {
                x = rand() % 10 + x;
                printf("Logical Clock Timestamp for process %c",a[j][k]);
                printf(":%d ",x);
                printf("\n");
            }
    getch();
    return;
}
```

Output

Enter the no. of physical clocks: 2

No. of nodes for physical clock 1: 2

Enter the name of process: a

Enter the name of process: b

No. of nodes for physical clock 2: 2

Enter the name of process: c

Enter the name of process: d

Press a key for watching timestamp of physical clocks

Physical Clock 1

Process a has P.T.: 6

Process b has P.T.: 7

Physical Clock 2

Process c has P.T.: 2

Process d has P.T.: 3

Press a key for watching timestamp of logical clocks

Logical Clock Timestamp for process a: 6

Logical Clock Timestamp for process b: 13

Logical Clock Timestamp for process c: 18

Logical Clock Timestamp for process d: 23

Experiment – 2

OBJECTIVE: Write a program to simulate the Distributed Mutual Exclusion in 'C'.

```
#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 process 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 critical 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("Process %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

OBJECTIVE: Write a program to implement a distributed chat server using TCP sockets in 'C'.

```
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 nothing_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_ISSET(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] == '\n') {
            /* 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-server.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"

eventset 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_disconnection(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,
                inet_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_event(&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

OBJECTIVE: Implement RPC mechanism for a file transfer across a network in 'C'.

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.getOutputStream();
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 void 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

OBJECTIVE: Write a JAVA code to implement ‘Java RMI’ mechanism for accessing methods of remote systems.

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.java

```
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.java

```
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

OBJECTIVE: Write a code in 'C' to implement sliding window protocol.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>.

void main()
{
    char sendFrame[4],receivedFrame[4],b;
    int acknowledge[4];
    int i,j,noFrame,sent,totalSent=0;
    clrscr();
    printf("Enter the number of frames: ");
    scanf("%d", &noFrame);
    for(i=0;i<noFrame;i++)
    {
        for(j=0;j<4;j++)
        {
            sendFrame[j]='0'+rand()%2;
        }
        printf("\n\nThe frame being sent is: %c",sendFrame);
        retrysend:
        for(j=0;j<4;j++)
        {
            if(rand()%500>80)
            {
                acknowledge[j]=1;
                receivedFrame[j]=sendFrame[j];
            }
            else
            {
                acknowledge[j]=0;
                receivedFrame[j]='x';
            }
        }
        sent=1;
        for(j=0;j<4;j++)
        {
            if(acknowledge[j]==0)
                sent=0;
            delay(40);
            printf("\nAcknowlegment for %d",j);
            printf("th bit was: %d",acknowledge[j]);
        }
        receivedFrame[4]=NULL;
        printf("\nThe frame received was:%s ",receivedFrame);
        if(sent==1)
        {
            printf("\nThe frame sent was sent successfully ");
            getch();
        }
        else
        {
            printf("\nThe frame was not sent!");
        }
    }
}
```

Distributed Systems Laboratory

```
if(sent==0)
{
    printf("\nDo you want to retry sending frame (y/n) ");
    scanf("%c",&b);
    if(b=='y' || b=='Y')
        goto retrysend;
}
if(sent==1)
    totalSent++;
}
printf("\n\nTotal Frames to be sent: %d",noFrame);
printf("\nTotal Frames sent successfully:%d ",totalSent);
getch();
}
```

Experiment – 7

OBJECTIVE: Implement corba mechanism by using c++ program at one end and java program at the other

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>

//
// IDL:Hello:1.0
//
CORBA_ULong Hello_skel::_ob_num_ = 0;

Hello_skel::Hello_skel(const char* name)
{
    assert_nca(name, OBNCANullString);

    try
    {
        _OB_createObjectKeyWithName(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_Object_skel::_OB_dispatch(_ob_op, _ob_seq, _ob_sw,
        _ob_offIn, _ob_offOut);
}

#ifndef __hello_h__
#define __hello_h__

//
// IDL:Hello:1.0
//
class Hello;
typedef Hello* Hello_ptr;
typedef Hello* HelloRef;
typedef OBObjVar< Hello > Hello_var;

//
// IDL:Hello:1.0
//
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 OBUmarshal(Hello_ptr&, const CORBA_Octet*&, bool);
    friend CORBA_Boolean operator>>=(const CORBA_Any&, Hello_ptr&);

    //
    // IDL:Hello/hello:1.0
    //
    virtual void hello();
};

extern const OBTypeCodeConst _tc_Hello;

//
// IDL:Hello:1.0
//
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 void
OBMarshalCount(Hello_ptr p, CORBA_ULong& count)
{
    OBMarshalCount((CORBA_Object_ptr)p, count);
}

void OBUmarshal(Hello_ptr&, const CORBA_Octet*&, bool);

void operator<=(CORBA_Any&, Hello_ptr);
void operator<=(CORBA_Any&, Hello_ptr*);
CORBA_Boolean 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>

//
// IDL:Hello:1.0
//
#ifndef HAVE_NO_EXPLICIT_TEMPLATES
template class OBObjVar< Hello >;
template 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
    OBUmarshal(Hello_ptr& val, const CORBA_Octet*& coct, bool swap)
    {
        Hello_var old = val;
        CORBA_Object_var p;
        OBUmarshal(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(
    "010000000E000000022000000010000000E000000049444C3A48656C6C6F3A312E3000000006000"
    "00048656C6C6F00"
    );

    void
    operator<<=(CORBA_Any& any, Hello_ptr val)
    {
        OBObjAny* o = new OBObjAny;
        o -> b = CORBA_Object::_duplicate(val);
        o -> d = CORBA_Object::_duplicate(val);
        any.replace(_tc_Hello, o, true);
    }

    void
    operator<<=(CORBA_Any& any, Hello_ptr* val)
    {
        OBObjAny* o = new OBObjAny;
        o -> b = *val;
        o -> 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_NO_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 _self, 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 void 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 ); } } }

```