JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 01	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Client Server Implementation using RPC and RMI.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 01

Aim : Client Server Implementation using RPC and RMI.

Theory :

Message Passing

Early and rudimentary distributed systems communicated via *message passing*. This form of communication is very simple. One side packages some data, known as a *message* and sends it to the other side where it is decoded and further action may be taken. The format of the message and the way in which it will be processed by the receiver are application dependent. In some applications the receiver may respond by sending a reply message. In other cases, this might not happen.

It is important to realize that the messages carry only data and are typically represented in a way that is known only by the sender and receiver -- there is nothing standard about it. Unless mitigating action is taken by the designer of the message format or the implementer of the application, the communication might not be interoperable across platforms, because of representation differences (e.g. big-endian vs. little-endian). This approach also makes it hard to reuse components of one distributed system in other distributed systems, because there is really no concept of a common library -- everything is "hand rolled".

Remote Procedure Calls (RPC)

Although message passing can be effective, it would be nice if there were a more uniform, reusable, and user-friendly way of doing things. Remote Procedure Calls (RPCs) provide such an abstraction. Instead of viewing the communication between two systems in terms of independent exchanges of data, we back up a step and examine the overall behaviours of the systems.

Often times the services of the system can often be decomposed into procedures, much like those used in traditional programming. These procedures accept certain types of information, perform some useful operation, and then return a result. The RPC abstraction allows us to extend this paradigm to distributed systems. One system can provide remote procedure calls for use by other systems. From the applications point of view, it can use these remote procedure calls much like local procedure calls. Behind the scenes, the RPC is actually connecting to a remote host, sending it the parameters, performing an operation on that remote host, and then returning the result.

This is very similar to a very specific use of message passing. In fact the function invocation is a message from the client to the server. This message names the function and also provides the parameters. After receiving this message, the server performs the operation and sends a message back to the client with a result. The client then treats this result as if it were the return value from a local procedure call.

The important characteristics of RPCs are these:

- They provide a very familiar interface for the application developer
- They one way of implementing the commonplace request-reply primitive
- The format of the messages is standard, not application dependent
- They make it easier to reuse code, since the RPCs have a standard interface and are separate from any one application-proper.

Limits of RPCs

From the perspective of the application programmer, RPCs operate much like local procedure calls, but there are some important differences:

- A "call be address (reference)" is not possible, because the two processes have different address spaces. Parameters that otherwise might be passed by address are often passed by "in-out" instead. In- out parameters are copied by value on procedure invocation, and again upon return. The end result is that the calling procedure has the new value of the data item. The difference in semantics is that parameters passed by in-out retain their old value until the function returns. Parameters that are passed by reference can change value, even from the perspective of the calling function, throughout.
- Addresses and other large objects are typically passed by address, these need to be copied.
- Byte ordering issues (big-endian/little-endian) might need correction
- Representation issues, such as ACII vs. EPCIDIC might need mitigation.
- "Calls by (C++, et al) reference" require more work as does the return of an object reference. This is because both sides don't share either the same address space or the same mapping tables.

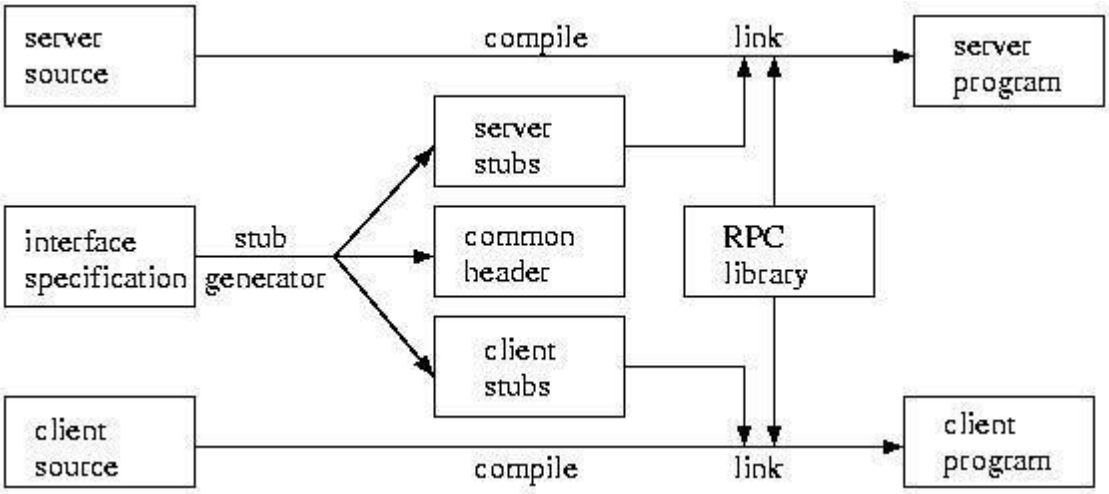
Marshalling and Stubs

The process of preparing and packaging the information for transmission is known as *marshalling* (think of the marshal leading people at a wedding). This often involves translating non-portable representations into a portable or canonical form. In the case of Sun's implementation of RPC, a set of conventions known as external Data Representation (XDR) is used.

In order to hide this process from both the application programmer and the author of the RPC library, it is often implemented using automatically generated *stubs*.

The process basically works like this. The programmer develops the *interface* for the RPC. The *stub generator* takes this interface definition and creates *server stubs* and *client stubs*, as well as a common header file. The server stubs and client stubs take care of the marshalling and unmarshalling of the parameters, as well as the communication and procedure invocations. This is possibly, because these actions are well defined given the procedure's identity and parameterization. Once this is done, the programmer can build the RPCs and the application. Each is linked against the RPC library which provides the necessary code to implement the RPC machinery.

This process is shown in the figure below:

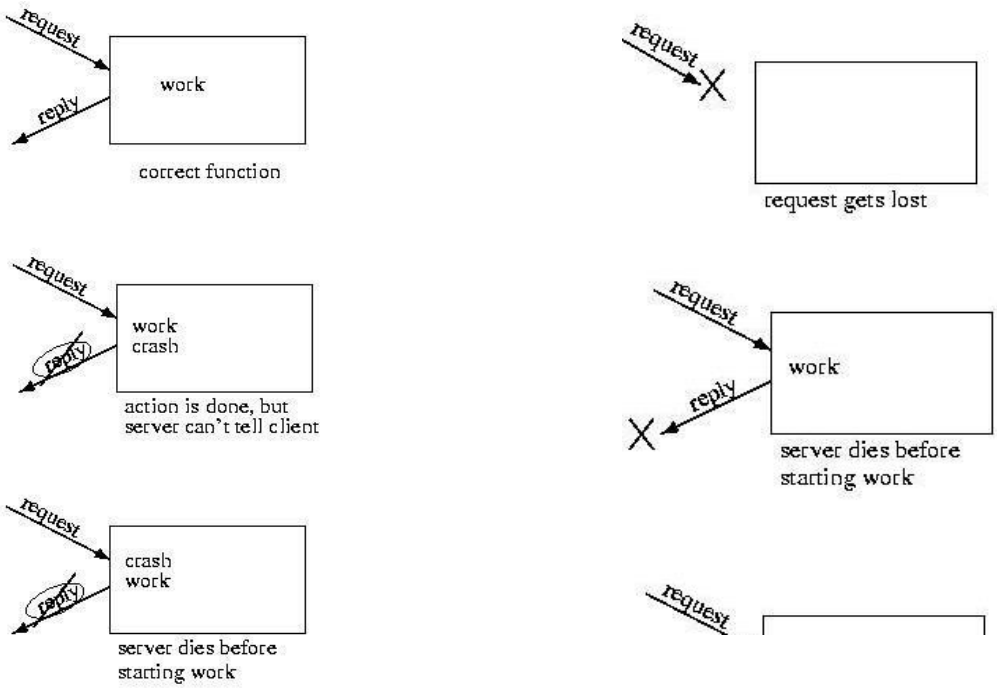


RPC and Failure

The failure modes of RPCs are different than those experienced by local function calls -- there are common failure modes! When was the last time that you can remember a local function call failing? I'm not asking when it was that you most recently observed a bug in a function. I'm asking when it was that you most recently observed the actual transfer of control fail. My point is that in conventional systems this doesn't happen -- and if it should ever happen, it is acceptable to do nothing, but roll over.

But this isn't the case in a distributed system. The communication to the RPC server can fail. The reply from the RPC server can fail. The server can crash. The client can crash. And worst of all, even if we know that something bad happened, we may not know when. What to say? Bad things happen -- but good software is prepared.

Please consider the situations shown below:



These failure modes lead to different semantics for RPCs in light of failure:

- exactly once -- the RPC will be executed exactly once -- never more, never less. Although this is most like local function calls, it is very expensive to implement
- at most once -- if all goes well -- "Hurray! It worked!" Otherwise, no big deal. The important thing is that the operation is never repeated.
- at least once -- if all goes well -- "Hurray! It worked!" Otherwise, keep repeating the operation -- even if there is a risk that it might have happened already (e.g. lost ACK).
- idempotent -- The operation can be repeated without any change. Often times at least once semantics leads to the design of idempotent operations.

Finding an RPC

RPCs live on a specific host, at a specific port. The *port mapper* on the host maps from the RPC name to the port number. Typically when a server is initialized, it registers its RPCs, and their version numbers with the port mapper. A client will first connect to the port mapper to get this *handle* to the RPC. The call to the RPC can then be made by connecting to this port.

"Hello World!" RPC

Below are the source files for a simple "Hello World!" RPC program. The server has a function which returns the string "Hello World!". The client invokes this function remotely, and then prints out the string that it received.

The helloworld.x file was fed to rpcgen, which in turn produced the server and client stubs, helloworld_svc.c and helloworld_clnt.c, and the common header file, helloworld.h.

Normally, the client code, the remote procedure's implementation, and Makefile are created by the programmer from scratch. But, I actually cheated here. "rpcgen - a" will create, in addition to the stubs, skeletons for three more files: the Makefile, the client program, and the remote procedure's implementation. I took the files it produced, filled the remote procedure's implementation into the skeleton, and slightly modified the client.

- helloworld.x: The definition file fed to rpcge

```
program HELLOWORLDPROG {
        version HELLOWORLDVERS {
                string HELLOWORLD(void) = 1;
        } = 1;
} = 0x30000498;
```

- Makefile: The Makefile generated by "rpcgen -a"

```
# This is a template Makefile

generated by rpcgen # Parameters


CLIENT =
helloworld_cl
ient SERVER =
helloworld_se
rver

SOURCES_CLNT.
c =
SOURCES_CLNT.
h =
SOURCES_SVC.c
=
SOURCES_SVC.h
=

SOURCES.x = helloworld.x


TARGETS_SVC.c = helloworld_svc.c
helloworld_server.c TARGETS_CLNT.c =
helloworld_clnt.c helloworld_client.c TARGETS
= helloworld.h           helloworld_clnt.c
helloworld_svc.c helloworld_client.c
helloworld_server.c

OBJECTS_CLNT = $(SOURCES_CLNT.c:%.c=%.o)
$(TARGETS_CLNT.c:%.c=%.o) OBJECTS_SVC =
$(SOURCES_SVC.c:%.c=%.o) $(TARGETS_SVC.c:%.c=%.o)

# Compiler flags


CFLAGS += -g

LDLIBS +=
-lnsl
RPCGENFLAG
S =

# Targets


all : $(CLIENT) $(SERVER)


$(TARGETS) : $(SOURCES.x)

  rpcgen $(RPCGENFLAGS) $(SOURCES.x)


$(OBJECTS_CLNT) : $(SOURCES_CLNT.c) $(SOURCES_CLNT.h)

$(TARGETS_CLNT.c)
```

```
$(OBJECTS_SVC) : $(SOURCES_SVC.c) $(SOURCES_SVC.h) $(TARGETS_SVC.c)


$(CLIENT) : $(OBJECTS_CLNT)

  $(LINK.c) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS)


$(SERVER) : $(OBJECTS_SVC)

  $(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS)


 clean:

   $(RM) core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT)
$(SERVER)
```

- helloworld_client.c: The client program's source, modified slightly from the one generated by "rpcgen -a"

```
#include "helloworld.h"

void

helloworldprog_1(char *host)

{

  CLI
  ENT
  *cl
  nt;
  cha
  r *
  *re
  sul
  t_1
  ;

  char *helloworld_1_arg;


#ifndef   DEBUG

  clnt = clnt_create (host, HELLOWORLDPROG,
  HELLOWORLDVERS, "udp"); if (clnt == NULL) {

          clnt_pcreate
          error
          (host); exit
          (1);

  }
```

```
#endif    /* DEBUG */

        /* GMK: The remote procedure call,
  itself *. result_1 =
  helloworld_1((void*)&helloworld_1_arg,
  clnt); if (result_1 == (char **) NULL) {

          clnt_perror (clnt, "call failed");

  }

#ifndef
  DEBUG
  clnt_dest
  roy
  (clnt);

#endif     /* DEBUG */


        /* GMK: Printing out the results */

        printf ("Got \"%s\" from the server.\n", *result_1);

}



int

main (int argc, char *argv[])

{

  char *host;


  if (argc < 2) {

          printf ("usage: %s
          server_host\n", argv[0]); exit
          (1);

  }

  host =
  argv[1];
  helloworldp
  rog_1
  (host);

        exit (0);

}
```

- helloworld_server.c: The remote method's

implementation, filled into the skeleton provided by "rpcgen -a"

```
#include

"helloworld.h

" char **

helloworld_1_svc(void *argp, struct svc_req *rqstp)

{

        /*

         * Notice two things:
           1)  Both of these are static, so that they exist upon
         *    return so RPC can still access the pointer to get
         *    to the string, and access the string to marshal it
         *
         * 2) I didn't just return &hello, but instead used a
         *    separate pointer. &&hello doesn't make sense and
         *    this must return a pointer to the data to return.
         */


        static char hello[256] =
        "Hello world"; static char
        *hello_p;

        printf("Entering

        helloworld_1_svc()\n");

        hello_p = hello;


  printf("Returning from helloworld_1_svc()\n");


        return(&hello_p);

}
```

- helloworld.h: The common header file generated by *rpcgen*

```
#ifndef
_HELLOWORLD_H_RPCGEN
#define
_HELLOWORLD_H_RPCGEN


#include <rpc/rpc.h>
```

```
#ifd
ef
cplu
splu
s
exte
rn
"C"
{
#end
if


#define
HELLOWORLDPROG
0x30000498 #define
HELLOWORLDVERS 1

#if defined(__STDC__) || defined(
cplusplus) #define HELLOWORLD 1

extern  char ** helloworld_1(void *, CLIENT *);

extern  char ** helloworld_1_svc(void *, struct
svc_req *); extern int helloworldprog_1_freeresult
(SVCXPRT *, xdrproc_t, caddr_t);


#else /*
K&R C */
#define
HELLOWORLD
1

extern  char **
helloworld_1(); extern
char **
helloworld_1_svc();

extern int
helloworldprog_1_freeresult ();
#endif /* K&R C */


#ifdef __cplusplus

}

#endif


#endif /* !_HELLOWORLD_H_RPCGEN */
```

- helloworld_clnt.c: The client stub produced by rpcgen from the helloworld.x file

```
#include <memory.h> /* for
memset */ #include
"helloworld.h"


/* Default timeout can be changed using
clnt_control() */ static struct timeval TIMEOUT
= { 25, 0 };

char **

helloworld_1(void *argp, CLIENT *clnt)

{

        static char *clnt_res;


        memset((char *)&clnt_res, 0,
        sizeof(clnt_res)); if (clnt_call
        (clnt, HELLOWORLD,

                (xdrproc_t) xdr_void, (caddr_t)
                argp, (xdrproc_t) xdr_wrapstring,
                (caddr_t) &clnt_res, TIMEOUT) !=
                RPC_SUCCESS) {

                return (NULL);

        }

        return (&clnt_res);

}
```

- helloworld_svc.c: The server stub produced by rpcgen from the helloworld.x file

```
#include
"helloworld.
h" #include
<stdio.h>
#include
<stdlib.h>
#include
<rpc/pmap_cl
nt.h>
#include
<string.h>
#include
<memory.h>
#include
<sys/socket.
```

```
h> #include
<netinet/in.
h>

#ifndef SIG_PF

#define SIG_PF
void(*)(int)
#endif

static void

helloworldprog_1(struct svc_req *rqstp, register SVCXPRT *transp)

{

  union {

          int fill;

  }

  xdrproc_t _xdr_argument,
  _xdr_result; char
  *(*local)(char *, struct
  svc_req *);

  switch (rqstp-
  >rq_proc) {
  case NULLPROC:

       (void) svc_sendreply (transp, (xdrproc_t) xdr_void,
                                 (char

*)NULL);

          return;


  case HELLOWORLD:

          _xdr_argument = (xdrproc_t) xdr_void;

          _xdr_result = (xdrproc_t)
          xdr_wrapstring; local = (char
          *(*)(char *, struct svc_req *))

helloworld_1_svc;

          break;


  default:

          svcerr_nopro
          c (transp);
          return;

  }

  memset ((char *)&argument, 0, sizeof (argument));
```

```
  if (!svc_getargs (transp, _xdr_argument, (caddr_t)
          &argument)) { svcerr_decode (transp);

          return;

  }

  result = (*local)((char *)&argument, rqstp)

  if (result != NULL && !svc_sendreply(transp, _xdr_result,
          result)){ svcerr_systemerr (transp);

  }

  if (!svc_freeargs (transp, _xdr_argument, (caddr_t)
          &argument)) { fprintf (stderr, "unable to free
          arguments");

          exit (1);

  }

  return;

}


int

main (int argc, char **argv)

{

  register SVCXPRT *transp;


  pmap_unset (HELLOWORLDPROG, HELLOWORLDVERS);

  transp =
  svcudp_create(RPC_ANYSOC
  K); if (transp == NULL)
  {

          fprintf (stderr, "cannot create udp
          service."); exit(1);

  }

  if (!svc_register(transp, HELLOWORLDPROG,
HELLOWORLDVERS, helloworldprog_1, IPPROTO_UDP)) {

          fprintf (stderr, "unable to register
(HELLOWORLDPROG, HELLOWORLDVERS, udp).");

          exit(1);

  }
```

```
  transp =
  svctcp_create(RPC_ANYSOCK, 0,
  0); if (transp == NULL) {

          fprintf (stderr, "cannot create tcp
  service."); exit(1);

  }

  if (!svc_register(transp, HELLOWORLDPROG,
HELLOWORLDVERS, helloworldprog_1, IPPROTO_TCP)) {

          fprintf (stderr, "unable to register
(HELLOWORLDPROG, HELLOWORLDVERS, tcp).");

          exit(1);

}


svc_run ();

fprintf (stderr, "svc_run
returned"); exit (1);

/* NOTREACHED */

}
```

RMI

Remote method invocation(RMI) allow a java object to invoke method on an object running on another machine. RMI provide remote communication between java program. RMI is used for building distributed application.

Concept of RMI application

A RMI application can be divided into two part,**Client** program and **Server** program. A **Server** program creates some remote object, make their references available for the client to invoke method on it. A **Client** program make request for remote objects on server and invoke method on them. **Stub** and **Skeleton** are two important object used for communication with remote object.

Stub

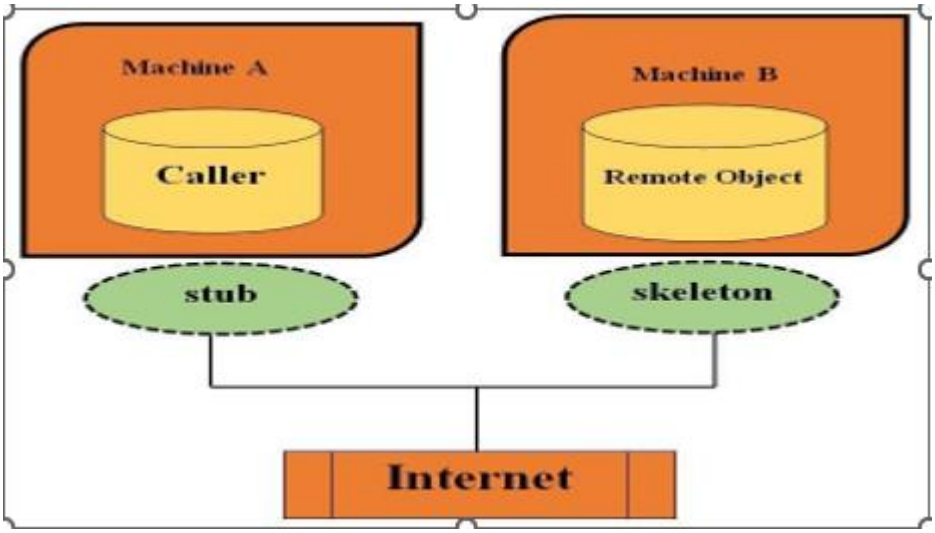
In RMI, a stub is an object that is used as a Gateway for the client-side. All the outgoing request are sent through it. When a client invokes the method on the stub object following things are performed internally:

1. A connection is established using Remote Virtual Machine.
2. It then transmits the parameters to the Remote Virtual Machine. This is also known as Marshals
3. After the 2nd step, it then waits for the output.
4. Now it reads the value or exception which is come as an output.
5. At last, it returns the value to the client.

Skeleton

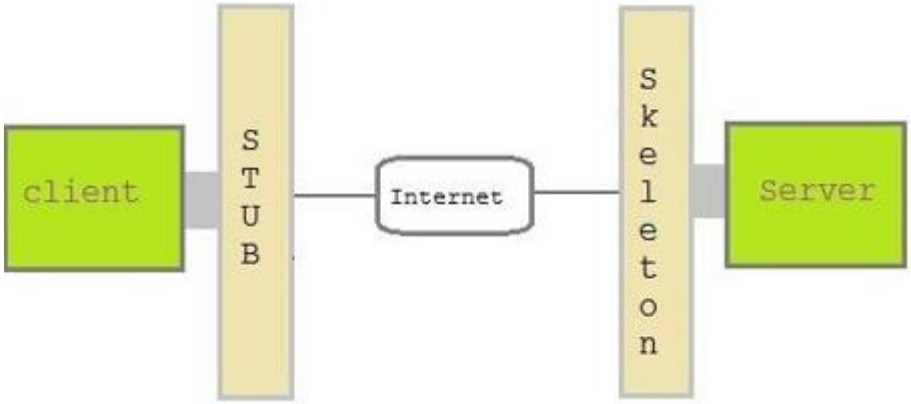
In RMI, a skeleton is an object that is used as a Gateway for the server-side.All the incoming request are sent through it. When a Server invokes the method on the skeleton object following things are performed internally:

1. All the Parameters are read for the remote method.
2. The method is invoked on the remote object.
3. It then writes and transmits the parameters for the result. This is also known as Marshals.



Stub and Skeleton

Stub act as a gateway for Client program. It resides on Client side and communicate with **Skeleton** object. It establish the connection between remote object and transmit request to it.



Skeleton object resides on server program. It is responsible for passing request from **Stub** to remote object.

Creating a Simple RMI application involves following steps

- Define a remote interface.
- Implementing remote interface.
- create and start remote application
- create and start client application

Define a remote interface

A remote interface specifies the methods that can be invoked remotely by a client. Clients

program communicate to remote interfaces, not to classes implementing it.
To be a remote interface, a interface must extend the **Remote** interface of **java.rmi** package.

```
import java.rmi.*;

public interface AddServerInterface extends Remote

{

public int sum(int a,int b);

}
```

Implementation of remote interface

For implementation of remote interface, a class must either extend **UnicastRemoteObject** or use exportObject() method of **UnicastRemoteObject** class.

```
import java.rmi.*;

import java.rmi.server.*;

public class Adder extends UnicastRemoteObject implements
AddServerInterface

{

	Adder()throws RemoteException{

	super();

}

public int sum(int a,int b)

{

	return a+b;

}

}
```

Create AddServer and host rmi service

You need to create a server application and host rmi service **Adder** in it. This is done using rebind()method of **java.rmi.Naming** class. rebind()method take two arguments, first represent the name of the object reference and second argument is reference to instance of **Adder**

```
import java.rmi.*;
import java.rmi.registry.*;
public class AddServer {
		try {
			AddServerInterface addService=new Adder();
			Naming.rebind("AddService",addService);

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

Create client application

Client application contains a java program that invokes the lookup()method of the **Naming** class. This method accepts one argument, the **rmi** URL and returns a reference to an object of type **AddServerInterface**. All remote method invocation is done on this object.

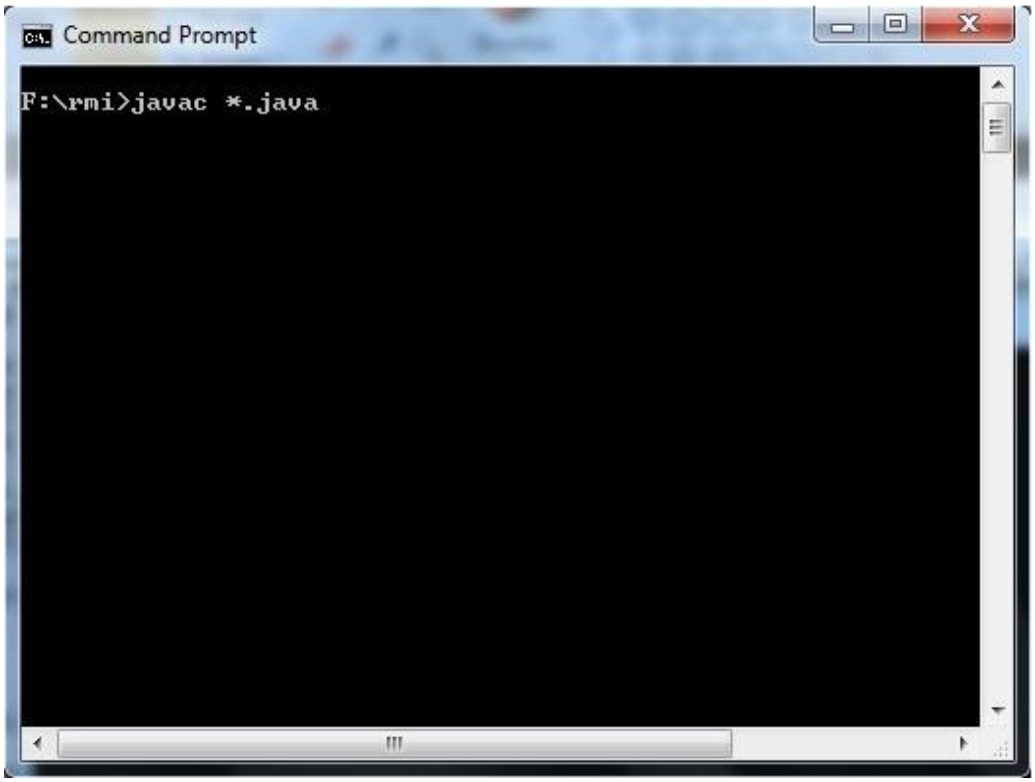
```
import java.rmi.*;
public class Client
{
	public static void main(String args[])
		{ try{
AddServerInterface st =
(AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddSer
e");
			System.out.println(st.sum(25,8));
		}
		catch(Exception e) {
			System.out.println(e);
		}
```

Steps to run this RMI application

Save all the above java file into a directory and name it as "rmi"

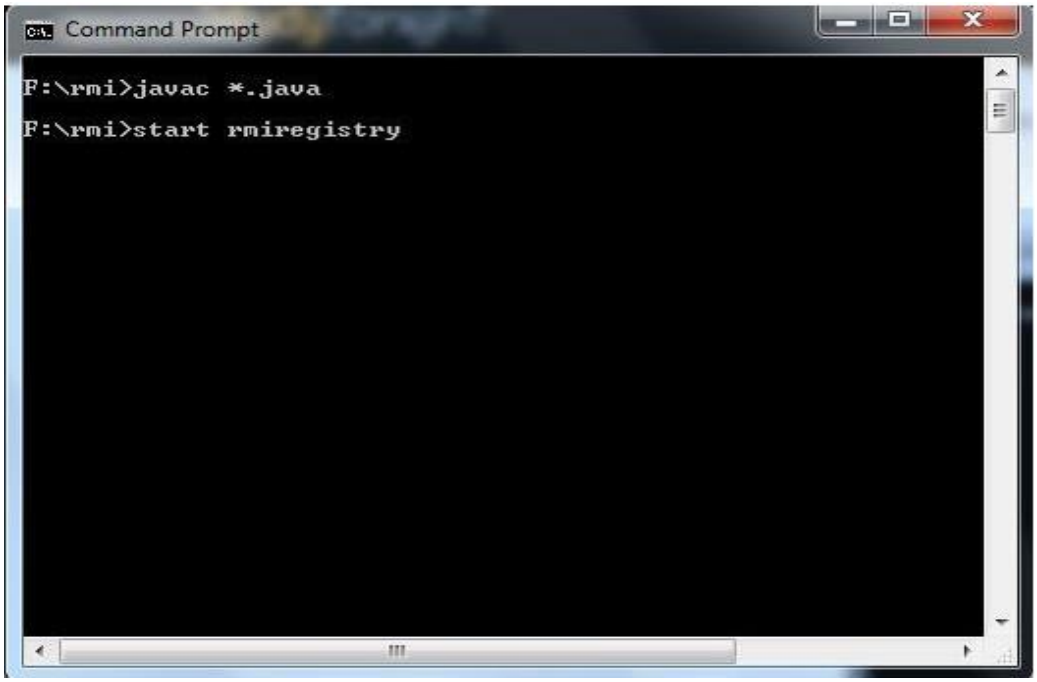
- compile all the java files

```
javac *.java
```



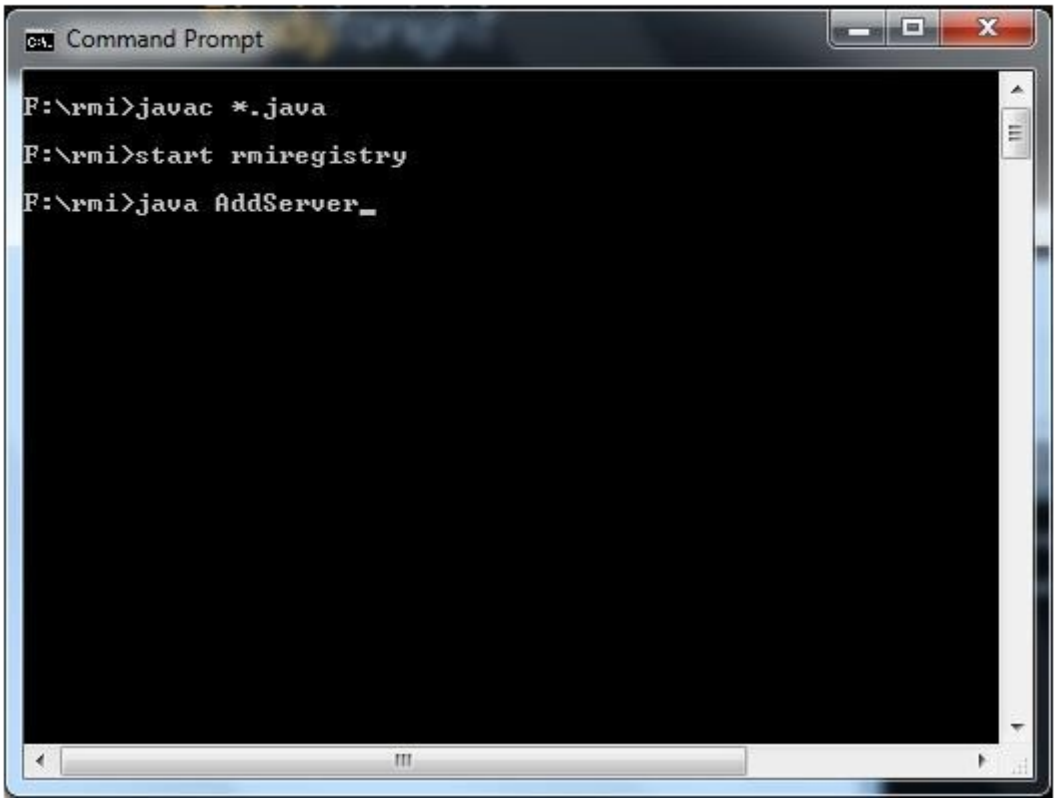
Start RMI registry

```
start rmiregistry
```



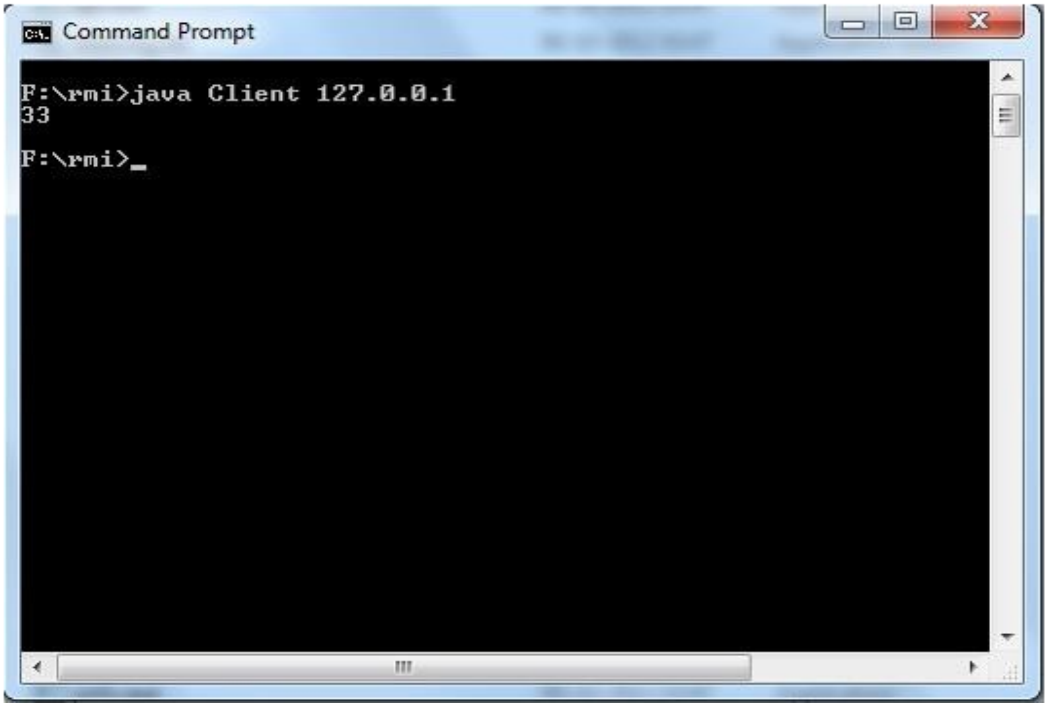
Run Server file

```
java AddServer
```



Run Client file in another command prompt abd pass local host port number at run time

```
java Client 127.0.0.1
```



Example:

Program: Power.java

```
import java.rmi.*;

public interface Power extends Remote

{

	public int power1()throwsRemoteException;

}
```

Program: PowerRemote.java

```
import java.rmi.server.*;

import java.util.Scanner;

public class PowerRemote extends UnicastRemoteObject
implements Power

{

PowerRemote()throws RemoteException

{

	super();

}

public int power1(int z)

{

int z;

Scanner sc = new Scanner(System.in);

System.out.println("Enter the base number ::");

int x = sc.nextInt();
```

```
System.out.println("Enter the exponent number ::");
int y = sc.nextInt();
                z=y^x;
System.out.println(z);
}
}
```

MyServer.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer
{
public static void main(String args[])
{
try
{
Power stub=new PowerRemote();
Naming.rebind("rmi://localhost:1995/shristee",stub);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

MyClient.java

```
import java.rmi.*;

public class MyClient

{

public static void main(String args[])

{

try

{

Power
stub=(Power)Naming.lookup("rmi://localhost:1995/shristee");

System.out.println(stub.power1());

}
```

```
E:\java\rmi>javac *.java

E:\java\rmi>rmic PowerRemote
```

```
E:\java\rmi>rmiregistry  5000
```

```
E:\>cd java

E:\java>cd rmi

E:\java\rmi>java MyServer
```

```
E:\>cd java

E:\java>cd rmi

E:\java\rmi>java MyClient
256.0
```

Conclusion : We have successfully studied Client Server Implementation using RPC and RMI.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 02	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Case study on Inter-process communication.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 03	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Case study on Group Communication.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 04	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To study Election Algorithm.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 04

Aim : To study Election Algorithm.

Theory :

Election Algorithms

Election Algorithms

- The **coordinator election problem** is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.
- An **election algorithm** is an algorithm for solving the coordinator election problem. By the nature of the coordinator election problem, any election algorithm must be a distributed algorithm.

-a group of processes on different machines need to choose a coordinator

-peer to peer communication: every process can send messages to every other process.

-Assume that processes have unique IDs, such that one is highest

-Assume that the priority of process P_i is i

(a) *Bully Algorithm*

Background: any process P_i sends a message to the current coordinator; if no response in T time units, P_i tries to elect itself as leader. Details follow:

Algorithm for process P_i that detected the lack of coordinator

1. Process P_i sends an "Election" message to every process with higher priority.
2. If no other process responds, process P_i starts the coordinator code running and sends a message to all processes with lower priorities saying "Elected P_i"
3. Else, P_i waits for T' time units to hear from the new coordinator, and if there is no response start from step (1) again.

Algorithm for other processes (also called P_i)

If P_i is not the coordinator then P_i may receive either of these messages

from P_j

if Pi sends "Elected Pj"; [this message is only received if $i <$

j] Pi updates its records to say that Pj is the

coordinator.

Else if Pj sends "election" message (i > j)

Pi sends a response to Pj saying it is alive Pi starts an election.

(b) *Election In A Ring => Ring Algorithm.*

-assume that processes form a ring: each process only sends messages to the next process in the ring

- Active list: its info on all other active processes
- assumption: message continues around the ring even if a process along the way has crashed.

Background: any process Pi sends a message to the current coordinator; if no response in T time units, Pi initiates an election

1. initialize active list to empty.
2. Send an "Elect(i)" message to the right. + add i to active list.

If a process receives an "Elect(j)" message

(a) this is the first message sent or seen
initialize its active list to [i,j]; send "Elect(i)" + send "Elect(j)"

(b) if i != j, add i to active list + forward "Elect(j)" message to active list

(c) otherwise (i = j), so process i has complete set of active processes in its active list.
=> choose highest process ID + send "Elected (x)" message to neighbor

If a process receives "Elected(x)"
message, set coordinator to x

Example:

Suppose that we have four processes arranged in a ring: P1 □ P2 □ P3 □ P4 □ P1

…

P4 is coordinator
Suppose P1 + P4
crash

Suppose P2 detects that coordinator P4 is not
responding P2 sets active list to [ ]

P2 sends “Elect(2)” message to P3; P2 sets active list to [2]
P3 receives “Elect(2)”

This message is the first message seen, so P3 sets its active list to
[2,3] P3 sends “Elect(3)” towards P4 and then sends “Elect(2)”
towards P4 The messages pass P4 + P1 and then reach P2

P2 adds 3 to active list
[2,3] P2 forwards
“Elect(3)” to P3

P2 receives the “Elect(2) message

P2 chooses P3 as the highest process in its list [2, 3] and sends an “Elected(P3)” message

P3 receives the “Elect(3)” message

P3 chooses P3 as the highest process in its list [2, 3] + sends an “Elected(P3)” message

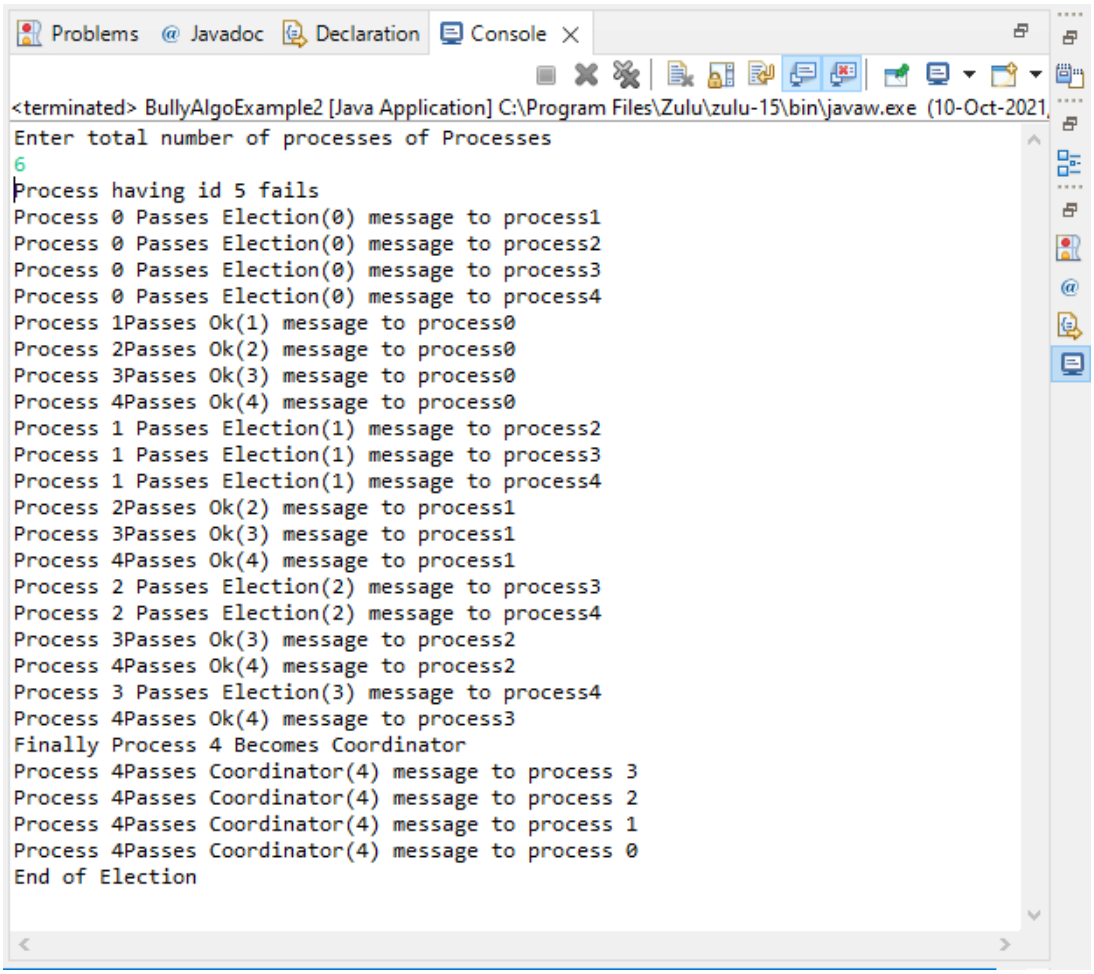
Program: Bully Algorithm

```
// import required classes and packages
packagejavaTpoint.javacodes;

import java.io.*;
importjava.util.Scanner;

// create class BullyAlgoExample to understand how bully algorithms works
classBullyAlgoExample{

// declare variables and arrays for process and their status
staticintnumberOfProcess;
staticintpriorities[] =
newint[100]; staticintstatus[] =
newint[100]; staticintcord;

   // main() method start
publicstaticvoid main(String args[])throwsIOException   // handle IOException
   {
```

```
    // get input from the user for the number of processes
System.out.println("Enter total number of processes:");

      // create scanner class object to get input from
     user Scanner sc = newScanner(System.in);
     numberOfProcess = sc.nextInt();

   inti;

  // use for loop to set priority and status of each process
for(i = 0; i<numberOfProcess; i++)
      {

        System.out.println("Status for process
        "+(i+1)+":"); status[i] = sc.nextInt();

        System.out.println("Priority of process
        "+(i+1)+":"); priorities[i] = sc.nextInt();
      }


System.out.println("Enter proces which will initiate
election"); intele = sc.nextInt();

sc.close();

// call electProcess() method
electProcess(ele);
System.out.println("After electing process the final coordinator is "+cord);
   }


    // create electProcess() method
staticvoidelectProcess(intele)
   {

        ele =
        ele - 1;
        cord =
        ele + 1;

for(inti = 0; i<numberOfProcess; i++)
```

```
        {
          if(priorities[ele]<priorities[i])
        {
                System.out.println("Election message is sent from "+(ele+1)+" to
                "+(i+1));
                if(status[i]==1)
                 electProcess(i+1);
                 }
             }
          }
        }
```

Output:



Program: Ring Algorithm

```
import
java.util.Scanner; class
Process{
public int id;
public boolean active;
public
Process(int id){
this.id=id;
active=true;
}
}
public
class Ring{
int
noOfProc
esses;
Process[]
processes;
Scanner
sc;
public
Ring(){ sc=new
Scanner(System.in
);
}
public void initialiseRing(){
System.out.println("Enter no of
processes");
noOfProcesses=sc.nextInt();
processes = new
Process[noOfProcesses]; for(int
i=0;i<processes.length;i++){
processes[i]= new Process(i);
}
}
```

```
        public int
getMax(){ int
maxId=-99;
int maxIdIndex=0;
for(int i=0;i<processes.length;i++){
if(processes[i].active &&
processes[i].id>maxId){
maxId=processes[i].id;
maxIdIn
dex=i;
    }

}

return maxIdIndex;
}

public void performElection(){
        System.out.println("Process no "+processes[getMax()].id+" fails");
processes[getMax()].active=false;
System.out.println("Election
Initiated by"); int
initiatorProcesss=sc.nextInt();
        int prev =
initiatorProcesss; int next
= prev+1;
        while(true
){
if(processes[next
].active){
System.out.println("Process "+processes[prev].id+" pass
Election("+processes[prev].id+") to"+processes[next].id);
prev=next;
}

        next =
(next+1)%noOfProcesses; if(next
== initiatorProcesss){
break;
}
```

```
}
        System.out.println("Process "+ processes[getMax()].id +" becomes
        coordinator");
int coordinator =
        processes[getMax()].id; prev
        = coordinator;
next
        =(prev+1)%no
        OfProcesses;
        while(true){
        if(processes[nex
        t].active)
{
System.out.println("Process "+ processes[prev].id +" pass
Coordinator("+coordinator+ ")
message to process
"+processes[next].id ); prev = next;
}
next = (next+1) %
noOfProcesses; if(next
== coordinator)
{
System.out.println("End Of Election ");
break;
}
}
        }
        public static void
main(String arg[]){ Ring r= new
Ring();
r.initialiseRing();
r.performElectio
n();
}
```

```
}
```

Output:

```
C:\Users\Garry\Desktop\CLIX\Bully>java
Bully Enter No of Processes
5
Process no 4 fails
Process 0Passes Election(0) message to
process 1 Process 0Passes Election(0)
message to process 2 Process 0Passes
Election(0) message to process 3 Process
1Passes Ok(1) message to process 0
Process 2Passes Ok(2) message to
process 0 Process 3Passes Ok(3)
message to process 0 Process 1Passes
Election(1) message to process 2 Process
1Passes Election(1) message to process 3
Process 2Passes Ok(2) message to
process 1 Process 3Passes Ok(3)
message to process 1 Process 2Passes
Election(2) message to process 3 Process
3Passes Ok(3) message to process 2
Finally Process 3 Becomes Coordinator
Process 3Passes Coordinator(3) message to
process 2 Process 3Passes Coordinator(3)
message to process 1
```

Conclusion : We have successfully studied Election Algorithm.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 05	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To Study Network operating system and Distributed Operating system with example	

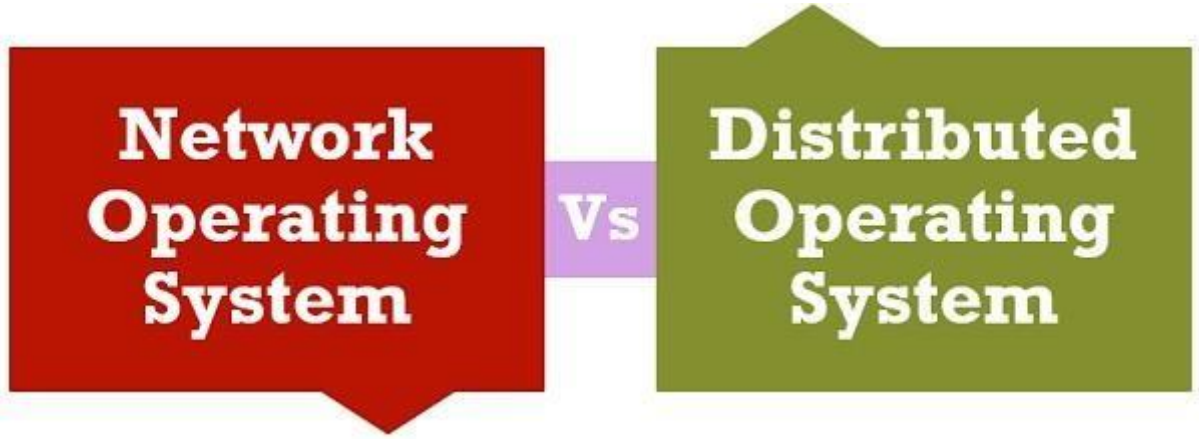
Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 05

Aim: To Study Network operating system and Distributed Operating system with example



Network Operating System falls under the category of Distributed architectures where a large number of computer systems are connected with each other with the help of a network. Although the implementation of the network operating system is simpler than the distributed operating system. The network operating system and distributed operating system are distinguished by the characteristics they are having, such as the in network operating system each system runs its own operating system while the distributed operating system runs a global system-wide operating system.

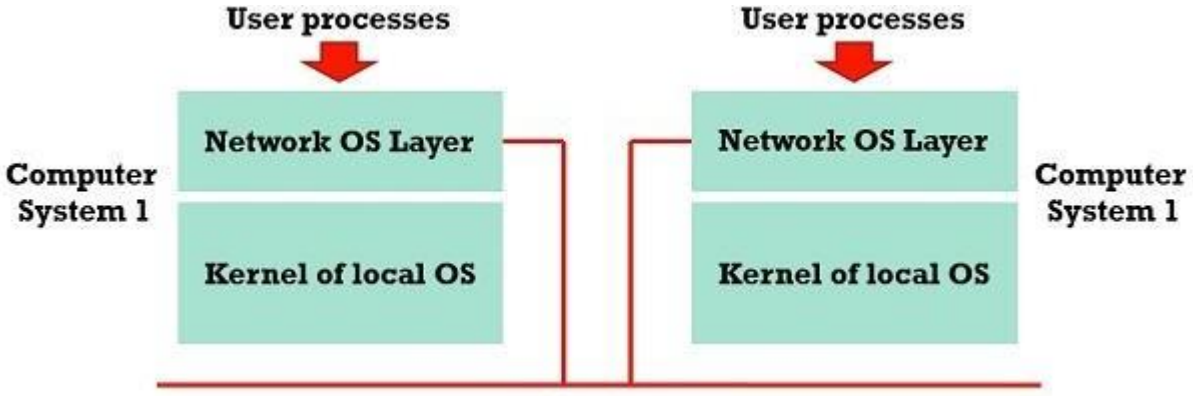
Comparison Chart

BASIS FOR COMPARISON	NETWORK OPERATING SYSTEM	DISTRIBUTED OPERATING SYSTEM
Objective	Provision of local services to the remote client.	Management of hardware resource.
Use	Loosely coupled system employed in heterogeneous computers.	Tightly coupled system used in multiprocessor and homogeneous computers.

Architecture	2-tier client/server architecture.	N-tier client/server architecture.
Level of transparency	Low	High
Basis for communication	Files	Shared memory and messages
Resource Management	Handled at each node.	Global central or distributed management.
Ease of implementation	High	Low
Scalability	More	Less or moderate.
Openness	Open	Closed
Operating system on all nodes	Can be different	Same
Rate of autonomy	High	Low
Fault tolerance	Less	High

Definition of Network operating system

1. The network operating system is the platform to run a system software on a server and allow the server to manage the users, data, groups, security, applications and other networking functions. It is considered as the primary form of an operating system for the distributed architecture. The idea behind the network operating system is to permit resource sharing among two or more computers operating under their own OSs. The functioning of the network operating system can be explained by the diagram depicted below.



Here the network OS layer present between the kernel of the local OS and user processes. Essentially the processes interact with the network OS layer instead of the kernel of the local operating system. When the process demand for the non-local resources, the network OS layer communicates to the network OS layer of the node that contains the resource and employs the access to the resource by using it. On the other way, if the process request for the local resource, the network OS layer sends the request to the local OS kernel.

2. Unlike the distributed operating system, the network operating system does not work in a collaborated way. The local operating system residing in each particular computer preserves its identity which is visible to users too and behaves as a solitary operating system. In some implementations, there is a remote login for the remote operating systems to access the resources. A network operating system cannot control the utilisation of resources which causes the improper distribution of the resources. There is no provision of fault tolerance in the network operating system.
3. Definition of Distributed operating system
4. The distributed operating system handles a group of independent computers and makes them look like an ordinary centralised operating system. This is achieved by enabling the proper communication between the different computers connected with each other. The main aim of the distributed operating system is the transparency where the use of multiple hardware resources is hidden from the users. The distributed operating system is less autonomous than network operating system as the system has complete control in this environment. It dynamically allocates processes to the random CPU and the files storage is also managed by the operating system which means that the user would not know which hardware has been used for the processing of its computation and for storing its file.

5. As it is mentioned above the distributed operating system allows resource sharing in which an application can use resources located in any computer system. It provides the availability (continuity of the services) rather than faults. A distributed operating system handles the operation of all nodes in the system in an integrated way since each node has its separate kernel to perform control functions on its behalf. It also increases the computational speed by executing the parts of computation in different computer systems.

Key Differences Between Network operating system and Distributed operating system

6. The main goal of the network operating system is to provide local services to the remote client. On the other hand, the objective of the distributed operating system is to provide the hardware resource management.
7. Network operating systems are said to be loosely coupled systems and are used in heterogeneous computers. As against, distributed operating system is considered as tightly coupled systems mainly used in multiprocessors or homogeneous computers.
8. The network operating system has two-tier client/server architecture, while n-tier architecture is employed in the distributed operating system.
9. Transparency in the network operating system is low. Conversely, the distributed operating system has high transparency, and it hides the resource utilisation.
10. In the distributed operating system the communication between the computers (nodes) is achieved by shared memory or sending messages. On the contrary, the network operating system sends files in order to communicate with other nodes.
11. Network operating system manages resources at each node while in the distributed operating system, the resources are globally managed whether it is centred or distributed.
12. The network operating system is easily implemented as compared to the distributed operating system.
13. Scalability of the network operating system is higher than the distributed operating system, and also it is more open to the user.
14. In network operating system the operating system installed in the computers can vary whereas it is not the case in the distributed operating system.
15. The network operating system is more autonomous than the distributed operating system. In contrast, the distributed operating system is more fault tolerant.

Conclusion : The prior difference between network operating system and distributed operating system lies within their implementation wherein network operating system there is no modification or changes are applied to the core system while in the distributed operating system the system components can be subjected to upgradation if required.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 06	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To Study stateful server and stateless server.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment N0. 06

Aim : To Study stateful server and stateless server.

Theory :

Like many industries, the API space has its own jargon. Many terms are used so often that it's a common assumption that the related party understands what is being talked about. But for for newcomers, these subtle definitions may not be as apparent.

Take the difference between **stateless** and **stateful**; an invaluable distinction within the development of APIs and the services that use those systems. Accordingly, in this piece, we're going to briefly discuss what these terms actually mean. We'll take a look at what makes *statefulness* and *statelessness* so different from one another, and what this actually means in terms of the API.

Stateful

To understand statelessness, one must understand statefulness. When we talk about computer systems, a "state" is simply the condition or quality of an entity at an instant in time, and to be stateful is to rely on these moments in time and to change the output given the determined inputs and state.

If that's unclear, don't worry — it's a hard concept to grasp, and doubly so with APIs. We can break this down even further — consider binary, a language of 1's and 0's. What this functionally represents is either "on" or "off" — a binary system cannot be both 1 and 0, and are so mutually exclusive.

Now, consider a theoretical situation in which you are given a piece of paper with these simple instructions — "if the number is 0, say no, if 1, say yes" — and you were placed into a room with a binary display which changed between the number 0 and 1 every five seconds.

This is a stateful system. Your answer will depend entirely on whether or not that clock says "0" or "1" — you cannot answer independently of the state of the grand machine. This is statefulness.

Stateful Web Services

With this in mind, what does a stateful web service looks like? Let's say you log into a resource, and in doing so, you pass your password and username. If the web server stores this data in a backend manner and uses it to identify you as a constantly connected client, the service is stateful. Do keep in mind that this is a very specific example that exists in other forms, so what seems stateful may not necessarily be stateful — more on this later.

As you use the web service, everything you do is referenced back to this stored state. When you request an account summary, the web service asks two things:

Who is making this request?

Using the ID stored for who is making this request, what should their page look like?

In a stateful web service like this, the response formed from a simple GET request is

entirely dependent on the state registered by the server. Without knowledge of that state, your request cannot be returned properly.

Another great example is FTP. When a user logs in to a traditional FTP server, they are engaging in an active connection with the server. Each change to the state of the user, such as active directory, is stored on the server as a client state. Each change made to the server is registered as a change of state, and when the user disconnects, their state is further changed to disconnected.

So far so good, right? Well, not quite. Stateful programming is fine in some very limited applications, but it has a lot of issues. First and foremost, when you have to reference a state, you're opening yourself up to a lot of incomplete sessions and transactions. Let's say you make a call to present a piece of data. In a stateful system where the state is determined by the client, how long is the system supposed to leave this connection open? How do we verify if the client has crashed or disconnected? How do we track the actions of the user while maintaining the ability to document changes and roll back when necessary?

While there are certainly workarounds for all of these questions, more often than not, statefulness is only really useful when the functions themselves depend on the statefulness quality. Most consumers are able to respond to the server in intelligent, dynamic ways, and because of this, maintaining server state independent of the consumer as if the consumer was simply a "dumb" client is wasteful and unnecessary.

Stateless

The answer to these issues is **statelessness**. Stateless is the polar opposite of stateful, in which any given response from the server is **independent of any sort of state**.

Let's go back to that binary room theoretical. You are given the same binary clock, only this time, the paper simply has a name — "Jack" — and the instructions are to respond when someone says the password "fish". You sit watching the clock slowly change, and each time someone says the special password, you say the name "Jack".

This is statelessness — there's no need to even reference the clock, because the information is stored locally in such a way that the requests are **self contained** — it's dependent only on the data you hold. The speaker could easily say the secret word, tell you to change the name, then walk away. He can then come back an hour later, say the secret password, and get the new name
— everything is contained within the request, and handled in two distinct phases, with a "request" and a "response".

This is a stateless system. Your response is independent of the "0" or "1", and each request is self contained.

Stateless Web Services

Statelessness is a fundamental aspect of the modern internet — so much so that every single day, you use a variety of stateless services and applications. When you read the news, you are using **HTTP** to connect in a stateless manner, utilizing messages that can be parsed and worked with in isolation of each other and your state.

If you have **Twitter** on your phone, you are constantly utilizing a stateless service. When the service requests a list of recent direct messages using the Twitter REST API, it issues the following request:

GET

https://api.twitter.com/1.1/direct_messages.json?since_id=240136858829479935&count=1

The response that you will get is entirely independent of any server state storage, and everything is stored on the client's side in the form of a cache.

Let's take a look at another example. In the example below, we are invoking a POST command, creating a record on HypotheticalService:

POST http://HypotheticalService/Entity Host: HypotheticalService

Content-Type: text/xml; charset=utf-8 Content-Length: 123

<!--?xml version="1.0" encoding="utf-8"?-->

<entity>

<id>1</id>

<title>Example</title>

<content>This is an example</content>

</entity>

In this example, we are creating an entry, but this entry does not depend on any matter of state. Do keep in mind that this is a simple use case, as it does not pass any authorization/authentication data, and the POST issuance itself contains only very basic data.

Even with all of this in mind, you can plainly see that doing a POST issuance in a stateless manner means that you do not have to wait for server synchronization to ensure the process has been properly completed, as you would with FTP or other stateful services. You receive a confirmation, but this confirmation is simply an affirmative response, rather than a mutual shared state.

As a quick note, it must be said that REST is specifically designed to be functionally stateless. The entire concept of Representational State Transfer (from which REST gets its name) hinges on the idea of passing all data to handle the request in such a way as to pair the data within the request itself. Thus, REST should be considered stateless (and, in fact, that is one of the main considerations as to whether something is RESTful or not as per the original dissertation by Roy Fielding which detailed the concept).

Smoke and Mirrors

We need to be somewhat careful when we talk about web services as examples of stateful or stateless, though, because what seems to fall in one category may not

actually be so. This is largely because stateless services have managed to mirror a lot of the behavior of stateful services without technically crossing the line.

Statelessness is, just like our example above, all about self-contained state and reference rather than depending on an external frame of reference. The difference between it and statefulness is really where the state is stored. When we browse the internet or access our mail, we are generating a state — and that state has to go somewhere.

When the state is stored by the server, it generates a session. This is **stateful computing**. When the state is stored by the client, it generates some kind of data that is to be used for various systems — while technically "stateful" in that it references a state, the state is stored by the client so we refer to it as stateless.

This seems confusing, but it's actually the best way to work around the limitations of statelessness. In a purely stateless system, we're essentially interacting with a **limited system** — when we would order an online good, that'd be it for us, it wouldn't store our address, our payment methods, even a record of our order, it would simply process our payment and, as far as the server was concerned, we'd cease to be.

That's obviously not the best case scenario, and so, we made some concessions. In the client cookie, we store some basic authentication data. On the server side, we create some temporary client data or store on a database, and reference it to an external piece of data. When we return to make another payment, it's our cookie that establishes the state, not the non-existent session.

What's So Bad About Sessions?

In terms of web services, the commonly accepted paradigm is to **avoid sessions** at all costs. While this certainly doesn't apply to every single use case, using sessions as a method for communicating state is generally something you want to avoid.

To start with, sessions add a large amount of complexity with very little added value. Sessions make it harder to replicate and fix bugs. Sessions can't really be "bookmarked", as everything is stored on the server side. All of these are significant issues, but they pale in comparison to the simple fact that **sessions are not scalable**.

Gregor Riegler at BeABetterDeveloper gave a wonderful explanation on why this is in his piece "Sessions, a Pitfall":

Lets say you are a professional chess player, and you'd like to play multiple people at the same time. If you'd try to remember every game and your strategy on it, you'll hit your capacity rather quick. Now imagine you were not remembering anything of those games, and you were just rereading the chessboard on every move. You could literally play 1.000.000 people at the same time, and it wouldn't make any difference to you.

Now draw an analogy to your server. If your application gets a lot of load, you might have to distribute it to different servers. If you were using sessions, you'd suddenly had to replicate all sessions to all servers. The system would become even more complex and error prone.

Simply said, sessions don’t do what they’re designed to do without introducing a ton of **overhead**, and their functionality can easily be replicated using cookies, client caching, and other such solutions. There are, of course, situations in which sessions make sense, especially when servers wanted to store state without having even the slight potential of modified client runtime data.

For instance, FTP is stateful for a very good reason, as it replicates changes on both the client side and server side while delivering increased security due to the nature of the requested access. This is doable because a single person needs to access a single server for a single stated data transferral, even if the transferral involves multiple folders, files, and directories.

That’s not the case with something like a shared Dropbox, in which stateful sessions would cause the added complexity without adding value. In this case, stateless would be a much better choice.

Conclusion

We hope this has cleared up the difference between stateful and stateless architectures when it comes to APIs. Understanding this simple concept is the foundation upon which most architectures and designs are based upon — such lofty concepts such as RESTful design are based around these ideas, so having a sound conceptual framing is extremely important.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 07	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To Study data centric & client centric consistency model.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 07

Aim : To Study data centric & client centric consistency model.

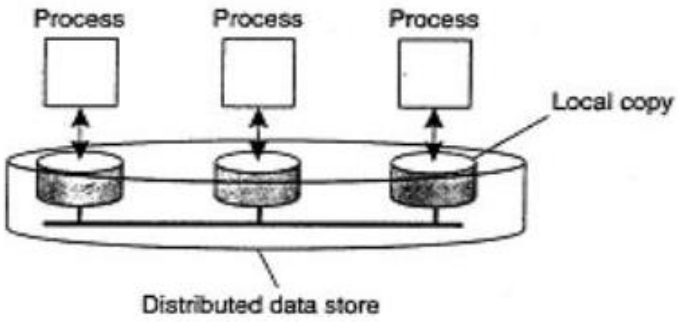
Theory :
To reduce access time of data caching is used. The effect of replication and caching increases complexity and overhead of consistency management.

Different Consistency models offer different degrees of consistency.

Depending on the order of operations allowed and how much inconsistency can be tolerated, so consistency models range from strong to weak.

Data centric model:

In this model it is guarantee that the results of read and write operations which is performed on data can be replicated to various stores located nearby immediately.



Fig(a) Data Centric model

Client centric model:

These consistency model do not handle simultaneous updates.

But, to maintain a consistent view for the individual client process to access different replicas from different locations has been carried out.

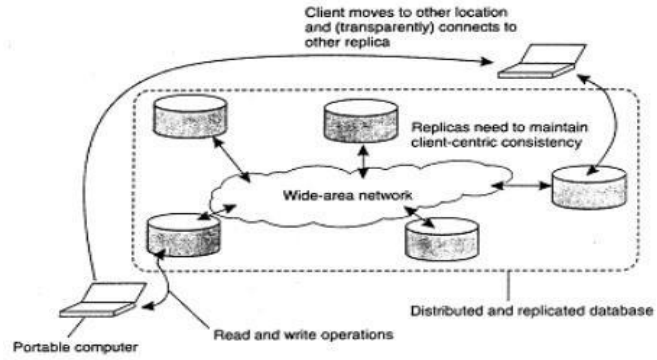


Fig (b) Client centric model

Data centric consistency models explanation

Strict consistency:

It is the strongest data centric consistency model as it requires that a write on a data be immediately available to all replicas.

This model states that "Any read on data item x returns a value corresponding to the result of the most recent write on x.

In a distributed system, it is very difficult to maintain a global time ordering, along with the availability of data which is processed concurrently at different locations, which causes delays for transferring of data from one location to another.

Thus Strict consistency model is impossible to achieve.

Eg P1 & P2 are two processes. P1 performs a write operation on its data item x and modifies its value to a.

Update is propogated to all other replicas

Suppose if P2 now reads and finds its value to be NIL due to the propogation delay. So the result of this value read by P2 is not strictly consistent.
But as the law of Strictly consistent model says that results should be immediately propogated to all the replicas as shown in figure below

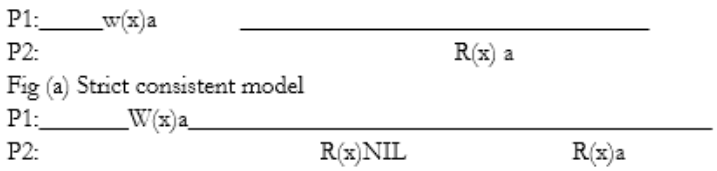


Fig (a) Strict consistent model

Fig (b) Propogation delay in strictly consistent model

Client centric model explanation Monotonic

Reads:

It states that "If a process P reads the value of a data item t, any successive read operation on x by that process at later time will always return that same or a recent value.

For eg Each time one connects to the email server (may be different replicas),the email server guarantees that all the time will always return that same or recent value.

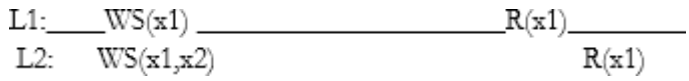


Fig (a) Monotonic read consistent

Here, in fig (a) operations WS(x1) at L1 before the second read operation. Thus WS(x1,x2) at L2 would see the earlier update.

Fig (b) A data store which is monotonic read inconsistent

Conclusion : We have successfully studied data centric & client centric consistency model.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 08	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- An introduction to naming services.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

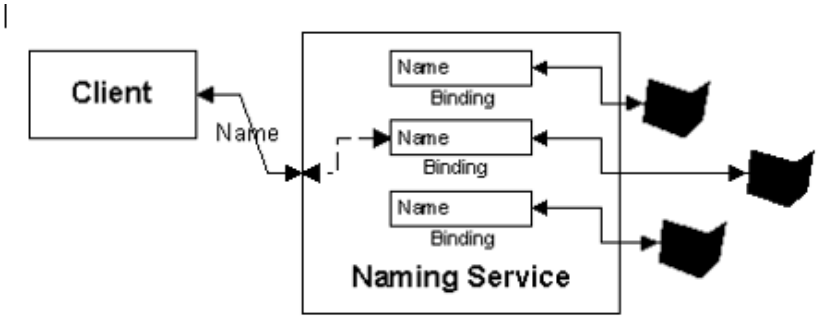
Faculty Signature with Date:-

Experiment No. 08

Aim : An introduction to naming services.

Theory :

The figure below depicts the organization of a generic naming service.



A generic naming service

A naming service maintains a set of *bindings*. Bindings relate names to objects. All objects in a naming system are named in the same way (that is, they subscribe to the same *naming convention*). Clients use the naming service to locate objects by name.

There are a number of existing naming services, a few of which I'll describe below. They each follow the pattern above, but differ in the details.

- **COS (Common Object Services) Naming:** The naming service for CORBA applications; allows applications to store and access references to CORBA objects.

- **DNS (Domain Name System):** The Internet's naming service; maps people-friendly names (such as www.etcee.com) into computer-friendly IP (Internet Protocol) addresses in dotted-quad notation (207.69.175.36). Interestingly, DNS is a *distributed*naming service, meaning that the service and its underlying database is spread across many hosts on the Internet.

- **LDAP (Lightweight Directory Access Protocol):** Developed by the University of Michigan; as its name implies, it is a lightweight version of DAP (Directory Access Protocol), which in turn is part of X.500, a standard for network directory services. Currently, over 40 companies endorse LDAP.

- **NIS (Network Information System) and NIS+:** Network naming services developed by Sun Microsystems. Both allow users to access files and applications on any host with a single ID and password.

Common features

As I mentioned earlier, the primary function of a naming system is to bind names to objects (or, in some cases, to references to objects -- more on which in a moment). In order to be a naming service, a service must at the very least provide the ability to bind names to objects and to look up objects by name.

Many naming systems don't store objects directly. Instead, they store references to objects. As an illustration, consider DNS. The address 207.69.175.36 is a reference to a computer's

location on the Internet, not the computer itself.

JNDI provides an interface that supports all this common functionality. I will present this interface later in this article.

Their differences

It's also important to understand how existing naming services differ, since JNDI must provide a workable abstraction that gets around those differences.

Aside from functional differences, the most noticeable difference is the way each naming service requires names to be specified -- its naming convention. A few examples should illustrate the problem.

In DNS, names are built from components that are separated by dots ("."). They read from right to left. The name "www.etcee.com" names a machine called "www" in the "etcee.com" domain. Likewise, the name "etcee.com" names the domain "etcee" in the top-level domain "com."

In LDAP, the situation is slightly more complicated. Names are built from components that are separated by commas (","). Like DNS names, they read from right to left. However, components in an LDAP name must be specified as name/value pairs. The name "cn=Todd Sundsted, o=ComFrame, c=US" names the person "cn=Todd Sundsted" in the organization "o=ComFrame, c=US." Likewise, the name "o=ComFrame, c=US" names the organization "o=ComFrame" in the country "c=US."

As the examples above illustrate, a naming service's naming convention alone has the potential to introduce a significant amount of theflavor of the underlying naming service into JNDI. This is not a feature an implementation-independent interface should have.

JNDI solves this problem with the Name class and its subclasses and helper classes. The Name class represents a name composed of an ordered sequences of subnames, and provides methods for working with names independent of the underlying naming service.

A look at JNDI naming

As I mentioned above, it's important to remember that JNDI is an *interface* rather than an *implementation.* This fact has some disadvantages -- you need access to an existing naming service (such as an LDAP service) and you need to understand something about how it works in order to play with JNDI. On the other hand, it does allow JNDI to integrate seamlessly into an existing computing environment where an established naming service holds sway.

JNDI naming revolves around a small set of classes and a handful of operations. Let's take a look at them.

Context and InitialContext

The Context interface plays a central role in JNDI. A context represents a set of bindings within a naming service that all share the same naming convention. AContext object provides the methods for binding names to objects and unbinding names from objects, for renaming

objects, and for listing the bindings.

Some naming services also provide subcontext functionality. Much like a directory in a filesystem, a subcontext is a context within a context. This hierarchical structure permits better organization of information. For naming services that support subcontexts, the Context class also provides methods for creating and destroying subcontexts.

JNDI performs all naming operations relative to a context. To assist in finding a place to start, the JNDI specification defines an InitialContext class. This class is instantiated with properties that define the type of naming service in use and, for naming services that provide security, the ID and password to use when connecting.

For those of you familiar with the RMI Naming class, many of the methods provided by the Context interface outlined below will look familiar. Let's take a look at Context's methods:

- **void bind(String stringName, Object object):** Binds a name to an object. The name must not be bound to another object. All intermediate contexts must already exist.
- **void rebind(String stringName, Object object):** Binds a name to an object. All intermediate contexts must already exist.
- **Object lookup(String stringName):** Returns the specified object.
- **void unbind(String stringName):** Unbinds the specified object.
- The Context interface also provides methods for renaming and listing bindings.
- **void rename(String stringOldName, String stringNewName):** Changes the name to which an object is bound.
- **NamingEnumerationlistBindings(String stringName):** Returns an enumeration containing the names bound to the specified context, along with the objects and the class names of the objects bound to them.
- **NamingEnumerationlist(String stringName):** Returns an enumeration containing the names bound to the specified context, along with the class names of the objects bound to them.
- Each of these methods has a sibling that takes a Name object instead of a String object. A Name object represents a generic name. The Name class allows a program to manipulate names without having to know as much about the specific naming service in use.

Conclusion : We have successfully studied about naming services.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 09	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Client Server implementation using CORBA.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 09

Aim : Client Server implementation using CORBA.

Theory :

Running the Hello World Application

Despite its simple design, the Hello World program lets you learn and experiment with all the tasks required to develop almost any CORBA program that uses static invocation.
This example requires a naming service to make the servant object's operations available to clients. The server needs an object reference to the naming service so that it can publish the references to the objects implementing various interfaces. These object references are used by the clients for invoking methods. The two options for Naming Services shipped with J2SE v.1.4 are tnameserv, a transient naming service, and orbd, which is a daemon process containing a Bootstrap Service, a Transient Naming Service, a Persistent Naming Service, and a Server Manager. This example uses orbd.

When running this example, remember that, when using Solaris software, you must become root to start a process on a port under 1024. For this reason, we recommend that you use a port number greater than or equal to 1024. The -ORBInitialPort option is used to override the default port number in this example. The following instructions assume you can use port 1050 for the Java IDL Object Request Broker Daemon, orbd. You can substitute a different port if necessary. When running these examples on a Windows machine, subtitute a backslash (\) in path names.

To run this client-server application on your development machine:

- Start `orbd`.

  To start `orbd` from a UNIX command shell, enter:

  ```
  orbd -ORBInitialPort 1050 -ORBInitialHost localhost&
  ```

  From an MS-DOS system prompt (Windows), enter:

  ```
  start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
  ```

  Note that `1050` is the port on which you want the name server to run. `-ORBInitialPort` is a required command-line argument. Note that when using Solaris software, you must become root to start a process on a port under 1024. For this reason, we recommend that you use a port number greater than or equal to 1024.

  Note that `-ORBInitialHost` is also a required command-line argument. For this example, since both client and server on running on the development machine, we have set the host to `localhost`. When developing on more than one machine, you will replace this with the name of the host. For an example of how to run this program on two machines, see The Hello World Example on Two Machines.

- Start the Hello server.

  To start the Hello server from a UNIX command shell, enter:

  ```
  java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost&
  ```

  From an MS-DOS system prompt (Windows), enter:

  ```
  start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
  ```

For this example, you can omit `-ORBInitialHost localhost` since the name server is running on the same host as the Hello server. If the name server is running on a different host, use `-ORBInitialHost` *nameserverhost* to specify the host on which the IDL name server is running.

Specify the name server (`orbd`) port as done in the previous step, for example, `-ORBInitialPort 1050`.

- Run the client application:
- `java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost`

For this example, you can omit `-ORBInitialHost localhost` since the name server is running on the same host as the Hello client. If the name server is running on a different host, use `-ORBInitialHost` *nameserverhost* to specify the host on which the IDL name server is running.

Specify the name server (`orbd`) port as done in the previous step, for example, `-ORBInitialPort 1050`.

- The client prints the string from the server to the command line:

```
Hello world!!
```

The name server, like many CORBA servers, runs until you explicitly stop it. To avoid having many servers running, kill the name server process after the client application returns successfully. To do this from a DOS prompt, select the window that is running the server and enter `Ctrl+C` to shut it down. To do this from a Unix shell, find the process, and kill it.

Program of HelloServer.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class HelloImpl extends HelloPOA {
 private ORB orb;

 public void setORB(ORB orb_val) {
  orb = orb_val;
 }

 // implement sayHello() method
 public String sayHello() {
  return "\nHello world !!\n";
 }
```

```
  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
}


public class HelloServer {

  public static void main(String args[]) {
    try{
      // create and initialize the ORB
      ORB orb = ORB.init(args, null);

      // get reference to rootpoa & activate the POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // create servant and register it with the ORB

    HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // get object reference from the servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // get the root naming context
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Use NamingContextExt which is part of the Interoperable
      // Naming Service (INS) specification.
  NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

  // bind the Object Reference in Naming
  String name = "Hello";
  NameComponent path[] = ncRef.to_name( name );
  ncRef.rebind(path, href);

  System.out.println("HelloServer ready and waiting ...");

  // wait for invocations from clients
  orb.run();
}

  catch (Exception e) {
    System.err.println("ERROR: " + e);
```

```
      e.printStackTrace(System.out);
    }

    System.out.println("HelloServer Exiting ...");

  }
}
```

****

Program of HelloClient.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{

static Hello helloImpl;

  public static void main(String args[])
   {
    try{
      // create and initialize the ORB
     ORB orb = ORB.init(args, null);

// get the root naming context
org.omg.CORBA.Object objRef =
        orb.resolve_initial_references("NameService");
// Use NamingContextExt instead of NamingContext. This is
// part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

// resolve the Object Reference in Naming
String name = "Hello";
helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

System.out.println("Obtained a handle on server object: " + helloImpl);
System.out.println(helloImpl.sayHello());
helloImpl.shutdown();

    } catch (Exception e) {
  System.out.println("ERROR : " + e) ;
      e.printStackTrace(System.out);
      }


    }

  }
```

OUTPUT : -

```
E:\exp4\Hello>idlj -fall Hello.idl

E:\exp4\Hello>javac HelloServer.java HelloApp/*.java
Note: HelloApp\HelloPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\exp4\Hello>javac HelloClient.java HelloApp/*.java
Note: HelloApp\HelloPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\exp4\Hello>start orbd -ORBInitialPort 1050 -ORBInitialHost
localhost


E:\exp4\Hello>start java HelloServer -ORBInitialPort 1050 -
ORBInitialHost localhost

E:\exp4\Hello>java HelloClient -ORBInitialPort 1050 -ORBInitialHost
localhost
Obtained a handle on server object:
IOR:000000000000001749444c3a48656c6c6f4170702f48656c6c6f3a312e30000000
0000010000000000000008a000102000000000f3139322e3136382e31302e3130310000
041300000000031afabcb00000000201fde1c4100000001000000000000000100000000
08526f6f74504f4100000000800000001000000001400000000000002000000010000
0020000000000001000100000002050100010001002000010109000000010001010000
000026000000020002

Hello world !!

**********************************************************
```

Conclusion : Successfully Implemented Client Server implementation using CORBA.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 10	
Subject : Distributed Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Study of DCOM.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No.10

Aim : Study of DCOM

Theory :

Introduction

Welcome to this tutorial. In this series, I will strip the mystique, the headache, and confusion from DCOM by giving you a comprehensive tutorial with a straight forward example. OK, no promises - but I will give it a good try. In this series of articles, I will show you how to use the following technologies:

- the ATL COM AppWizard;
- implementation of a Windows NT Service;
- MFC;
- ATL;
- smart pointers;
- Connection Points;
- MFC Support for Connection Points;
- MFC ClassWizard support for implementing Connection Points (yes, it's true!).

These will be used to develop a sample client/server system to say "Hello, world from < machine >" to the user! I haven't heard of anybody needing to do client/server development in order to say, "Hello, world!", but that's what I'm doing.

If you want to follow along with this tutorial and add code and use the Visual C++ Wizards as we go along, that's great. In fact, I very very highly recommend that, because otherwise this tutorial is a big waste of electronic ink (?). However, I follow along exactly with the tutorial myself, as I write it, and develop the code and use the Visual C++ wizards just as I say you should. The screenshots, in fact, are from my development of the files for each step! To download this already- developed code to compare with your own, simply click the 'Download the Step *n* Files - *n* KB" links at the top of each step. There's also an archive of the files for all the steps at the Questions and Answers page for this tutorial. I *still* recommend that you follow along with us as we go; this way, you can learn while you code.

Environment

The tutorial's steps were completed by the author on a Windows NT 4, SP 5 network with Visual C++ 6.0 SP 4. The author hasn't tested this on Windows 2000 because he doesn't have a copy. If anything in a particular step of this tutorial doesn't work or works differently under Windows 2000, please post a message to the message board. Pretty much anything which works under Windows NT 4 SP 5 should work the same under Windows 2000, because of backward-compatibility, unless Microsoft's been making changes.

What This Tutorial Assumes

I have a motto: never write anything if it asks too much of the reader and not enough of the writer. To this end, I am going to tell you straight away what this tutorial assumes that you know. I would also include a 'Who Needs This Tutorial" section, much like they include a "Who Needs to Read This Book," but I'll let you be the judge of whether or not you need something.

This tutorial assumes that:

- You know how to use a Windows-based computer;
- You know how to use AppWizard;
- You're familiar with basic MFC programming;

- You've heard about ATL, COM, and DCOM;
- You know how Services work under Windows NT and Windows 2000;
- And that you're developing all this on Windows NT 4.0 or higher.

I hope I'm not assuming too much of you. I'll guide you along doing this at the same level or a little higher as how Microsoft guided us all through the Scribble tutorial. That is, I won't hold your hand too much, but I won't confuse you either (at least, I don't *think* Microsoft's explanations are confusing...).

I welcome any and all input to how I explain things; to yell at me or praise me (I can take both), **post on the message boards at the bottom of this step.** Believe me, I will get your post. **This way, people can see both your question, and my answer.**

Conventions Used

Since we'll be referring to class names, symbol names, and such, it's good to follow a convention with this so that everything is consistent. Here's what we'll do:

Code Fragments

Code fragments will appear in their own yellowish blocks on the page. The lines of new code that you'll have to add will appear in **bold**. If you don't have to add anything, but I'm just pointing some code out for my health, then nothing will be in **bold**:

```
voidCClass::Func()
{
// NOTE: Add this code
CAnotherClassclass;    // And this code
class.AnotherFunc();    // And this code

// Done adding code
}
```

Fair enough? OK. How about when some functions are a bajillion lines long, and we're only interested in the three lines at the top of the function? Then I'll use an ellipsis (. . .), like the Visual C++ docs do. This just means that there's a bunch of code there that we'll ignore:

```
voidCClass::LongFunc()
{
    ...

    HRESULT hResult = S_OK;

CClassclass;
hResult = class.Func();



    ...
}
```

Variable Names

I confess, I am a big fan of Microsoft's version of Hungarian notation. I guess that's a failing of mine, but I went through the Scribble tutorial until it got involved with all the OLE stuff. This drilled the MS Hungarian notation into my brain, unfortunately. I am a strong supporter of using m_for member variables, Cfor classes, Ifor interfaces, and Dfor dispinterfaces, or those things that we use to fire off connection point events:

- m_szDisplayName
- CHelloWorld
- IHelloWorld
- DHelloWorldEventsinstead of _IHelloWorldEvents

Functions

When I refer to a function, method, or variable, I will show it in a fixed-width font. Whenever I refer to a compiler keyword, I'll also put it in a fixed-width font. Symbol names, regardless of who uses them, will be in the fixed-width font. Chris goes around shaking his fingers at us writers and making sure that we follow these conventions. (Just kidding, ha ha ha...) Here are some examples:

- Function()
- THIS_IS_A_SYMBOL
- ISayHello::SayHello()
- CServiceModule
- [IDL_attribute]
- __stdcalland typedef
- END_OBJECT_MAP()

Acknowledgements and Attributions

I would like to take just a quick second and acknowledge some sources and people which led to this, because without their work and contributions, I might still be in the dark. I would like to thank, in particular, Dr. Richard Grimes, who wrote the excellent book *Professional DCOM Programming*. Dr. Grimes is a highly knowledgeable authority on DCOM and COM programming and he has a talent for explaining things in a way that they're easy to understand.

Professional DCOM Programming, published by Wrox Press, very thoroughly covers DCOM with intelligent discussion, working samples, and demystification of the really complicated stuff. By this, I mean threading, security, IDL, marshaling, and the Microsoft Transaction Server, to name just a few. I highly recommend that you buy this book (it's $49.95), you don't have to have read it in order to understand this tutorial.

Also, I would like to acknowledge the contributors to various articles and columns in MSDN Magazine (formerly MSJ), whose work, reprinted in the MSDN Library, helped me through the jungle of DCOM. This included columns by George Shepard and Don Box, two very knowledgeable COM experts. Thanks also go to Charles Steinhardt, and Tony Smith, two authors who have written for Visual C++ Developer.

The Approach

We will proceed step by step, following the methodology employed by Microsoft for writing the explanation of the Scribble tutorial. In this tutorial, you will develop:

- A DCOM server, implemented as a Windows NT Service. This server will expose a 'SayHello()' method which will say hello to the client user with a connection point.

- An MFC client, with support for smart pointers and connection points made *easy!* We'll even use ClassWizard to implement the connection point handling (!).

Before we plunge into writing any software, it's always good to design (just a little!) what it's going to do! What we'll develop in this tutorial will work as shown in **Figure 1** below:

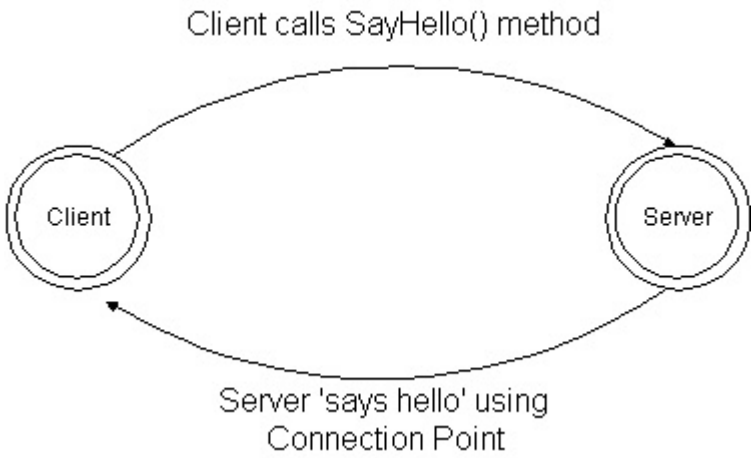


Figure 1. The way things will work.

As you can see in **Figure 1**, our software will follow a simple pattern:

1. The client will call a SayHello()method;
2. The server will say hello to the client by firing an event over the network;
3. The client will show the user a message in response to this event to indicate that the server did indeed say, "Hello!"

Enough With the Blathering; On to the Code!!

Yes, yes; I get the hint. We'll start by doing Step 1 of the tutorial in this article, and then you can go on to the next step of the tutorial by simply clicking the Next button at the bottom of this page.

- Step 1: Create the server, HelloServ, using the ATL COM AppWizard.
- Step 2: Modify the starter files provided by AppWizard.
- Step 3: Use the New ATL Object Wizard to add a simple COM object, the HelloWorldobject, to the server.
- Step 4: Modify the IHelloWorldinterface to include a SayHello()method.
- Step 5: Add an event method, OnSayHello(), to the connection point source interface, DHelloWorldEvents.
- Step 6: Build the server, and install it on the server computer.
- Step 7: Create a MFC client, HelloCli, which calls the server and handles the connection point event sink.

When the steps above are done, we'll have a living, breathing, fully functional DCOM application. At the bottom of the page for each step of this tutorial are Back and Next links, which you can use to go to the previous or next step. There's also a link which takes you to the 'Questions and Answers' page.

The Questions and Answers page gives me more space and the capability to display screenshots when I post answers to readers' questions. Feel free to post your question to The Code Project's message board at the bottom of each step. If there's a question that gets asked often or whose answer is not so simple, I'll see it and put it on the Questions and Answers page. This way, the

answers can be helpful to all of the readers of The Code Project. I'll then reply to the question on the message board and e-mail the author saying "I answered this question on the Questions And Answers page!".

Step 1: Create the Skeleton HelloServServer With the ATL COM AppWizard

OK; time to get started. Close anything you're working on in Visual C++ and save the changes. Then close any open Workspace too. Next, click the File menu, and then click New. This should bring up the New dialog box, as shown in **Figure 2** below. Click the ATL COM AppWizard in the list, and type 'HelloServ' for the Project Name:

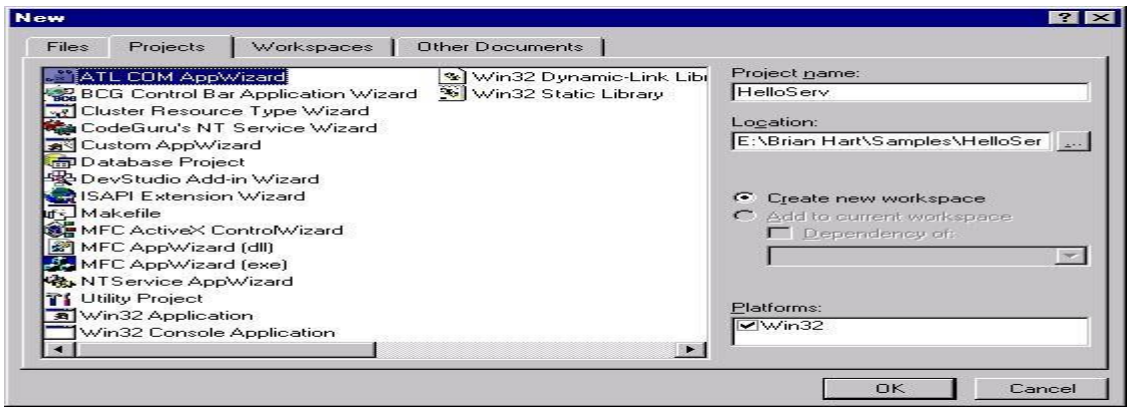


Figure 2. Selecting the ATL COM AppWizard in the New dialog box.

When the settings look the way you want them to, click OK. This brings up Step 1 of the ATL COM AppWizard (which only has one step). This is shown in **Figure 3**, below, with the 'Service' option button selected. This is OK; since the AppWizard doesn't let us set anything else, the next thing to do is to click Finish:

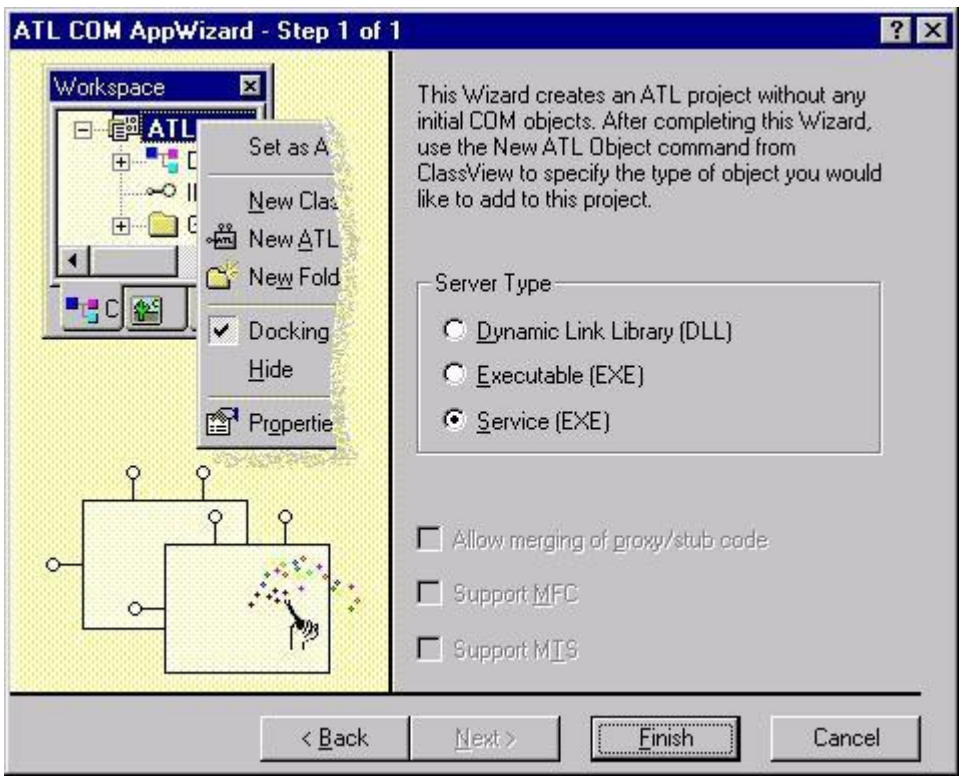


Figure 3. Selecting that we want AppWizard to give us a service.

Why A Service?

A Windows Service is a great type of process to have as your ATL/COM server. Windows Services, in my experience, have performed better as DCOM servers. Services can live while the machine is not logged on, whereas most EXE programs may not run. Also, Services are totally non-interactive, which is just fine when all you want your components to do is to perform routine system tasks, such as reading files or running programs, or providing monitoring services. You don't want windows popping up all willy-nilly, say, on your server computer sitting in a room in Ireland when you are running the client over in India. Plus, services are able to be started and stopped, and kept track of, using the Control Panel.

And for that matter, why an EXE and not DLL?

See above as far as the stand-alone EXE is concerned. A DLL is not very utilitarian as a remote server, because all DLLs *must* be mapped into the address space of an EXE process on the server system. A special EXE process which maps DCOM server DLLs into its address space, and then remotes the DLLs' component(s), is known as a *surrogate*. DLLs and surrogates are altogether complicated beasts to maintain and configure when you need remote access to your components. Especially since these are not reference-counted or monitored by the system, in case the number of clients drops to zero or if the surrogate hangs, leaving you dead in the water. So a Service is my favorite choice.

Finishing Up

After you click Finish, the AppWizard displays the New Project Information dialog box, shown in **Figure 4**. The New Project Information box only has a little bit of information in it; it just tells us that AppWizard is going to give us the starting point of a brand-new Windows NT Service which is also a DCOM server. However, this service doesn't have any COM objects yet. We'll add those in Step 3.

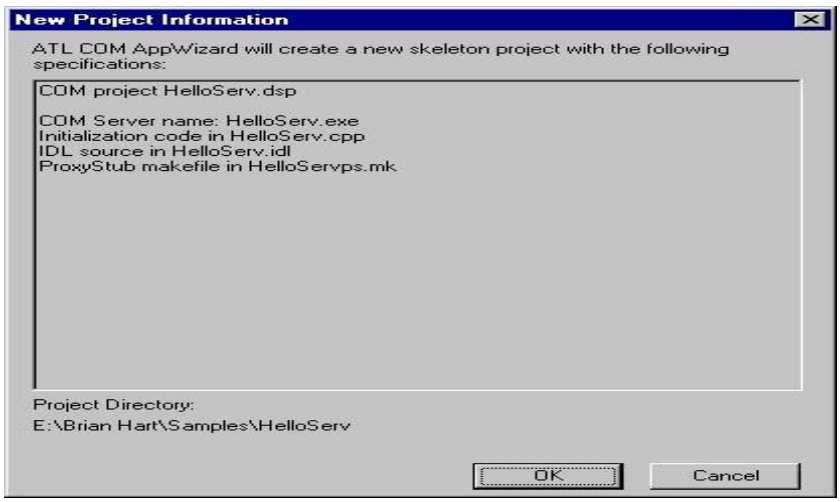


Figure 4. The New Project Information dialog box.

Click OK to have AppWizard generate the HelloServstarter program. Now, we are ready to go on to the next step, modifying the source files.

Step 2

Welcome to Step 2 of our DCOM tutorial. In this series, I will strip the mystique, the headache, and confusion from DCOM by giving you a comprehensive tutorial with a straightforward example. OK, no promises -- but I will give it a good try.

A diagram of how our software will eventually work is shown in **Figure 1**. The client calls a method on the server, which then fires an event back to the client using a connection point. This connection point's event sink is implemented in the client (using MFC and ClassWizard!!!), and the client shows its user a message telling the user that the server said "Hello!":

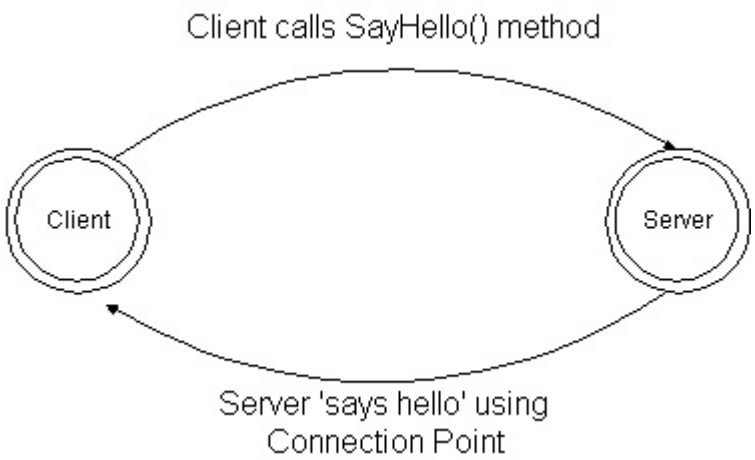


Figure 1. Diagram of our DCOM client/server set-up.

We're currently on Step 2 of the tutorial, where we modify some things in the starter source code provided to us by the ATL COM AppWizard. We do this to:

- Provide a user-friendly "Display Name" for this service;
- Add code to initialize security properly.

Step 2: Modify the Starter Files Provided by AppWizard

To start, we need to provide Windows with a user-friendly name it can use to display this service to the user. This is called the "Display Name" of the service. AppWizard already set up and specified a "Service Name" for us: HelloServ. For the user's sake, let's come up with something which is easier to understand: Hello World Server. To do this, we'll add a m_szDisplayName member variable to the CServiceModule class, and a String Table entry to hold the display name. We do things this way so that it's easy for us to change the display name of the service to whatever we wish.

First, double-click on the CServiceModule icon in ClassView. This will jump you to where the CServiceModule class is declared, in STDAFX.H. We need to add a m_szDisplayName member variable to the class. Move the cursor to the // data members section, and add the code shown below in **bold**:

Collapse | Copy Code

```
class CServiceModule : public CComModule
{
```

```
    ...

// data members
public:
TCHAR    m_szDisplayName[256];         // display name of service
    TCHAR    m_szServiceName[256];

    ...
};
```

Listing 1. Adding the m_szDisplayNamemember variable to the CServiceModuleclass.

The next thing to do is to add an entry to the String Table, IDS_DISPLAY_NAME, to hold the display name we want to use. **Figure 2**, shown below, illustrates adding the String Table entry. To do this, complete these steps:

1. Click the ResourceView tab.
2. Double-click the String Table folder to open it.
3. Double-click the String Table icon in the folder to open the String Table.
4. Find the blank entry in the list, and double-click it. The Properties window appears, as in **Figure 2**, below.
5. In the ID box, type IDS_DISPLAY_NAME.
6. Press TAB to move to the Caption box, which contains what the IDS_DISPLAY_NAME symbol maps to in your code.
7. In the Caption box, type Hello World Server, as shown in **Figure 2**.
8. Press Enter. This saves the new String Table entry to the String Table.

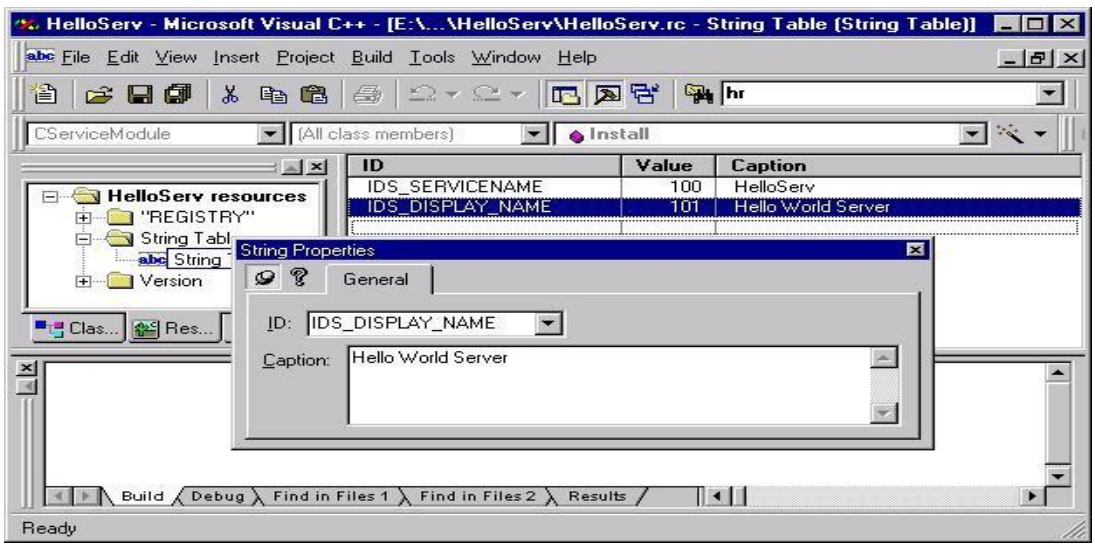


Figure 2. Adding the IDS_DISPLAY_NAMEentry to the String Table.

On to the next part of Step 2. We have a member variable, m_szDisplayName, which we want to fill with the contents of the String Table entry IDS_DISPLAY_NAME. To do this, double-click the CServiceModule::Init()function in ClassView, and then add the line shown in **bold**:

⊟Collapse | Copy Code

```
inlinevoidCServiceModule::Init(_ATL_OBJMAP_ENTRY* p, HINSTANCE h,
    UINT nServiceNameID, const GUID* plibid)
{
    ...
```

```
LoadString(h, nServiceNameID, m_szServiceName,
sizeof(m_szServiceName) / sizeof(TCHAR));

LoadString(h, IDS_DISPLAY_NAME, m_szDisplayName,
sizeof(m_szDisplayName) / sizeof(TCHAR));

    ...
}
```

Listing 2. Adding a LoadString()call to CServiceModule::Init().

Now we have made sure that the IDS_DISPLAY_NAMEstring gets loaded into the m_szDisplayNamemember variable of CServiceModule. The next thing to do is to change the call to CreateService()in the CServiceModule::Install() function. The call is shown in **Listing 3** below in **bold**. The call is again shown in **Listing 4**, also below, but this time the argument which you need to replace is shown in **bold**: Collapse | Copy

Code

```
inline BOOL CServiceModule::Install()
{
    ...

    SC_HANDLE hService= ::CreateService(
hSCM, m_szServiceName, m_szServiceName,
        SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
        SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
szFilePath, NULL, NULL, _T("RPCSS\0"), NULL, NULL);

    ...
}
```

Listing 3. The call to the Windows CreateService()function, as called by AppWizard.

Change the second passing of m_szServiceNameto m_szDisplayName, as shown in **Listing 4**: Collapse | Copy Code

```
inline BOOL CServiceModule::Install()
{
    ...

    SC_HANDLE hService= ::CreateService(
hSCM, m_szServiceName, m_szDisplayName,
        SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
        SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
szFilePath, NULL, NULL, _T("RPCSS\0"), NULL, NULL);

    ...
}
```

Listing 4. Changing the second m_szServiceNameto m_szDisplayName.

Finally, the last part of Step 2 is to initialize everything properly. Go to the place in the HELLOSERV.CPP file shown // PLACE THE CURSOR HEREcomment in **Listing 5**:

Collapse | Copy Code

```
#include "stdafx.h"
#include "resource.h"
#include <initguid.h>
#include "HelloServ.h"

#include "HelloServ_i.c"


#include <stdio.h>

CServiceModule _Module;

BEGIN_OBJECT_MAP(ObjectMap)
END_OBJECT_MAP()

// PLACE THE CURSOR HERE

LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2)
{
    ...
```

Listing 5. Where in the HELLOSERV.CPP file to place the cursor.

Add all of the following code at where I've just told you to place the cursor: Collapse | Copy

Code

```
extern"C" BOOL WINAPI InitApplication()
{
    HRESULT hResult = CoInitialize(NULL);
if (FAILED(hResult))
return FALSE;        // failed to initialize COM

// Turn security off so that everyone has access to us
CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_NONE,
        RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);

// Initialization successful
return TRUE;
}
```

Listing 6. Adding the InitApplication()function to the server. Next, we need to replace some of the code in CServiceModule::Run()with a call to the InitApplication()function which we added in **Listing 6** above. Using ClassView, go to the CServiceModule::Run()function, and delete the code shown in **bold**:

Collapse | Copy Code

```
voidCServiceModule::Run()
{
    ...

    HRESULT hr = CoInitialize(NULL);
//  If you are running on NT 4.0 or higher you can use the following call
//  instead to make the EXE free threaded.
```

```
//  This means that calls come in on a random RPC thread
//  HRESULThr = CoInitializeEx(NULL, COINIT_MULTITHREADED);

    _ASSERTE(SUCCEEDED(hr));

// This provides a NULL DACL which will allow access to everyone.
CSecurityDescriptorsd;
sd.InitializeFromThreadToken();
hr = CoInitializeSecurity(sd, -1, NULL, NULL,
        RPC_C_AUTHN_LEVEL_PKT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);
    _ASSERTE(SUCCEEDED(hr));

    ...
}
```

Listing 7. Code to delete from the implementation of CServiceModule::Run().

Now replace what I told you to delete with the code shown in **Listing 8** below. The code to add is shown in **bold**:

Collapse | Copy Code

```
voidCServiceModule::Run()
{
    ...

HRESULT hr = S_OK;

if (!InitApplication())
return;

    ...
}
```

Listing 8. Code to add in order to replace what we've deleted.

Notes From the Rear

We've now completed Step 2 of this tutorial. We added a display name to help the user, and we fixed the security-initialization code for the service.

Step 4: Modify the IHelloWorldInterface to Add the SayHello()Method

We're currently on Step 4 of this tutorial, where we finally add working code to our DCOM server. We'll add a method to the IHelloWorldinterface, and we'll call this method SayHello(). This method will get the network name of the host that it's executing on, plus it will call a as-yet-unimplemented function Fire_OnSayHello(), which in Step 5 we'll add as an event to our DHelloWorldEventsevent interface. This function will take a single [in] BSTRparameter, the name of the host. Anyway, enough of my jabber; let's plunge in:

Step 4: Modify the IHelloWorldInterface to Add the SayHello()Method

This step of the tutorial is really short. All we will do is add one method to our ISayHello interface, and implement it using the CHelloWorldATL class. Then we'll be ready to move on to

Step 5! Since the user of our client would like to have some indication as to what computer on their network this code ran on, we'll add some code to get the network name of that computer. The following listing, **Listing 1**, shows a piece of code which you can cut-and-paste into any application you wish. This code calls the Windows GetComputerName()function:

Collapse | Copy Code

```
    TCHAR szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;

if (!GetComputerName(szComputerName, &dwSize))
    {
// Display the cause of the error to the user with the _com_error class
// To use the _com_error class, you need to #include <comdef.h> in
// your STDAFX.H file

AfxMessageBox(_com_error(GetLastError()).ErrorMessage(), MB_ICONSTOP);
return/*whatever error code: -1 or E_FAIL or whatnot here*/;
    }

// Now szComputerName holds this computer's name
```

Listing 1. Calling the GetComputerName()function.

Let's now add the IHelloWorld::SayHello()method, and then add its code. To do this, right- click the IHelloWorld interface icon in ClassView, and click Add Method. The Add Method to
Interface dialog box appears. Type SayHelloin the Method Name box, and leave the Return Type set to HRESULT.

TIP: When doing DCOM programming and adding methods to interfaces, *always* set the Return Type of the method to HRESULT.
This allows DCOM to report network errors and other status to the client.

Anyway, getting back to what we're doing, when you're done filling in the Add Method to Interface dialog box, it should look like that shown in **Figure 1**, below:

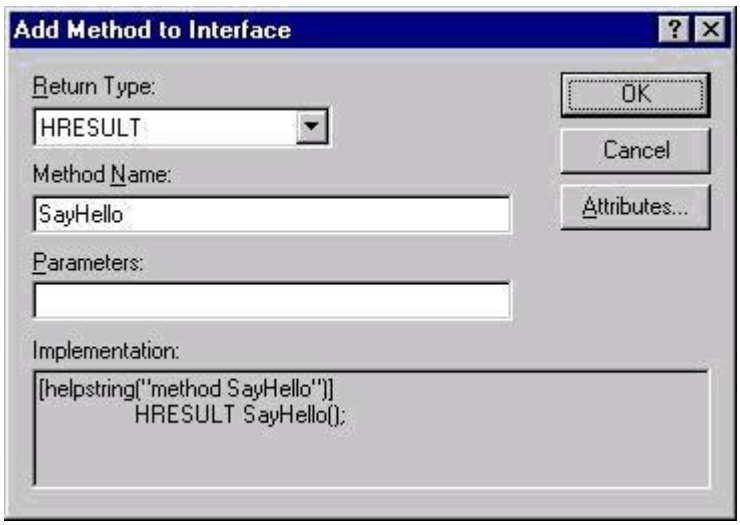


Figure 1. Adding the SayHello()method to the IHelloWorldinterface.

Click OK. When you do, the Add Method to Interface dialog will add code in all the right places to make sure that when a call to the IHelloWorld::SayHello()method comes in over the wire,

the CHelloWorld::SayHello()member function will get called and executed. After the method has been added, ClassView should resemble that shown in **Figure 2** below:

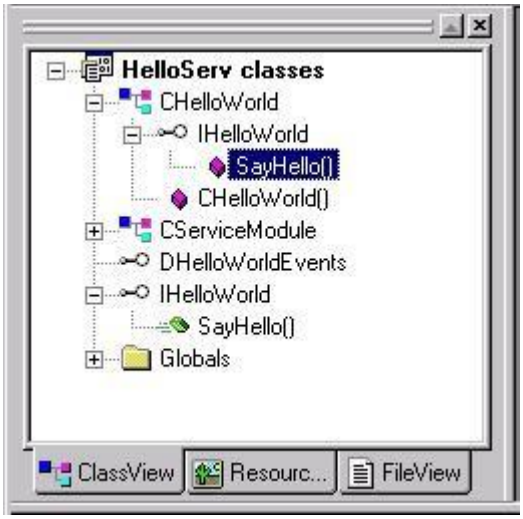


Figure 2. ClassView after the SayHello()method has been added.

Look at **Figure 2**. See the highlighted item? Double-click that item in your ClassView to open up the CHelloWorld::SayHello()member function. This function implements the IHelloWorld::SayHello()method. Let's add some code, shown here in **bold**, to implement the method:

Collapse | Copy Code

```
STDMETHODIMP CHelloWorld::SayHello()
{
// Get the network name of this computer
    TCHAR szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;

if (!GetComputerName(szComputerName, &dwSize))
return E_FAIL;    // failed toget the name of this computer

    // TODO: Add more code here

return S_OK;
}
```

Listing 2. Adding code to implement the SayHello()method.

Notes From the Rear

Notice the line which says // TODO: Add more code here? This tells us that we're not done implementing yet; according to the design, this method should fire off some sort of event back to the client.

Connection Points Demystified

Before we plunge in with Step 5 of our tutorial, let's just take a moment for me to rip the shrouds of mystery off of Connection Points. **Figure 1** below shows a generic scenario which is true for COM, DCOM, and even function call backs, for goodness' sake.

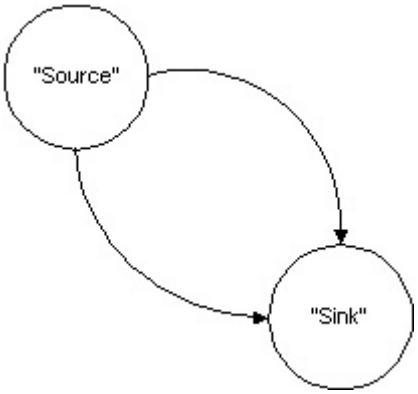


Figure 1.A source and a sink.

This involves two objects, a "source" and a "sink". Think of the "source" like the water faucet of the kitchen sink at home. You turn a handle, and stuff comes out of it (hopefully water). Where does it go? If nothing's backed up, this water flows down into the bottom and goes into the drain (which can be thought of as the "sink"). OK, so things flow *from* the source, *to* the sink. In the kitchen
sink analogy above, this is water. However, I've never seen a computer network system run with water flowing across the wires, so obviously something else is at work in DCOM.

In DCOM, there is a "client," somewhere on the network, and there is a "server," also somewhere on the network. Without the use of connection points, things flow only one way: method calls
replace our water, the client replaces our faucet, and the server replaces the drain. This is *way oversimplifying things*, but the user "turns a handle" (that is, clicks a button, for example), and "stuff" (that is, method calls) "comes out of" the client. This "stuff that comes out" then "flows"
over the network using DCOM. These calls "flow" to the server, which then collects them and acts like the "drain", or our sink. Here's **Figure 2**, which is almost exactly like **Figure 1**, but puts the client in place of the "source" and the server in place of the "sink," with the network in between:

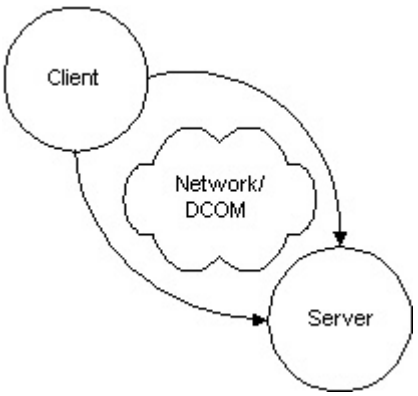


Figure 2. Our client and server as the source and the sink.

OK, so now we have method calls flowing like water; wonderful. However, when the client calls methods, the server does all kinds of things that might be interesting to clients. So the server fires events all over the place. If our client doesn't care if the server fires events, it will just ignore them. However, if it cares, it will Advise()the server. Then the source-sink relationship of **Figure 2** can be thought of in reverse:

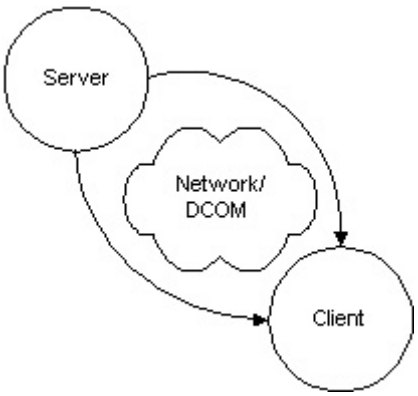


Figure 3. The reverse of **Figure 2**.

Connection points come in when you have the following happening:

1. The client is the *source* of a method call,
2. The server *sinks* (that is, acts as the sink for) the method call.
3. An "event call" comes out of the now-server-as-*source*.
4. The client *sinks* the event call and does something.

As you can see, this is a round-trip. A method call goes *from* the client *to* the server, and then an event call goes *from* the server, *to* the client, as seen in **Figure 4**.

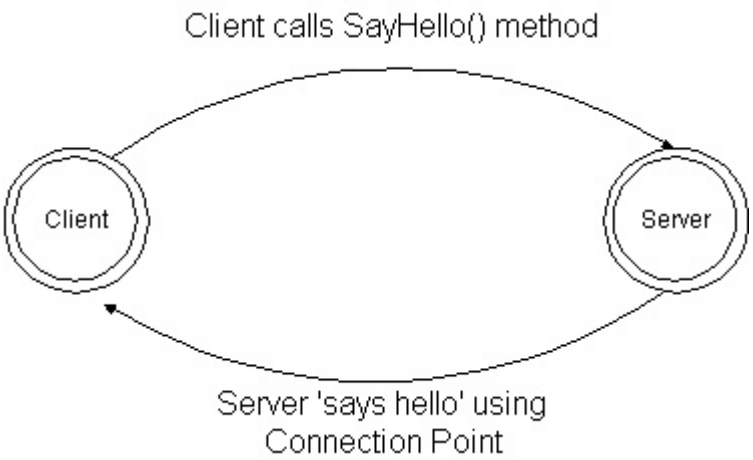


Figure 4.A round-trip.

The Advise() step is done before item number 1 above, and the Unadvise() step (where the client goes back to being aloof) happens after item number 4. The points of contact on both the client and server and the Advise()ing and Unadvise()ing that happens all together form...

A CONNECTION POINT!!

Whew... what a revelation... Let's start Step 5 before I get too carried away...

Step 5: Add the OnSayHello Event to the Event Source Interface

DHelloWorldEvents

Let's plunge in, shall we? To add an event to the source, it's really, really *easy*. Just use the Visual C++ Wizards! Open up ClassView, and right-click the DHelloWorldEventsicon, and then click Add Method. The Add Method to Interface dialog box appears. Type OnSayHelloin the Method Name box, and type [in] BSTR bstrHostin the Parameters box, as shown in **Figure 5**, below.

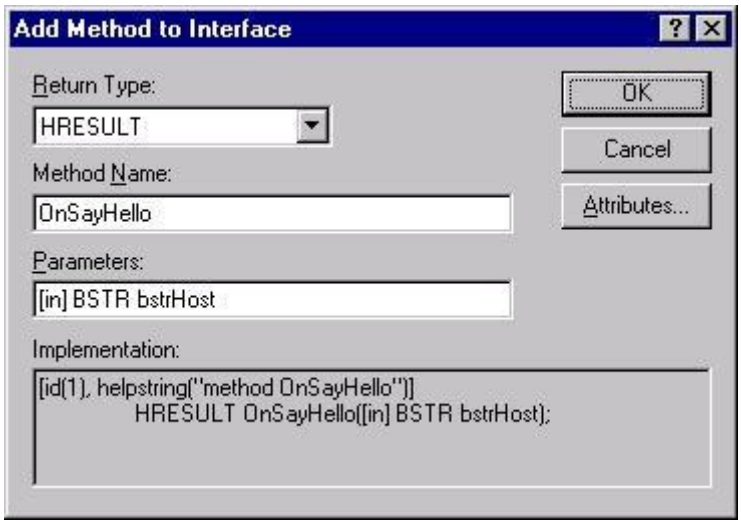


Figure 5. Adding the OnSayHello()event to the DHelloWorldEventsevent interface.

Once you're done, click OK. ClassView should resemble **Figure 6** below. Now click FileView, and find the HelloServ.idl file, under the **Source Files** folder. Right-click that baby, and then
choose Compile. Watch the compiler work away in the Output window, and wait until the build is complete.

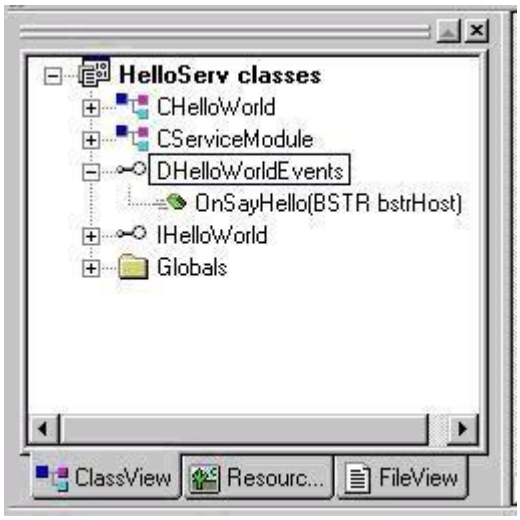


Figure 6. ClassView after adding the OnSayHelloevent.

Once the build has been finished, click on ClassView. Right-click the CHelloWorldclass, and then click Implement Connection Point. The Implement Connection Point dialog box appears. If you
haven't compiled the IDL file yet like I told you to, Visual C++ will prompt you to do so. **Figure 7**, below, shows you how to select that you want to make the server able to fire off its OnSayHello event:

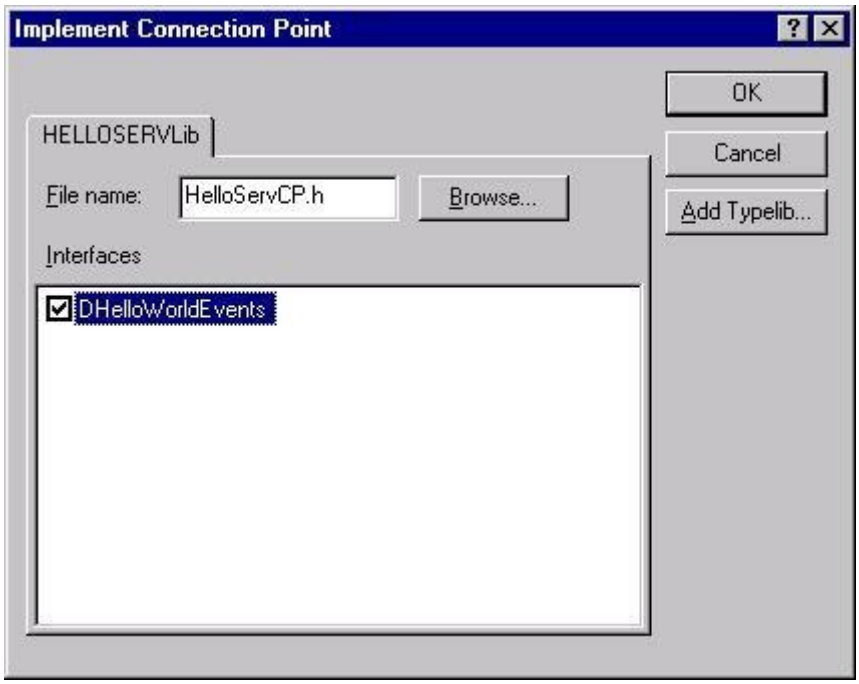


Figure 7. Specifying that we want to implement Connection Points for the DHelloWorldEvents event interface.

When everything looks like **Figure 7**, click OK. The Visual C++ IDE will now generate all the server-side code you need for Connection Points. Each time you change an event in the DHelloWorldEventsevent interface, you need to do the (1) compile the IDL, (2) right-click CHelloWorldand choose Implement Connection Point, (3) check the box by DHelloWorldEvents, and (4) click OK steps.

Notes From the Rear

The final part of Step 5 which we have to take care of is firing the event. Remember, we declared the OnSayHello()event in the IDL file as:

Collapse | Copy Code

```
HRESULT OnSayHello(BSTR bstrHost);
```

Listing 1. The declaration of the OnSayHello()event.

To fire the event from any CHelloWorldmember function, just call Fire_OnSayHello(). It's a member function of a new base class, CProxyDHelloWorldEvents<>that the Implement Connection Points dialog box just added for us. To this end, let's add code to the CHelloWorld::SayHello()function to fire the event to the client:

Collapse | Copy Code

```
STDMETHODIMP CHelloWorld::SayHello()
{
USES_CONVERSION;

    // Get the network name of this computer
    TCHAR szComputerName[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD dwSize = MAX_COMPUTERNAME_LENGTH + 1;

if (!GetComputerName(szComputerName, &dwSize))
return E_FAIL;    // failed toget the name of this computer
```

```
// Say Hello to the client
Fire_OnSayHello(T2OLE(szComputerName));

return S_OK;
}
```

Listing 2. Code to add to finish the CHelloWorld::SayHello()member function. That's it! We're finished with

Step 5.

We're currently on Step 6 of this tutorial, where we build the server and also build and register the Proxy-Stub DLL which goes along with it. Let's plunge in:

Step 6: Build the Server and Install It on the Server Computer

When you have reached this step, it's time to build our DCOM server, which is implemented as a Windows NT Service. Before we click that Build button, there are some things to do first. We start by making a few changes to the project settings, add a Custom Build Step to the project in order to build and register our Proxy-Stub DLL, and then we will make sure and change the configuration we're using. After doing all of this, we will be ready to click the Build button.

To change the project settings, click the Project menu, and then click Settings. Click the Custom Build tab; scroll over if you can't see it. Make sure you select the **Win32 Release MinDependency** configuration in the Settings For drop-down. After you've done that, erase everything in all of the fields of the Custom Build tab, so that what you have matches **Figure 1** below:

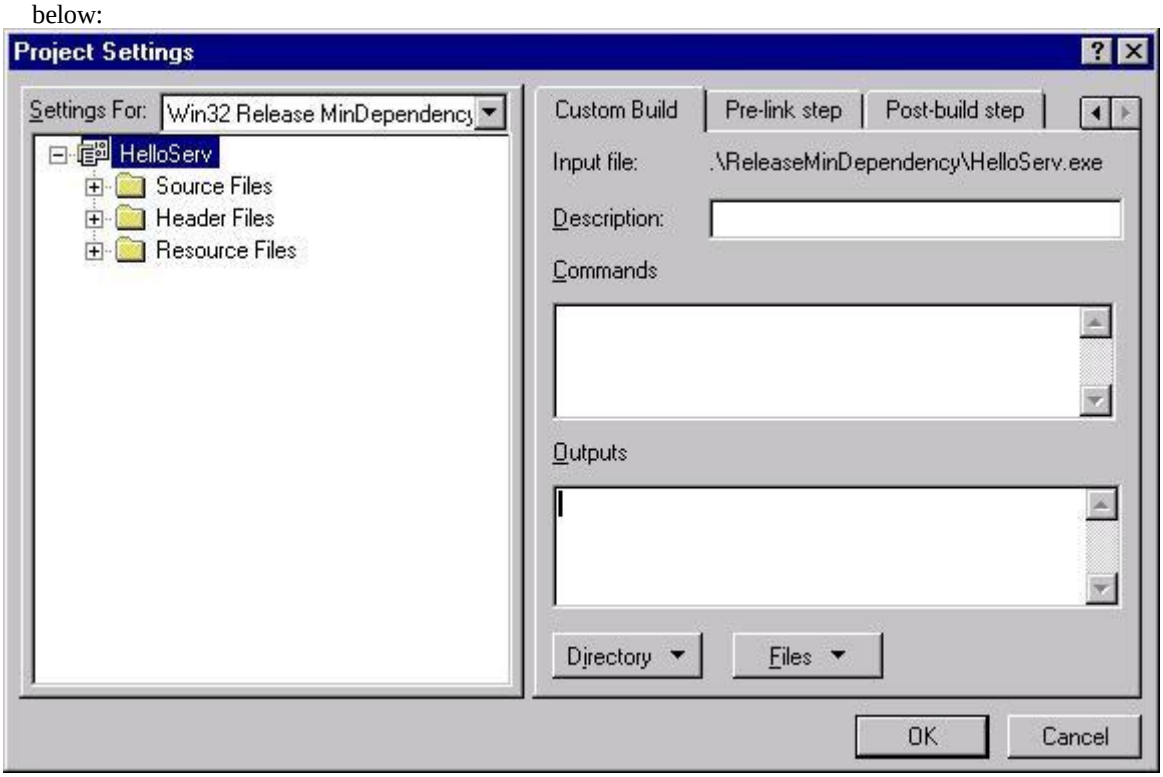


Figure 1. Removing the Custom Build step in the Project Settings dialog box.

Next, click the Post-Build Step tab. Again, before making changes, make sure you have the **Win32 Release MinDependency** configuration selected in the Settings For dropdown, as illustrated by **Figure 2**:

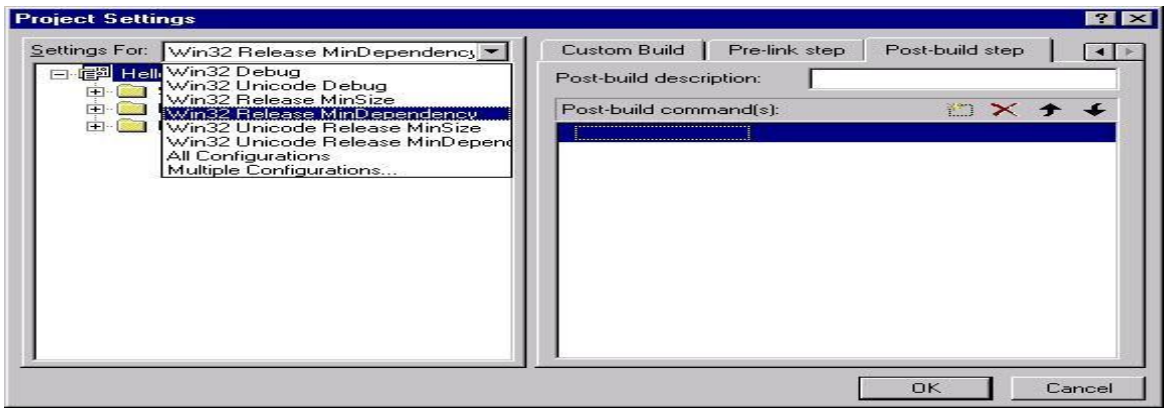


Figure 2. Making sure that **Win32 Release MinDependency** is selected.

Then type *Building and registering Proxy-Stub DLL...* in the Post-Build Description box, and then type the following lines into the Post-Build Command(s) area, to match **Figure 3**:

Collapse | Copy Code

```
start /wait nmake -f
HelloServps.mk regsvr32
```

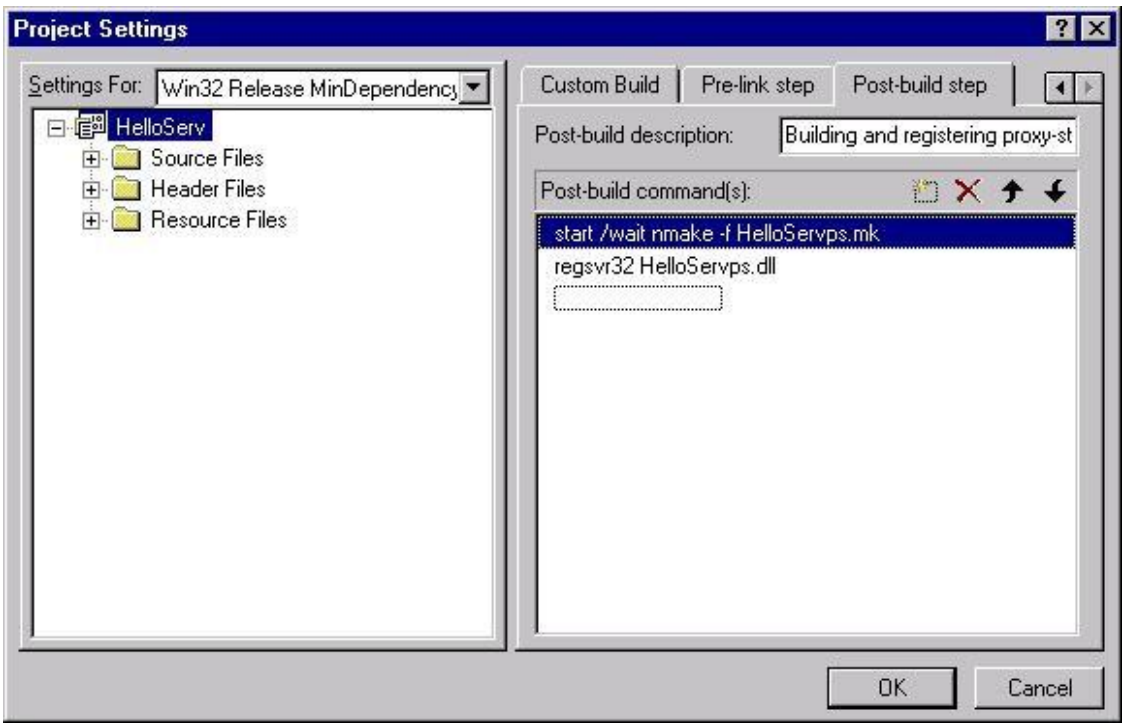


Figure 3. Specifying the Post-Build Step settings.

The last thing to make sure to do before beginning the build is to make sure that the right configuration is the active configuration. In our case, this is the **Win32 Release MinDependency** configuration. Click Build on the menu bar, and then click Set Active Configuration. This brings up
the Set Active Configuration dialog box, as shown in **Figure 4**. Click the *HelloServ - Win32 Release MinDependency* entry in the listbox, and then click OK.

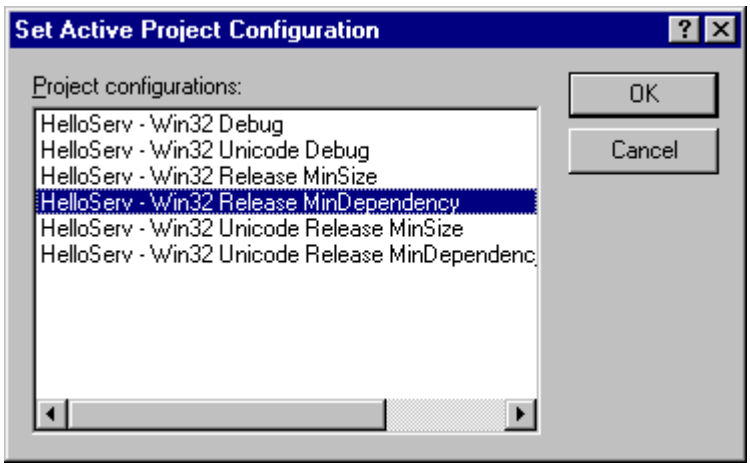


Figure 4. Selecting the **Win32 Release MinDependency** configuration.

Building and Installing the Server, and Other Notes From the Rear

At last! Now we're ready to build. Click that good ol' Build button on the toolbar, and watch the magic happen. When everything's done, you should have a *HelloServ.exe* EXE file in the
\ReleaseMinDepenedency subfolder of the project, and you should also see a *HelloServps.dll* DLL in the main project directory. Copy those two files to a floppy disk, and then put those in the *C:\Winnt\System32* directory on the computer you want to use as the server. Make sure the server machine is running Windows NT 4.0 Workstation or Server, or Windows 2000. Then, using the Run dialog box from the Start menu, run the following command lines, in this order:

1. *HelloServ /Service*
2. *regsvr32 HelloServps.dll*

Now, we'll use your development machine (the one you're following this tutorial with) as the client computer. To proceed, however, we'll need to follow these steps if the client machine is running either Windows NT 4.0 or Windows 2000:

1. Copy the *HelloServ.exe* and *HelloServps.dll* files from the floppy disk to the *C:\Winnt\System32* directory of the client machine.
2. Click the Start button, and then click Run.
3. Run the command line *HelloServ /Service.*
4. Click the Start button, and then click Run again.
5. Run the command line *regsvr32 HelloServps.dll.*

If your client machine is *not* running either Windows NT or Windows 2000, then you have to follow these steps:

1. Make sure that the DCOM98 extensions, available here, are installed.
2. Copy the *HelloServ.exe* and *HelloServps.dll* files from the floppy disk to the *C:\Windows\System* directory of the client machine.
3. Click the Start button, and then click Run.
4. Run the command line *HelloServ /RegServer*.
5. Click the Start button, and then click Run again.
6. Run the command line *regsvr32 HelloServps.dll*.

Now we are ready to proceed with Step 7. To move to Step 7, click Next below. If you need to go back to Step 5, click Back. If you have questions or problems, try clicking *Questions and Answers* below to jump to a page which offers some help.

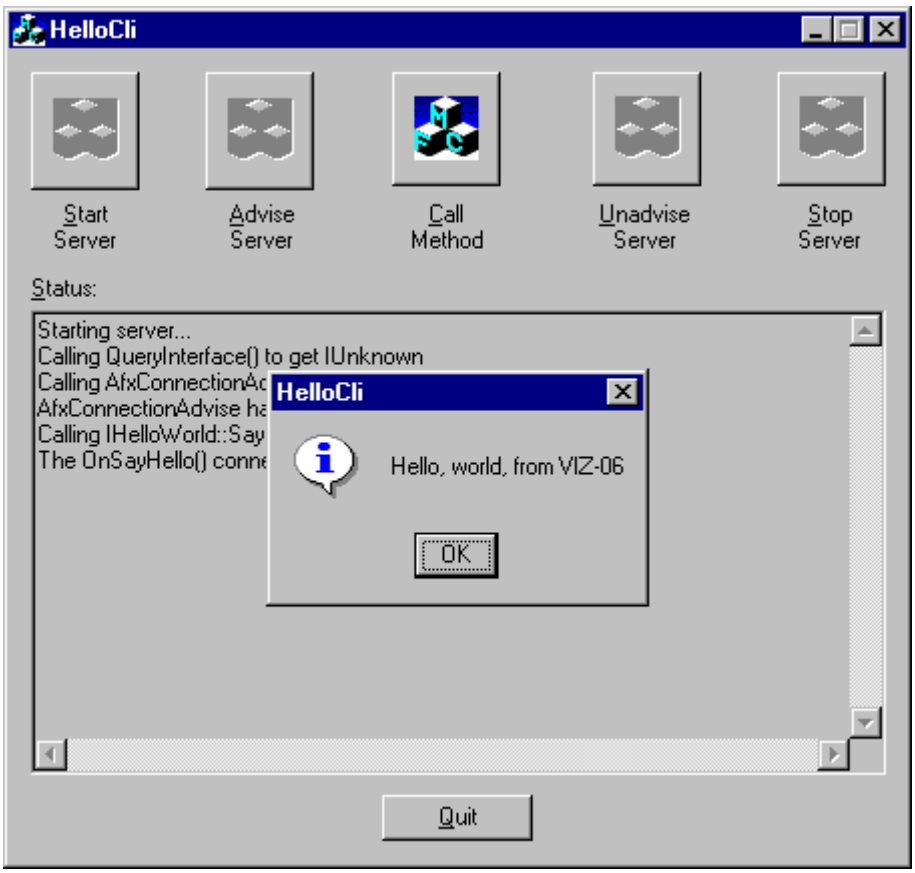


Figure 1. Image of the HelloCli sample program.

Introduction

We're currently on Step 7 of this tutorial (the last!), where we put together a little MFC program, called HelloCli, to test our server. Let's plunge in:

Step 7: Create a MFC HelloCli Client to Test the Server

As you can see from the screenshot above, I built a dialog-based application using MFC AppWizard (EXE). I added a status window to report status, so I can see just where errors occurred, and I also successfully handled Connection Points. To add text to the status window, which is just an Edit control with the read-only style set, I added a CString member variable, m_strStatus to the
dialog class with ClassWizard, and then anytime I needed to add a message line to the edit control, it was made easy with this code:

Collapse | Copy Code

```
m_strStatus += "This is a status line for the edit control\r\n";
UpdateData(FALSE);
```

Listing 1. Adding a status line to the edit control.

We add on text to the contents of m_strStatus with the += operator of CString, and then we call UpdateData(FALSE) to move the contents of the member variable from the variable to the edit control.

To start my sample project, I brought up the New dialog box, clicked 'MFC AppWizard (EXE)' in the list, and then typed 'HelloCli' for the name of my project. After completing AppWizard, I opened the STDAFX.H file and added the line shown in **bold** in **Listing 2**:

Collapse | Copy Code

```
#if !defined(AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_)
#define AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_

#if _MSC_VER >1000
#pragma once
#endif// _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#define _WIN32_WINNT        0x0400

#include <afxwin.h>      // MFC core and standard components
#include <afxext.h>      // MFC extensions
#include <afxdisp.h>     // MFC Automation classes
#include <afxdtctl.h>    // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>            // MFC support for Windows Common Controls
#endif// _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
//  the previous line.

#endif// !defined(AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_)
```

Listing 2. Adding the #define _WIN32_WINNT line to STDAFX.H so that DCOM works.

The next thing to do is to add something to let the client know about the interfaces and everything else that the server supports. There are two ways we can do this:

- Using #import to bring in the **type library** of the server. This file, HelloServ.tlb, is produced by MIDL when it compiles the HelloServ.idl file.
- Using #include "HelloServ.h" to just include the C++ and C declarations of interfaces. This is nice, but then you have to also define, in your code, all the GUIDs that the server responds to. These are CLSID_HelloWorld, IID_IHelloWorld, DIID_DHelloWorldEvents, and LIBID_HELLOSERVLib. If you use #import, this is done for you.

I like the idea of using #import, because of not only what was explained above, but also because you get to use smart pointers with #import, too. Be careful, though; we have a custom (that is, IUnknown-derived) interface that our server uses. We can use #import just fine in this example since the IHelloWorld::SayHello() method takes no parameters. If the IHelloWorld::SayHello() method took parameters, and they weren't of OLE-Automation-compatible types, then we would have to skip using #import, because it will only recognize those types. However, if you mark your custom interface with the [oleautomation] attribute and use OLE Automation-compatible types in your methods, this will work.

With custom interfaces, it's generally a better idea to use the second method above. However, like I said earlier, we'll go ahead and use #import this time because our method doesn't take any parameters. So this means that we need to copy the HelloServ.tlb file to our HelloCli project folder from the HelloServ project folder, and then add an #import line somewhere. How about in good ol' STDAFX.H again? We'll also add #include lines for atlbase.h and afxctl.h, since

these files give us support for things we'll use later on. Doing all this in STDAFX.Hwill help us when we build our program to keep the build time down, too:

Collapse | Copy Code

```
#if !defined(AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_)
#define AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_

#if _MSC_VER >1000
#pragma once
#endif// _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#define _WIN32_WINNT        0x0400

#include <afxwin.h>       // MFC core and standard components
#include <afxext.h>       // MFC extensions
#include <afxdisp.h>      // MFC Automation classes
#include <afxdtctl.h>     // MFC support for Internet Explorer 4 Common
//  Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>       // MFC support for Windows Common Controls
#endif// _AFX_NO_AFXCMN_SUPPORT

#include <atlbase.h>    // Support for CComPtr<>
#include <afxctl.h>     // MFC support for Connection Points

#import"HelloServ.tlb"no_namespacenamed_guidsraw_interfaces_only

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
// the previous line.

#endif// !defined(AFX_STDAFX_H__8495B5E0_67FF_11D4_A358_00104B732442__INCLUDED_)
```

Listing 3. Adding other needed code to STDAFX.H.

Bring in the Connection Point

NOTE: This only works if the event source interface is a dispinterface, like our DHelloWorldEventsinterface.

The next thing to do is to use ClassWizard to give us a class which will implement our connection point for us (!). Yes, we've finally arrived!! To do this, open up ClassWizard, click the Message Maps tab, click Add Class, and then click New, as shown below in **Figure 2**:

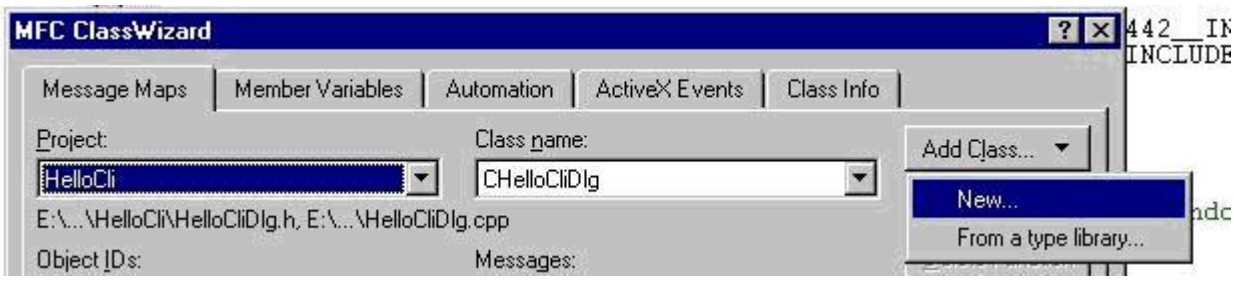


Figure 2. Adding a new class with ClassWizard.

The next thing to do is to specify the new class we want to add to our project. Since this class is (kind of) implementing an interface, that is, the DHelloWorldEventsdispinterface, we'll call this class the CHelloWorldEventsclass. Next, we specify that we want to derive this class from the MFC CCmdTargetclass, which helps us with all the COM implementation. People have often said that "MFC doesn't really have any COM support besides that needed for OLE and UI stuff. And
then, it only does dispinterfaces." None of the preceeding sentence is entirely correct. MFC is great at helping us out with UI stuff, but I have seen example code (that works) where MFC is used to
implement any interface you like, even in COM servers with non-dispinterface and non-IDispatch interfaces! The CCmdTargetclass is the key.

Anyway, enough of my blathering. The last thing to do before we can click OK in the New Class
dialog box is to click the Automation option button. This turns on the support in CCmdTargetthat we need to use; don't worry, choosing this won't even add so much as an .ODL file to your project, and you needn't have checked 'Automation' in AppWizard to use this. When everything in the New Class dialog box is as it should be, it should look like **Figure 3**, below:

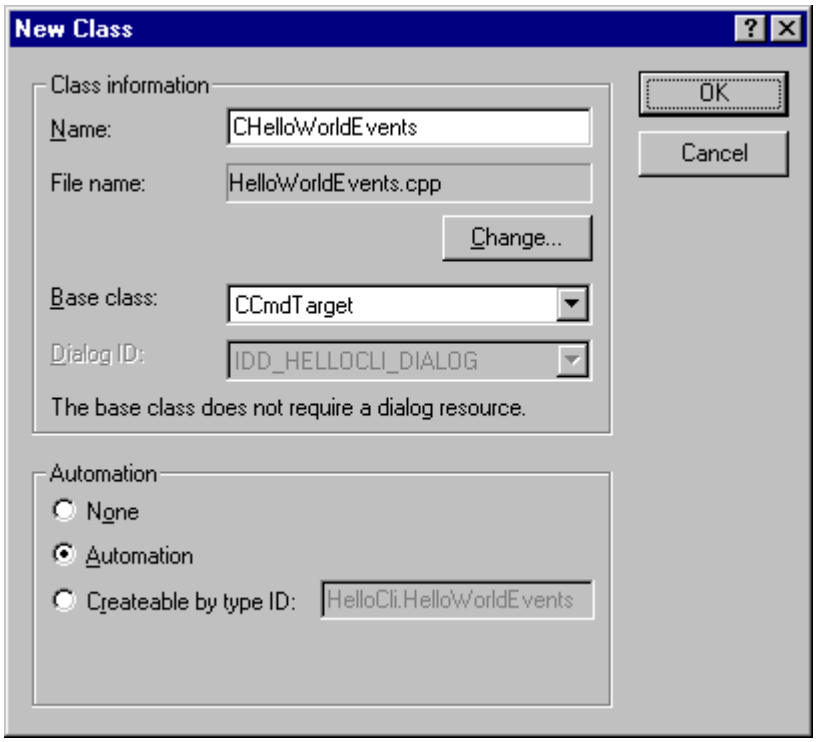


Figure 3. Specifying the settings for our new CHelloWorldEventsclass in ClassWizard.

ClassWizard will add the CHelloWorldEventsclass to your project, but it will whine because you didn't specify Automation support in AppWizard. Since you didn't, your project doesn't have a HelloCli.odlfile. Too bad for ClassWizard; it shows you the protest message below, but you can click OK and ignore it:

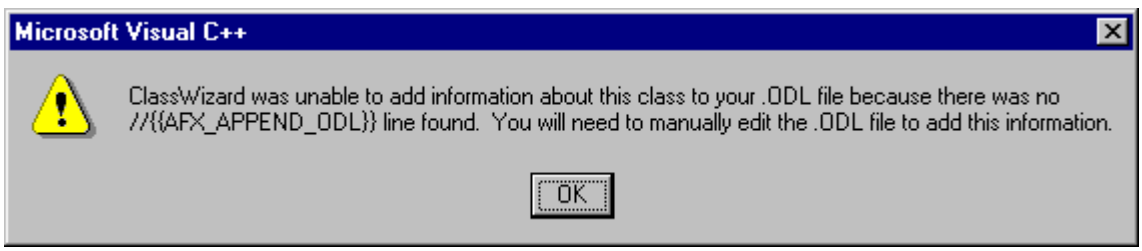


Figure 4. ClassWizard should just grow up, and quit its whining; but, oh well... Ignore this warning and click OK.

Make Changes to CHelloWorldEvents

ClassWizard, helpful as it is, did make one booboo that we'll want to erase. Open the HelloWorldEvents.cppfile and remove the line shown below in **bold**: Collapse | Copy Code

```
BEGIN_MESSAGE_MAP(CHelloWorldEvents, CCmdTarget)
//{{AFX_MSG_MAP(CHelloWorldEvents)
// NOTE - the ClassWizard will add and remove mapping macros here.
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(CHelloWorldEvents, CCmdTarget)
//{{AFX_DISPATCH_MAP(CHelloWorldEvents)
// NOTE - the ClassWizard will add and remove mapping macros here.
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()

// Note: we add support for IID_IHelloWorldEvents to support typesafe binding
//  from VBA.  This IID must match the GUID that is attached to the
//  dispinterface in the .ODL file.

// {B0652FB5-6E0F-11D4-A35B-00104B732442}
staticconst IID IID_IHelloWorldEvents =
{ 0xb0652fb5, 0x6e0f, 0x11d4, { 0xa3, 0x5b, 0x0, 0x10, 0x4b, 0x73, 0x24, 0x42 } };
```

Listing 4. Delete the lines of code that are shown in **bold**.

Next, find the code shown in **bold** in **Listing 5**, below. We're going to replace it with the DIID (**D**isp**I**nterface**ID**) of the DHelloWorldEvents interface:

Collapse | Copy Code

```
BEGIN_INTERFACE_MAP(CHelloWorldEvents, CCmdTarget)
    INTERFACE_PART(CHelloWorldEvents, IID_IHelloWorldEvents, Dispatch)
END_INTERFACE_MAP()
```

Listing 5. The code to look for, shown in **bold.**

Replace IID_IHelloWorldEventswith DIID_DHelloWorldEvents, as shown in **Listing 6**: Collapse | Copy Code

```
BEGIN_INTERFACE_MAP(CHelloWorldEvents, CCmdTarget)
        INTERFACE_PART(CHelloWorldEvents, DIID_DHelloWorldEvents, Dispatch)
END_INTERFACE_MAP()
```

Listing 6. Putting DIID_DHelloWorldEventsin place of the Class-Wizard-added IID_IHelloWorldEventsidentifier.

Now, we're going to use ClassWizard to add the handler function which gets called when the server fires the OnSayHello event. Bring up ClassWizard, and select the Automation tab. Make sure that CHelloWorldEvents is selected in the Class Name box, as shown in **Figure 5** below:

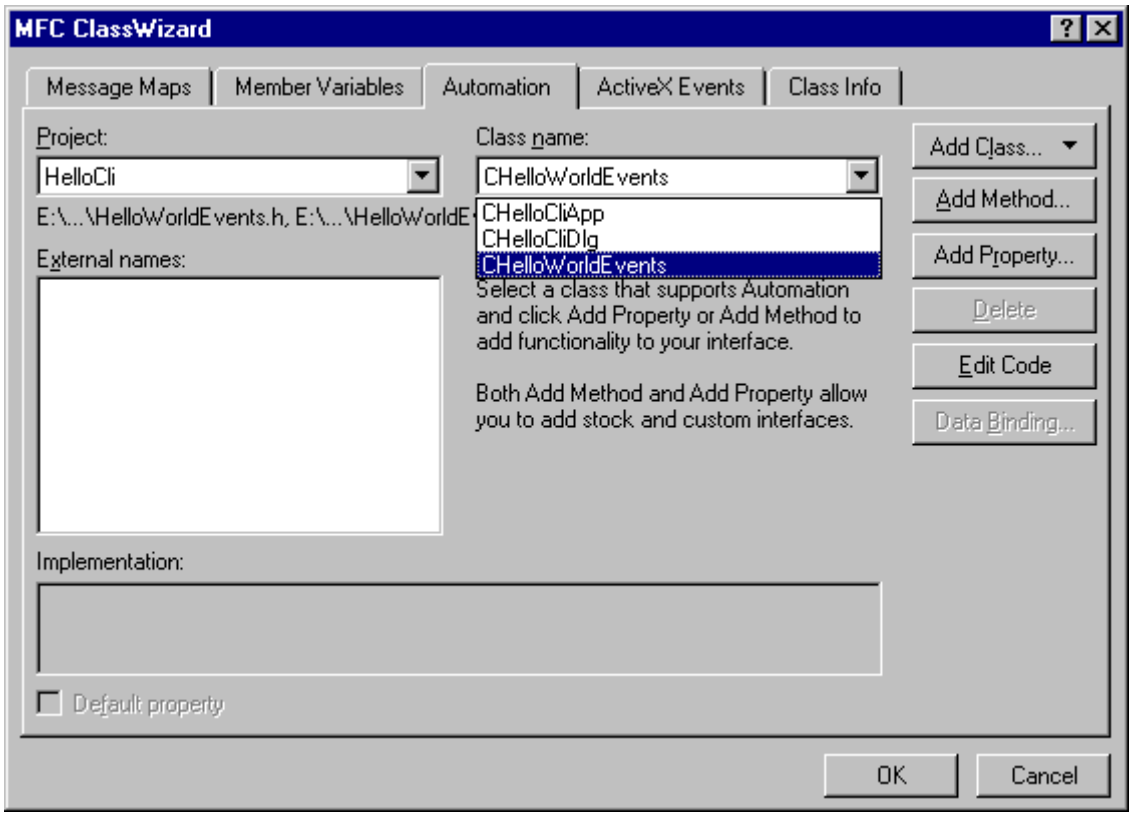


Figure 5. Selecting the CHelloWorldEventsclass on the Automation tab of ClassWizard.

Click Add Method. The Add Method dialog box appears, as shown in **Figure 6** below. Here we're going to specify the "external name" for our event handler method, as well as other information.
The "external name" should ALWAYS match the name of the event method that we used when we added it to the server! The Internal Name box should hold the name of the member function that will get called when the event comes in; this can be whatever you please. We're going to use what ClassWizard suggests; a name that matches the OnSayHello"external name." ALWAYS specify voidfor the return type of an event handler, because the server always uses HRESULTas its return types. For clients, anytime we're handling connection point events, the return type should always be void. Next, specify a parameter, LPCTSTR lpszHostas the event handler's single parameter. You notice that BSTRisn't in the list of event handler types (alright!). This is because you use LPCTSTRinstead; ClassWizard makes sure that MFC will convert between BSTRand LPCTSTR for you (!).

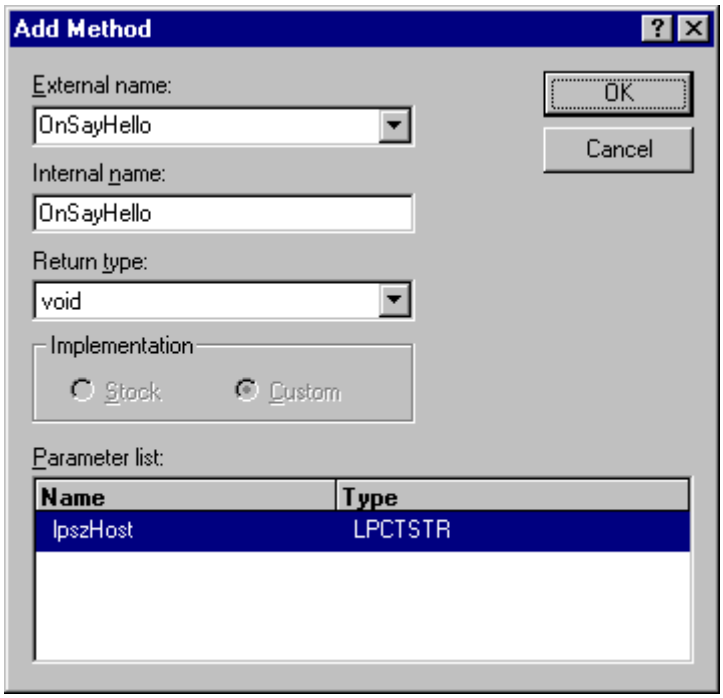


Figure 6. Setting up a handler for the OnSayHelloevent.

Click OK. ClassWizard adds code to CHelloWorldEventsto make the magic happen (with CCmdTarget'shelp), and then shows a new entry in its External Names listbox to show that the event handler has been added:

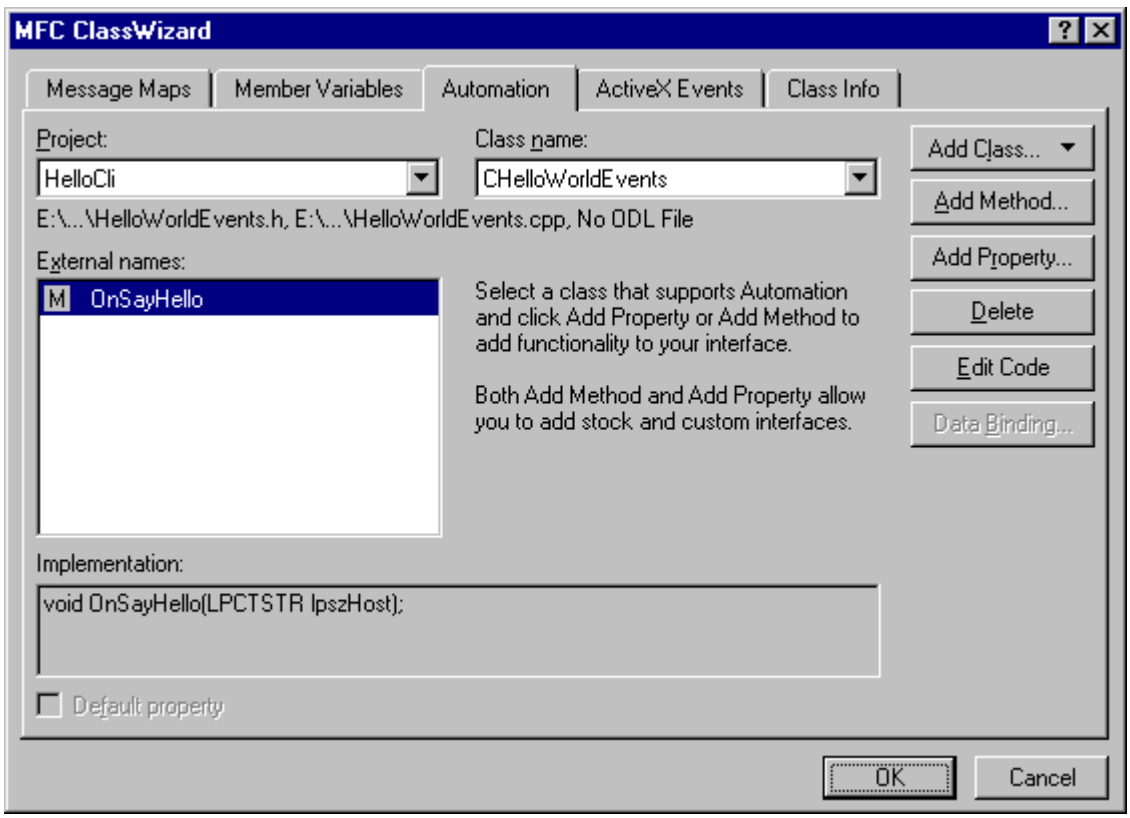


Figure 7. ClassWizard showing the addition of our new event handler.

Save your changes! Make sure and click OK in ClassWizard, otherwise it will roll-back all of its changes that it made. If you click Cancel, you'll have to add the event handler again. Just a word of warning. Now it's time to implement our event handler. We'll grab the name of the server computer from lpszHost, and then we'll show the user a message box saying that the server said Hello. We might also want to add text to the Status window of our dialog saying
that the event handler function got called. Here's how I did that:

Collapse | Copy Code

```
#include "HelloCliDlg.h"
voidCHelloWorldEvents::OnSayHello(LPCTSTR lpszHost)
{
CHelloCliDlg* pDlg = (CHelloCliDlg*)AfxGetMainWnd();

if (pDlg != NULL)
    {
pDlg->m_strStatus
           += "The OnSayHello() connection point method has been called\r\n";
pDlg->UpdateData(FALSE);
    }

// Show a message box saying 'Hello, world, from host ' + lpszHost:
CStringstrMessage = "Hello, world, from ";
strMessage += lpszHost;

AfxMessageBox(strMessage, MB_ICONINFORMATION);
}
```

Listing 7. Implementing the CHelloWorldEvents::OnSayHello()event handler function. I also had to add a friend statement to the declaration of CHelloWorldEvents, because CWnd::UpdateData()is a protectedfunction: Collapse | Copy Code

```
classCHelloWorldEvents : publicCCmdTarget
{
friendclassCHelloCliDlg;

    ...
};
```

The next thing to do is to add some data members to the CHelloCliDlgclass in order to hold the pointers and objects that we'll be using in working with the server. There are quite a few of them, and you'll have to make sure to add the line

Collapse | Copy Code

```
#include "HelloWorldEvents.h"
```

to the top of the HelloCliDlg.hfile: Collapse | Copy Code

```
// Implementation
protected:
    HICON        m_hIcon;

DWORD        m_dwCookie;    // Cookie to keep track of connection point
    BOOL         m_bSinkAdvised;    // Were we able to advise the server?

IHelloWorldPtr* m_pHelloWorld;  // Pointer to the IHelloWorld interface pointer
IUnknown*       m_pHelloWorldEventsUnk; // Pointer to the IUnknown of the
// event "sink"
```

```
CHelloWorldEventsm_events;    // Our event-handler object
```

Listing 8. Data members we need to add to the CHelloCliDlgclass. Next, we need to add code

to the dialog's constructor:

Collapse | Copy Code

```
CHelloCliDlg::CHelloCliDlg(CWnd* pParent/*=NULL*/)
    : CDialog(CHelloCliDlg::IDD, pParent)
{
    ...

m_dwCookie = 0;
m_pHelloWorldEventsUnk = m_events.GetIDispatch(FALSE);
// So we don't have to call Release()
m_bSinkAdvised = FALSE;
}
```

Listing 9. Code to add to the CHelloCliDlg::CHelloCliDlg()constructor function. To advise the server, I

added a AdviseEventSink(), protectedmember function to CHelloCliDlgusing ClassView. This function is implemented the basically the same for anytime you want to advise a server about your MFC connection point:

Collapse | Copy Code

```
BOOL CHelloCliDlg::AdviseEventSink()
{
if (m_bSinkAdvised)
return TRUE;

IUnknown* pUnk = NULL;
CComPtr<IUnknown>spUnk = (*m_pHelloWorld);
pUnk = spUnk.p;

// Advise the connection point
    BOOL bResult = AfxConnectionAdvise(pUnk, DIID_DHelloWorldEvents,
m_pHelloWorldEventsUnk, TRUE, &m_dwCookie);

returnbResult;
}
```

Listing 10. Implementation of advising the server. This demonstrates how to call AfxConnectionAdvise(). You must have properly registered the server like we did in Step 6, or else this won't work.

When we're ready to go back to being aloof to the server and its events that it fires, we can call AfxConnectionUnadvise(): Collapse | Copy

Code

```
BOOL CHelloCliDlg::UnadviseEventSink()
{
if (!m_bSinkAdvised)
```

```
return TRUE;

// Get the IHelloWorldIUnknown pointer using a smart pointer.
// The smart pointer calls QueryInterface() for us.
IUnknown* pUnk = NULL;

CComPtr<IUnknown>spUnk = (*m_pHelloWorld);
pUnk = spUnk.p;

if (spUnk.p)
    {
// Unadvise the connection with the event source
returnAfxConnectionUnadvise(pUnk, DIID_DHelloWorldEvents,
m_pHelloWorldEventsUnk, TRUE, m_dwCookie);
    }

// If we made it here, QueryInterface() didn't work and we can't
// unadvise the server
return FALSE;
}
```

Listing 11. Unadvising the event source and sink with AfxConnectionUnadvise().

To actually make the method call, you can see how I implemented all of this and where my AdviseEventSink()and UnadviseEventSink()play in in the sample program. Remember, though, to add this code to OnInitDialog():

Collapse | Copy Code

```
BOOL CHelloCliDlg::OnInitDialog()
{
CDialog::OnInitDialog();

    ...

CoInitialize(NULL);

CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_NONE,
        RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);

return TRUE;
}
```

Listing 12. Adding intialization code to OnInitDialog().

The OnStartServer()function handles a button the user clicks when they want to start the server (how circular). As you can see, I had a server computer on the network named \\Viz-06 which I connected to with DCOM:

```
voidCHelloCliDlg::OnStartServer()
{
    COSERVERINFO serverInfo;
ZeroMemory(&serverInfo, sizeof(COSERVERINFO));
```

```
        COAUTHINFO athn;
ZeroMemory(&athn, sizeof(COAUTHINFO));

// Set up the NULL security information athn.dwAuthnLevel =
RPC_C_AUTHN_LEVEL_NONE; athn.dwAuthnSvc =
RPC_C_AUTHN_WINNT; athn.dwAuthzSvc =
RPC_C_AUTHZ_NONE; athn.dwCapabilities = EOAC_NONE;
athn.dwImpersonationLevel = RPC_C_IMP_LEVEL_IMPERSONATE; athn.pAuthIdentityData =
NULL;
athn.pwszServerPrincName = NULL;

serverInfo.pwszName = L"\\\\Viz-06"; serverInfo.pAuthInfo = &athn;
        serverInfo.dwReserved1 = 0;
        serverInfo.dwReserved2 = 0;

        MULTI_QI qi = {&IID_IHelloWorld, NULL, S_OK};

...

try
        {
m_pHelloWorld = newIHelloWorldPtr;
        }
catch(...)
        {
AfxMessageBox(AFX_IDP_FAILED_MEMORY_ALLOC, MB_ICONSTOP);

...
return;
        }

        HRESULT hResult = CoCreateInstanceEx(CLSID_HelloWorld, NULL, CLSCTX_LOCAL_SERVER |
                CLSCTX_REMOTE_SERVER, &serverInfo, 1, &qi);

if (FAILED(hResult))
        {
...
return;
        }

m_pHelloWorld->Attach((IHelloWorld*)qi.pItf);

// Now we have a live pointer to the IHelloWorld interface
// on the remote host

...

return;
}
```

Listing 13. How to get an interface pointer to the IHelloWorldinterface on the remote server.

Calling the method is a simple matter of executing this statement:

```
HRESULT hResult = (*m_pHelloWorld)->SayHello();
```

Listing 14. Calling the IHelloWorld::SayHello()method.

To release the server when we're done with it, simply deletethe m_pHelloWorldpointer:

```
deletem_pHelloWorld;
m_pHelloWorld = NULL;
```

Listing 15.

Conclusion : We successfully studied DCOM.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 01	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To analyse the Linux based computer systems using following commands: a. top , b.ps , c. kill, d. cat /proc/cpuinfoe.vmstat Hardware/Software Requirement: Linux Operating System.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 01

Aim : To analyse the Linux based computer systems using following commands: a. top , b.ps, c. kill, d. cat /proc/cpuinfoe.vmstat Hardware/Software Requirement: Linux Operating System.

Theory :

What is a Process?

An instance of a program is called a Process. In simple terms, any command that you give to yourLinux machine starts a new process.

Having multiple processes for the same program
is possible.Types of Processes:

- Foreground Processes: They run on the screen and need input from the user. For example Office Programs
- Background Processes: They run in the background and usually do not need user input. Forexample Antivirus.

Click here if the video is not accessible

Running a Foreground Process

To start a foreground process, you can either run it from the dashboard, or you can run it from theterminal.

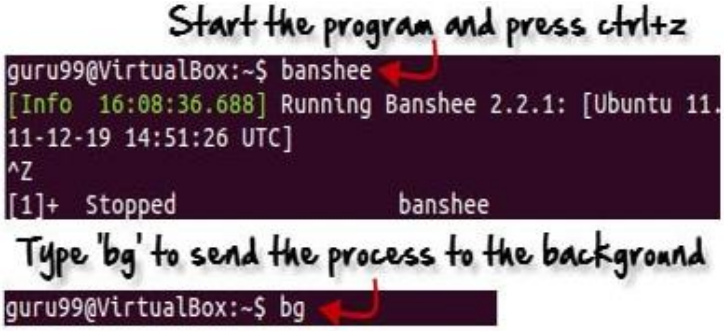
When using the Terminal, you will have to wait, until the foreground process runs.

OR

```
home@VirtualBox:~$ banshee
```

Running a Background process

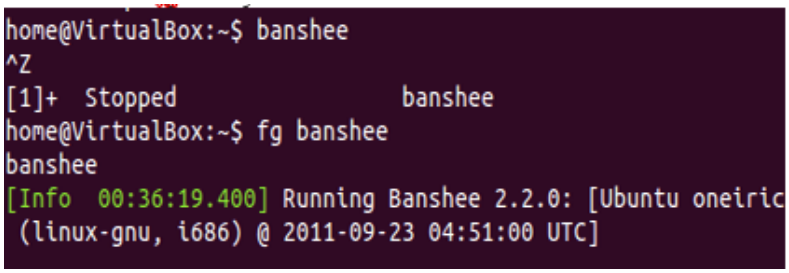
If you start a foreground program/process from the terminal, then you cannot work on the terminal,till the program is up and running.
Particular, data-intensive tasks take lots of processing power and may even take hours to complete.You do not want your terminal to be held up for such a long time.
To avoid such a situation, you can run the program and send it to the background so that terminalremains available to you. Let's learn how to do this –



Fg

You can use the command "fg" to continue a program which was stopped and bring it to the foreground.
The simple syntax for this utility is:
fg jobnameExample

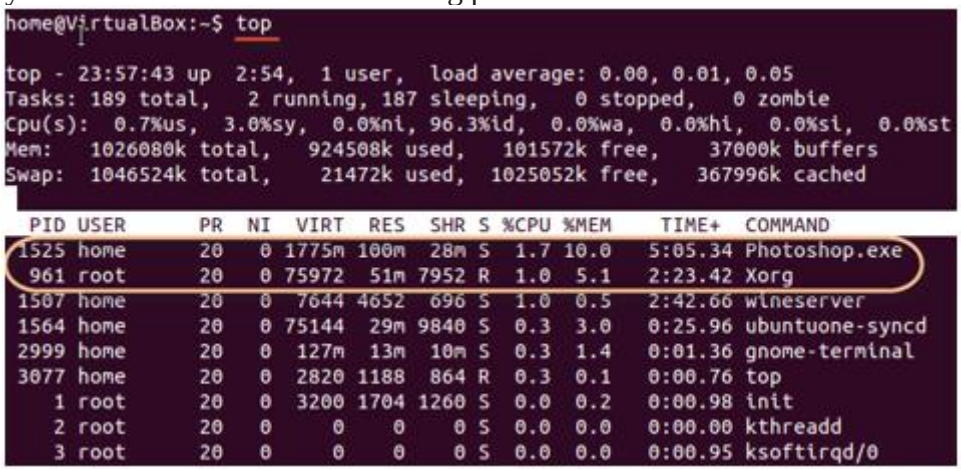
1. Launch 'banshee' music player
2. Stop it with the 'ctrl +z' command
3. Continue it with the 'fg' utility.



Let's look at other important commands to manage processes –

Top

This utility tells the user about all the running processes on the Linux machine



Press 'q' on the keyboard to move out of the process display.

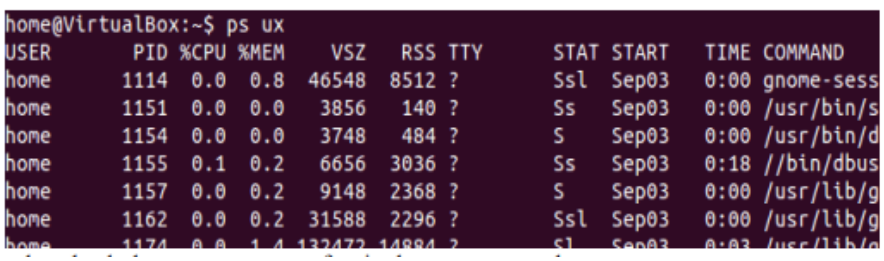
The terminology follows:

Field	Description	Example 1	Example 2
PID	The process ID of each task	1525	961
User	The username of task owner	Home	Root
PR	Priority Can be 20(highest) or -20(lowest)	20	20
NI	The nice value of a task	0	0
VIRT	Virtual memory used (kb)	1775	75972
RES	Physical memory used (kb)	100	51
SHR	Shared memory used (kb)	28	7952
S	Status There are five types: 'D' = uninterruptible sleep 'R' = running 'S' = sleeping 'T' = traced or stopped 'Z' = zombie	S	R
%CPU	% of CPU time	1.7	1.0
%MEM	Physical memory used	10	5.1
TIME+	Total CPU time	5:05.34	2:23.42
Command	Command name	Photoshop.exe	Xorg

PS

This command stands for 'Process Status'. It is similar to the "Task Manager" that pop-ups in a Windows Machine when we use Cntrl+Alt+Del. This command is similar to 'top' command but the information displayed is different. To check all the processes running under a user, use the command –

ps ux



ps PID

```
guru99@VirtualBox:~$ ps 1268
  PID TTY      STAT   TIME COMMAND
 1268 ?        S<l    0:02 /usr/bin/pulseaudio --start --log-target=syslog
```

Kill

This command **terminates running processes** on a Linux machine.

To use these utilities you need to know the PID (process id) of the process you want to kill

Syntax –

kill PID

To find the PID of a process simply type

pidof Process name

Let us try it with an example.

```
home@VirtualBox:~$ pidof Photoshop.exe
1525
home@VirtualBox:~$ kill 1525
```

NICE

Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.

To avoid this, you can tell your machine to prioritize processes as per your requirements.

This priority is called Niceness in Linux, and it has a value between -20 to 19. The lower the Niceness index, the higher would be a priority given to that task.

The default value of all the processes is 0.

To start a process with a niceness value other than the default value use the following syntax

nice -n 'Nice value' process name

```
home@VirtualBox:~$ nice -n 19 banshee
```

If there is some process already running on the system, then you can 'Renice' its value using syntax.

renice 'nice value' -p 'PID'

To change Niceness, you can use the 'top' command to determine the PID (process id) and its Nice value. Later use the renice command to change the value.

Let us understand this by an example.

Checking the niceness value of the process 'banshee'

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3293 home      20   0  277m  64m  35m S 96.4  6.4   9:56.72 banshee
```

Renicing the value to -20

```
home@VirtualBox:~$ sudo renice -20 -p 3293
[sudo] password for home:
3293 (process ID) old priority 0, new priority -20
```

The value changed to -20

```
 3293 home      0 -20  277m  64m  35m S 95.2  6.4   3:32.95 banshee
```

DF

This utility reports the free disk space(Hard Disk) on all the file systems.

```
guru99@guru99-VirtualBox:~$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda1        7837756  2921376   4523216  40% /
udev              246488        4    246484   1% /dev
tmpfs             101512      752    100760   1% /run
none                5120        0      5120   0% /run/lock
none              253776       76    253700   1% /run/shm
```

If you want the above information in a readable format, then use the command 'df -h'

```
guru99@guru99-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       7.5G  2.8G  4.4G  40% /
udev            241M  4.0K  241M   1% /dev
tmpfs           100M  752K   99M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            248M   76K  248M   1% /run/shm
```

Free

This command shows the free and used memory (RAM) on the Linux system.

```
home@VirtualBox:~$ free
             total       used       free     shared    buffers     cached
Mem:       1026080     803604     222476          0      36312     343376
-/+ buffers/cache:     423916     602164
Swap:      1046524      35832    1010692
```

You can use the arguments
free -m to display output in MB
free -g to display output in GB

Summary:

- Any running program or a command given to a Linux system is called a process
- A process could run in foreground or background
- The priority index of a process is called Nice in Linux. Its default value is 0, and it can vary between 20 to -19
- The lower the Niceness index, the higher would be priority given to that task

Command	Description
bg	To send a process to the background
fg	To run a stopped process in the foreground
top	Details on all Active Processes
ps	Give the status of processes running for a user
ps PID	Gives the status of a particular process
pidof	Gives the Process ID (PID) of a process
kill PID	Kills a process
nice	Starts a process with a given priority
renice	Changes priority of an already running process
df	Gives free hard disk space on your system
free	Gives free RAM on your system

Conclusion : We have successfully analyse the Linux based computer systems using following commands: a. top , b.ps , c. kill, d. cat /proc/cpuinfoe.vmstat

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 02	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To setup SSH passwordless logins for two or more Linux based machines and execute commands on a remote machine. Hardware/Software Requirement.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 02

Aim : To setup SSH passwordless logins for two or more Linux based machines and execute commands on a remote machine. Hardware/Software Requirement.

Theory :

SSH (Secure Shell) allows secure remote connections between two systems. With this cryptographic protocol, you can manage machines, copy, or move files on a remote server via encrypted channels.

There are two ways to login onto a remote system over SSH – using **password authentication** or **public key authentication** (passwordless SSH login).

In this tutorial, you will find out how to set up and enable passwordless SSH login.

Prerequisites

- Access to command line/terminal window
- User with **sudo** or **root** privileges
- A local server and a remote server
- **SSH access** to a remote server via command line/terminal window

Before You Start: Check for Existing SSH Keys

You may already have an SSH key pair generated on your machine. To see whether you have SSH keys on the system, run the command:

```
ls -al ~/.ssh/id_*.pub
```

If the output tells you there are no such files, move on to the next step, which shows you how to generate SSH keys.

In case you do have them, you can use the existing keys, back them up and create a new pair or overwrite it.

Step 1: Generate SSH Key Pair

1. The first thing you need to do is **generate an SSH key pair** on the machine you are currently working on.

In this example, we generate a 4096-bit key pair. We also add an email address, however this is optional. The command is:

```
ssh-keygen -t rsa -b 4096 -C "your_email@domain.com"
```

```
sofija@sofija-VirtualBox:~$ ssh-keygen -t rsa -b 4096 -C "sofija@phoenixnap.com
"
Generating public/private rsa key pair.
```

2. Next, type in the location where you want to store the keys or hit **Enter** to accept the default path.

3. It also asks you to set a passphrase. Although this makes the connection even more secure, it may interrupt when setting up automated processes. Therefore, you can type in a passphrase or just press **Enter** to skip this step.

```
Enter file in which to save the key (/home/sofija/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
```

4. The output then tells you where it stored the identification and public key and gives you the key fingerprint.

```
Your identification has been saved in /home/sofija/.ssh/id_rsa.
Your public key has been saved in /home/sofija/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:mEDEJEmq9Ls4AC2kvoynknDywGKR0pDsHkoYx2oM0Pg sofija@phoenixnap.com
The key's randomart image is:
+---[RSA 4096]----+
|+*o=+            |
|*+=o.            |
|OX. .            |
|XOE  . o         |
|@.o.  o S        |
|B=. .            |
|B*..             |
|+++ .            |
|o+..             |
+----[SHA256]-----+
```

5. Verify you have successfully created the SSH key pair by running the command:

```
ls -al ~/.ssh/id_*.pub
```

You should see the path of the identification key and the public key, as in the image below:

```
sofija@sofija-VirtualBox:~$ ls ~/.ssh/id *
/home/sofija/.ssh/id_rsa  /home/sofija/.ssh/id_rsa.pub
sofija@sofija-VirtualBox:~$
```

Step 2: Upload Public Key to Remote Server

You can upload the public SSH key to a remote server with the **ssh-copy-id** command or the cat command. Below you can find both options.

Option 1: Upload Public Key Using the ssh-copy-id Command

To enable passwordless access, you need to upload a copy of the public key to the remote server.

1. Connect to the remote server and use the **ssh-copy-id** command:

```
ssh-copy-ide [remote_username]@[server_ip_address]
```

2. The public key is then automatically copied into the **.ssh/authorized_keys** file.

Option 2: Upload Public Key Using the cat Command

Another way to copy the public key to the server is by using the **cat** command.

1. Start by connecting to the server and creating a **.ssh** directory on it.

```
ssh [remote_username]@[server_ip_address] mkdir -p .ssh
```

2. Then, type in the password for the remote user.

3. Now you can upload the public key from the local machine to the remote server. The command also specifies that the key will be stored under the name ***authorized_keys*** in the newly created **.ssh** directory:

```
cat .ssh/id_rsa.pub | ssh [remote_username]@[server_ip_address] 'cat >> .ssh/authorized_keys'
```

Step 3: Log in to Server Without Password

With the SSH key pair generated and the public key uploaded to the remote server, you should now be able to connect to your dedicated server without providing a password.

Check whether the setup works by running the command:

```
ssh [remote_username]@[server_ip_address]
```

The system should directly log you in to the remote server, no password required.

Note: Once you verify that you can SHH into the remote serve without a password, consider disabling SSH password authentication altogether. It will add another layer of security and secure your server from brute force attacks.

Optional: Troubleshooting Remote Server File Permissions

File permissions on the remote server may cause issues with passwordless SSH login. This is a common issue with older versions of SSH.

If you are still prompted for a password after going through all the steps, start by editing file permissions on the remote server.

- Set permissions 700 for the **.ssh** directory.
- Set permissions 640 for the **.ssh/authorized_keys** directory.

Edit file permissions with the following command:

```
ssh [remote_username]@[server_ip_address] "chmod 700 .ssh; chmod 640 .ssh/authorized_keys"
```

Enter your password when prompted. There will be no output if the action was successful. The issue should be resolved now.

Conclusion : We successfully studied To setup SSH passwordless logins for two or more Linux based machines and execute commands on a remote machine.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 03	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Write a program in C to multiply two matrices of size 10000 x 10000 each and find it's execution-time using "time" command.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 03

Aim : Write a program in C to multiply two matrices of size 10000 x 10000 each and find it's execution-time using "time" command. Try to run this program on two or more machines having different configurations and compare execution-times obtained in each run. Comment on which factors affect the performance of the program.

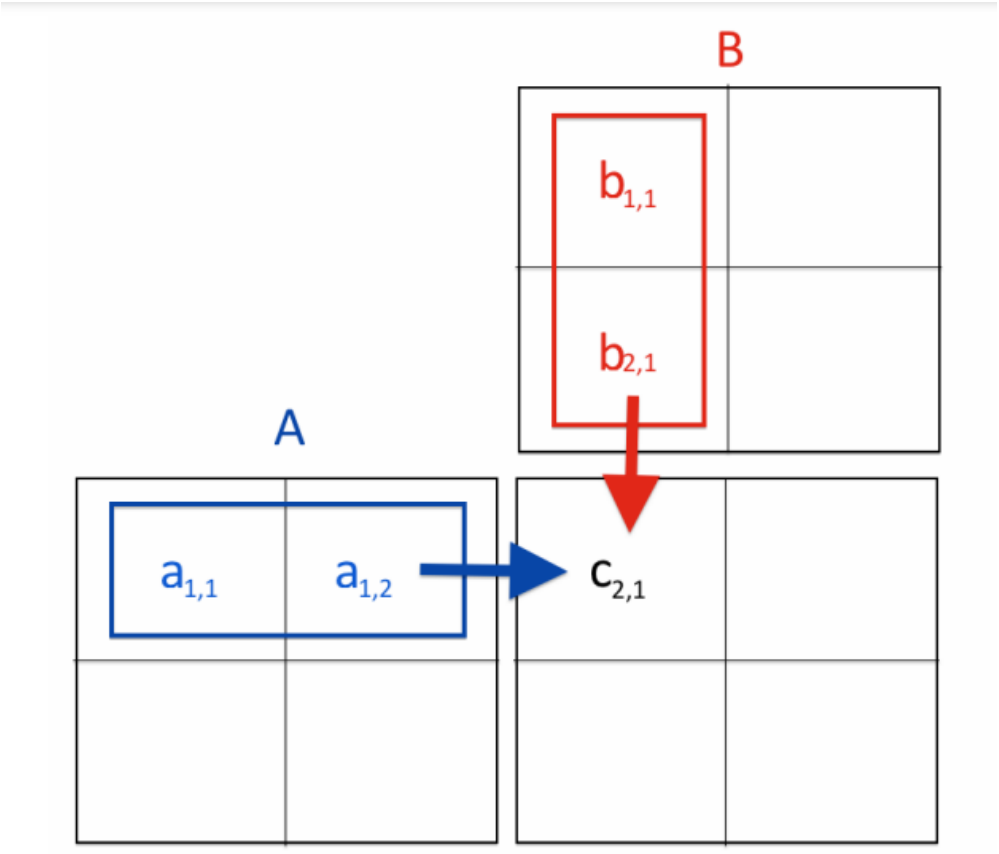
Theory :

Matrix-Matrix Multiplication

Before starting, it is helpful to briefly recap how a matrix-matrix multiplication is computed. Let's say we have two matrices, A and B. Assume that A is a n×m matrix, which means that it has n rows and m columns. Also assume that B is a m×w matrix. The result of the multiplication A∗B (which is different from B∗A!) is a n×w matrix, which we call M. That is, the number of rows in the resulting matrix equals the number of rows of the first matrix A and the number of columns of the second matrix B.

Why does this happen and how does it work? The answer is the same for both questions here. Let's take the cell 1,1 (first row, first column) of M. The number inside it after the operation M=A∗B is the sum of all the element-wise multiplications of the numbers in A, row 1, with the numbers in B, column 1. That is, in the cell i,j of M we have the sum of the element-wise multiplication of all the numbers in the i-th row in A and the j-th column in B.

The following figure intuitively explains this idea:



It should be pretty clear now why matrix-matrix multiplication is a good example for parallel computation. We have to compute every element in C, and each of them is independent from the others, so we can efficiently parallelize.

We will see different ways of achieving this. The goal is to add new concepts throughout this article, ending up with a 2D kernel, which uses shared memory to efficiently optimize operations.

Grids And Blocks

After the previous articles, we now have a basic knowledge of CUDA thread organisation, so that we can better examine the structure of grids and blocks.
When we call a kernel using the instruction <<< >>> we automatically define a dim3 type variable defining the number of blocks per grid and threads per block. In fact, grids and blocks are 3D arrays of blocks and threads, respectively.
This is evident when we define them before calling a kernel, with something like this:

```
dim3 blocksPerGrid(512, 1, 1)
dim3 threadsPerBlock(512, 1, 1)
kernel<<<blocksPerGrid, threadsPerBlock>>>()
```

In the previous articles you didn't see anything like that, as we only discussed 1D examples, in which we didn't have to specify the other dimensions. This is because, if you only give a number to the kernel call as we did, it is assumed that you created a dim3 mono-dimensional variable, implying y=1 and z=1.

As we are dealing with matrices now, we want to specify a second dimension (and, again, we can omit the third one). This is very useful, and sometimes essential, to make the threads work properly.

Indeed, in this way we can refer to both the x and y axis in the very same way we followed in previous examples. Let's have a look at the code:

```
int row = blockIdx.y * blockDim.y + threadIdx.y;
int col = blockIdx.x * blockDim.x + threadIdx.x;
C++
```

As you can see, it's similar code for both of them. In CUDA, blockIdx, blockDim and threadIdx are built-in functions with members x, y and z. They are indexed as normal vectors in C++, so between 0 and the maximum number minus 1. For instance, if we have a grid dimension of blocksPerGrid = (512, 1, 1), blockIdx.x will range between 0 and 511.

As I mentioned here the total amount of threads in a single block cannot exceed 1024.

Using a multi-dimensional block means that you have to be careful about distributing this number of threads among all the dimensions. In a 1D block, you can set 1024 threads at most in the x axis, but in a 2D block, if you set 2 as the size of y, you cannot exceed 512 for the x! For example, dim3 threadsPerBlock(1024, 1, 1) is allowed, as well as dim3 threadsPerBlock(512, 2, 1), but not dim3 threadsPerBlock(256, 3, 2).

Linearise Multidimensional Arrays

In this article we will make use of 1D arrays for our matrixes. This might sound a bit confusing, but the problem is in the programming language itself. The standard upon which CUDA is developed needs to know the number of columns before compiling the program. Hence it is impossible to change it or set it in the middle of the code.

However, a little thought shows that this is not a big issue. We cannot use the comfortable notation A[i][j], but we won't struggle that much as we already know how to properly index rows and columns.

In fact, the easiest way to linearize a 2D array is to stack each row lengthways, from the first to the last. The following picture will make this concept clearer:

The Kernel

Now that we have all the necessary information for carrying out the task, let's have a look at the kernel code. For sake of simplicity we will use square N×N matrices in our example.

The first thing to do, as we already saw, is to determine the x and y axis index (i.e. row and column numbers):

```
__global__ void multiplication(float *A, float* B, float *C, int N){
  int ROW = blockIdx.y*blockDim.y+threadIdx.y;
  int COL = blockIdx.x*blockDim.x+threadIdx.x;
```

C++

Copy

Then, we check that the row and column total does not exceed the number of actual rows and columns in the matrices. As the threads will access the memory in random order, we have to do this for preventing unnecessary threads from performing operations on our matrices. That is, we are making blocks and grids of a certain size, so that if we don't set our N as a multiple of that size, we would have more threads than we need:

```
  if (ROW < N && COL < N) {
```

C++

Copy

We won't have any problem here as we are using square matrices, but it's always a good idea to keep this in mind. We then have to initialise the temporary variable tmp_sum for summing

all the cells in the selected row and column. It is always a good procedure to specify the decimal points and the f even if they are zeroes. So let's write:

```
    float tmp_sum = 0.0f;
```

C++

Copy

Now the straightforward part: as for the CPU code, we can use a for loop for computing the sum and then store it in the corresponding C cell.

```
        for (int i = 0; i < N; i++) {
            tmpSum += A[ROW * N + i] * B[i * N + COL];
        }
    C[ROW * N + COL] = tmpSum;
```

C++

Copy

The host (CPU) code starts by declaring variables in this way:

```
int main() {
    // declare arrays and variables
    int N = 16;
    int SIZE = N*N;
    vector<float> h_A(SIZE);
    vector<float> h_B(SIZE);
    vector<float> h_C(SIZE);
```

C++

Copy

Note that, given the if condition in the kernel, we could set N values that are not necessarily a multiple of the block size. Also, I will make use of the library dev_array, which I discussed in this article.

Now let's fill the matrices. There are several ways to do this, such as making functions for manual input or using random numbers. In this case, we simply use a for loop to fill the cells with trigonometric values of the indices:

```
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            h_A[i*N+j] = sin(i);
            h_B[i*N+j] = cos(j);
        }
    }
```

C++

Copy

Recalling the "Grids and Blocks" paragraph, we now have to set the dim3 variables for both blocks and grids dimensions. The grid is just a BLOCK_SIZE × BLOCK_SIZE grid, so we can write:

```
dim3 threadsPerBlock (BLOCK_SIZE, BLOCK_SIZE)
```

C++

Copy

As we are not working only with matrices with a size multiple of BLOCK_SIZE, we have to use the ceil instruction, to get the next integer number as our size, as you can see:

```
int n_blocks = ceil(N/BLOCK_SIZE);
dim3 blocksPerGrid (n_blocks, n_blocks)
```

C++

Copy

Note that in this way we will use more threads than necessary, but we can prevent them working on our matrices with the if condition we wrote in the kernel.

This is the general solution showing the reasoning behind it all, but in the complete code you will find a more efficient version of it in kernel.cu.

Now we only have to create the device arrays, allocate memory on the device and call our kernel and, a as result, we will have a parallel matrix multiplication program. Using dev_array, we simply write:

```
    dev_array<float> d_A(SIZE);
    dev_array<float> d_B(SIZE);
    dev_array<float> d_C(SIZE);
```

C++

Copy

for declaring arrays and:

```
    d_A.set(&h_A[0], SIZE);
    d_B.set(&h_B[0], SIZE);
```

C++

Copy

for allocating memory. For getting the device results and copying it on the host, we use the get method instead. Once again, this is simply:

```
d_C.get(&h_C[0], SIZE);
```

C++

Copy

At the bottom of this page you can find the complete code, including performance comparison and error computation between the parallel and the serial code.

In the next article I will discuss the different types of memory and, in particular, I will use the shared memory for speeding-up the matrix multiplication. But don't worry, just after this we will come back to an actual finance application by applying what we learned so far to financial problems.

Full Code

dev_array.h

```
#ifndef _DEV_ARRAY_H_
#define _DEV_ARRAY_H_

#include <stdexcept>
#include <algorithm>
#include <cuda_runtime.h>

template <class T>
class dev_array
{
// public functions
public:
    explicit dev_array()
        : start_(0),
          end_(0)
    {}

    // constructor
```

```
    explicit dev_array(size_t size)
    {
        allocate(size);
    }
    // destructor
    ~dev_array()
    {
        free();
    }

    // resize the vector
    void resize(size_t size)
    {
        free();
        allocate(size);
    }

    // get the size of the array
    size_t getSize() const
    {
        return end_ - start_;
    }

    // get data
    const T* getData() const
    {
        return start_;
    }
```

```
    T* getData()
    {
        return start_;
    }

    // set
    void set(const T* src, size_t size)
    {
        size_t min = std::min(size, getSize());
        cudaError_t result = cudaMemcpy(start_, src, min *
sizeof(T), cudaMemcpyHostToDevice);
        if (result != cudaSuccess)
        {
            throw std::runtime_error("failed to copy to device
memory");
        }
    }
    // get
    void get(T* dest, size_t size)
    {
```

```
        size_t min = std::min(size, getSize());
        cudaError_t result = cudaMemcpy(dest, start_, min *
sizeof(T), cudaMemcpyDeviceToHost);
        if (result != cudaSuccess)
        {
            throw std::runtime_error("failed to copy to host
memory");
        }
    }


// private functions
private:
    // allocate memory on the device
    void allocate(size_t size)
    {
        cudaError_t result = cudaMalloc((void**)&start_, size *
sizeof(T));
        if (result != cudaSuccess)
        {
            start_ = end_ = 0;
            throw std::runtime_error("failed to allocate device
memory");
        }
        end_ = start_ + size;
    }

    // free memory on the device
    void free()
    {
        if (start_ != 0)
        {
            cudaFree(start_);
            start_ = end_ = 0;
        }
    }

    T* start_;
    T* end_;
};

#endif
```

C++

Copy

matrixmul.cu

```
#include <iostream>
```

```cpp
#include <vector>
#include <stdlib.h>
#include <time.h>
#include <cuda_runtime.h>
#include "kernel.h"
#include "kernel.cu"
#include "dev_array.h"
#include <math.h>

using namespace std;

int main()
{
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    int N = 16;
    int SIZE = N*N;

    // Allocate memory on the host
    vector<float> h_A(SIZE);
    vector<float> h_B(SIZE);
    vector<float> h_C(SIZE);

    // Initialize matrices on the host
    for (int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            h_A[i*N+j] = sin(i);
            h_B[i*N+j] = cos(j);
        }
    }

    // Allocate memory on the device
    dev_array<float> d_A(SIZE);
    dev_array<float> d_B(SIZE);
    dev_array<float> d_C(SIZE);

    d_A.set(&h_A[0], SIZE);
    d_B.set(&h_B[0], SIZE);

    matrixMultiplication(d_A.getData(), d_B.getData(),
d_C.getData(), N);
    cudaDeviceSynchronize();

    d_C.get(&h_C[0], SIZE);
    cudaDeviceSynchronize();

    float *cpu_C;
    cpu_C=new float[SIZE];
```

```
    // Now do the matrix multiplication on the CPU
    float sum;
    for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){
            sum = 0.f;
            for (int n=0; n<N; n++){
                sum += h_A[row*N+n]*h_B[n*N+col];
            }
            cpu_C[row*N+col] = sum;
        }
    }

    double err = 0;
    // Check the result and make sure it is correct
    for (int ROW=0; ROW < N; ROW++){
        for (int COL=0; COL < N; COL++){
            err += cpu_C[ROW * N + COL] - h_C[ROW * N + COL];
        }
    }

    cout << "Error: " << err << endl;

    return 0;
}
```

C++

Copy

kernel.h

```
#ifndef KERNEL_CUH_
#define KERNEL_CUH_

void matrixMultiplication(float *A, float *B, float *C, int N);

#endif
```

C++

Copy

kernel.cu

```
#include <math.h>
#include <iostream>
#include "cuda_runtime.h"
#include "kernel.h"
#include <stdlib.h>

using namespace std;
```

```
__global__ void matrixMultiplicationKernel(float* A, float* B,
float* C, int N) {

    int ROW = blockIdx.y*blockDim.y+threadIdx.y;
    int COL = blockIdx.x*blockDim.x+threadIdx.x;

    float tmpSum = 0;

    if (ROW < N && COL < N) {
        // each thread computes one element of the block sub-matrix
        for (int i = 0; i < N; i++) {
            tmpSum += A[ROW * N + i] * B[i * N + COL];
        }
    }
    C[ROW * N + COL] = tmpSum;
}


void matrixMultiplication(float *A, float *B, float *C, int N){

    // declare the number of blocks per grid and the number of
threads per block
    // use 1 to 512 threads per block
    dim3 threadsPerBlock(N, N);
    dim3 blocksPerGrid(1, 1);
        if (N*N > 512){
            threadsPerBlock.x = 512;
            threadsPerBlock.y = 512;
            blocksPerGrid.x =
ceil(double(N)/double(threadsPerBlock.x));
            blocksPerGrid.y =
ceil(double(N)/double(threadsPerBlock.y));
        }

    matrixMultiplicationKernel<<<blocksPerGrid,threadsPerBlock>>>(A,
B, C, N);
}
```

Conclusion : We have successfully implemented program in C to multiply two matrices of size 10000 x 10000 each and find it's execution-time using "time" command.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 04	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Write a "Hello World" program using OpenMP library also display number of threads created during execution.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 04

Aim : Write a "Hello World" program using OpenMP library also display number of threads created during execution.

Theory :

In this section we will learn how to make a simple parallel hello world program in C++. Let's begin with the creation of a program titled: parallel_hello_world.cpp. From the command line run the command:
nano parallel_hello_world.cpp

We will begin with include statements we want running at the top of the program:

```
#include <omp.h>
```

These flags allow us to utilize the stdio and omp libraries in our program. The `<omp.h>` header file will provide openmp functionality. The `<stdio.h>` header file will provide us with print functionality.

Let's now begin our program by constructing the main function of the program. We will use `omp_get_thread_num()` to obtain the thread id of the process. This will let us identify each of our threads using that unique id number.

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
    printf("Hello from process: %d\n", omp_get_thread_num());

    return 0;
}
```

Let's compile our code and see what happens. We must first load the compiler module we want into our environment. We can do so as such:

GCC:

```
module load gcc
```

Or

Intel:

```
module load intel
```

From the command line, where your code is located, run the command:

GCC:

```
g++ parallel_hello_world.cpp -o parallel_hello_world.exe -fopenmp
```

Or

Intel:

```
icc parallel_hello_world.cpp -o parallel_hello_world.exe -qopenmp
```

This will give us an executable we can submit as a job to our cluster. Simply submit the job to Slurm, running the executable. Your job script should look something like this:

```
#!/bin/bash
```

```
#SBATCH --nodes=1
#SBATCH --time=0:01:00
#SBATCH --partition=atesting
#SBATCH --constraint=ib
#SBATCH --ntasks=4
#SBATCH --job-name=CPP_Hello_World
#SBATCH --output=CPP_Hello_World.out

./parallel_hello_world.exe
```

Our output file should look like this:

```
Hello from process: 0
```

As you may have noticed, we only get one thread giving us a Hello statement. How do we parallelize the print statement? We parallelize it with `pragma` ! The `#pragma omp parallel { … }` directive creates a section of code that will be run in parallel by multiple threads. Let's implement it in our code:

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
    #pragma omp parallel
    {
        printf("Hello from process: %d\n", omp_get_thread_num());
    }
    return 0;
}
```

We must do one more thing before achieving parallelization. To set the amount of threads we want OpenMP to run on, we must set an Linux environment variable to be specify how many threads we wish to use. The environment variable: `OMP_NUM_THREADS` will store this information. Changing this variable does not require recompilation of the the program, so this command can be placed in either the command line or on your job script:

```
export OMP_NUM_THREADS=4
```

Important to note: this environment variable will need to be set every time you exit your shell. If you would like to make this change permanent you will need to add these lines to your `.bash_profile` file in your home directory:

```
OMP_NUM_THREADS=4;
export OMP_NUM_THREADS
```

Now let's re-compile the code and run it to see what happens:

GCC

```
g++ parallel_hello_world.cpp -o parrallel_hello_world.exe -fopenmp
```

Or

Intel

```
icc parallel_hello_world.cpp -o parrallel_hello_world.exe -qopenmp
```

Running our job script and we should end with an output file similar to this one:

```
Hello from process: 3
Hello from process: 0
Hello from process: 2
Hello from process: 1
```

Conclusion : We have successfully written a "Hello World" program using OpenMP library also display number of threads created during execution.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 05	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Write a parallel program to calculate the value of PI/Area of Circle using OpenMP library.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

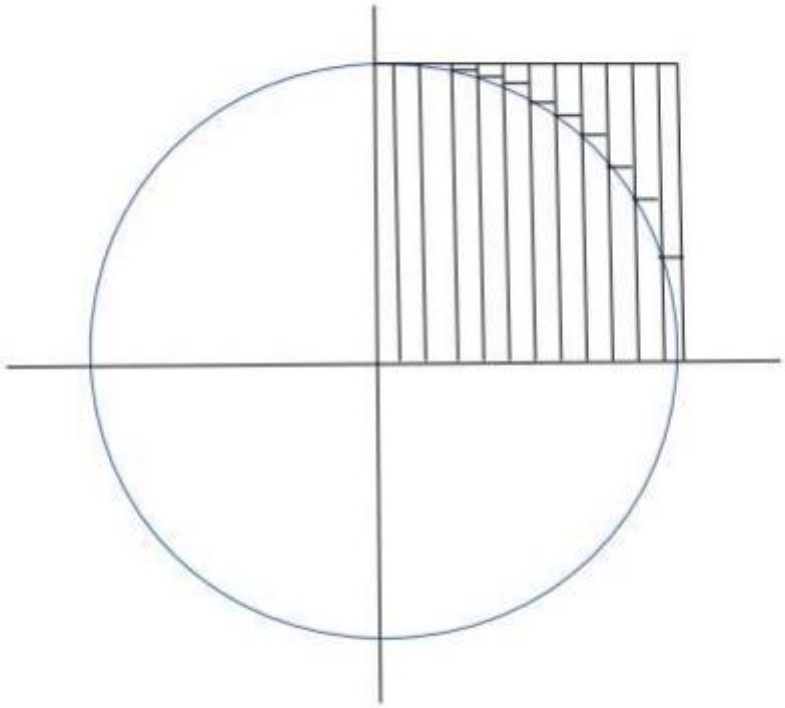
Faculty Signature with Date:-

Experiment No. 05

Aim : Write a parallel program to calculate the value of PI/Area of Circle using OpenMP library.

Theory :

A darts method for approximating π uniformly generates the points in the square . Then it counts the points which lie in the inside of the unit circle.

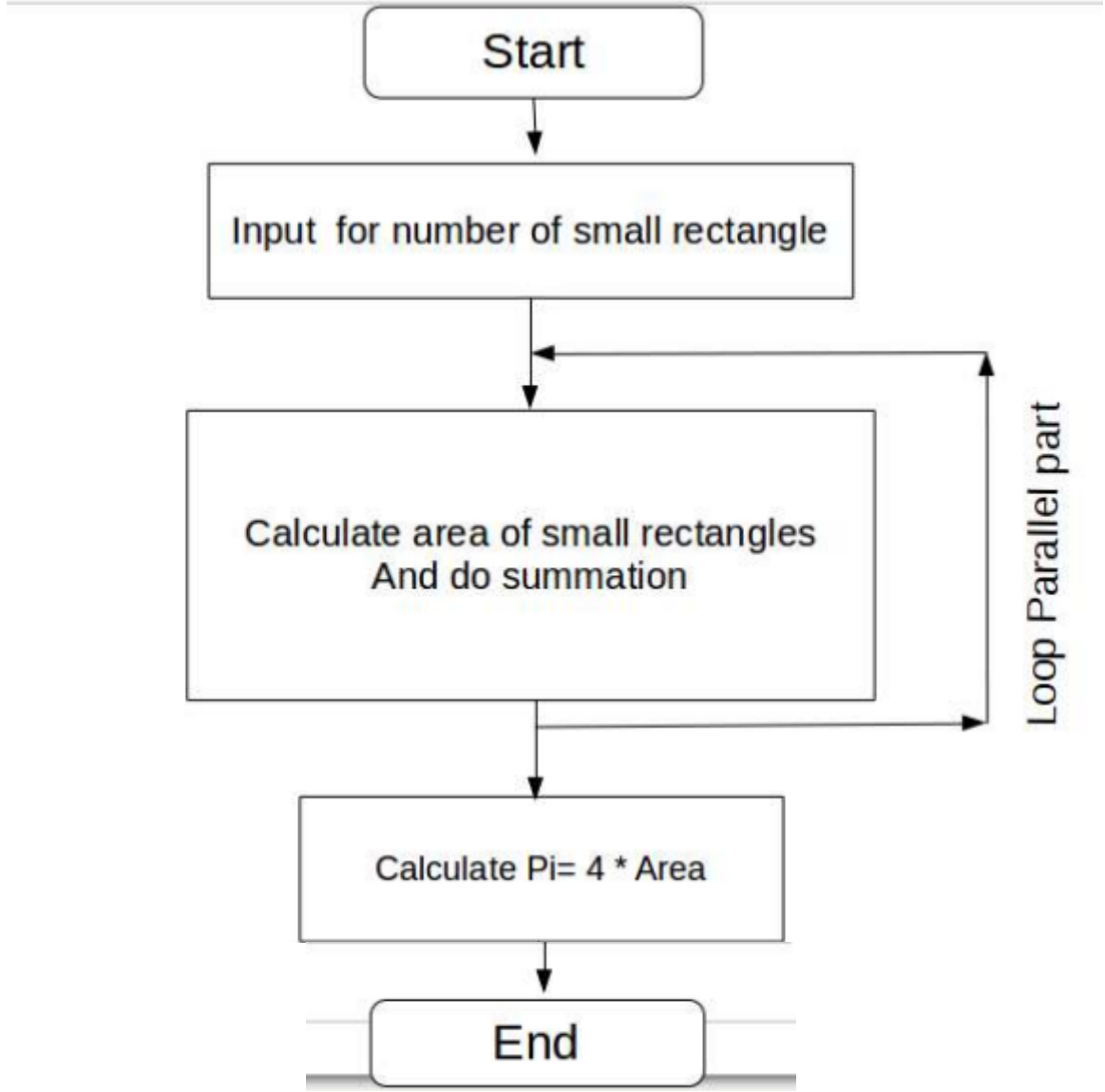


An approximation of π is then computed by the following formula:
We divide the circle in 4 equal part
consider upper right corner
consider radius of circle r = 1
Now area of circle A = π(r)2 = π
one part area(B) = A/4= π/4
so calculate total area = 4* B= π
we can approximately calculate B by summation of area of small rectangles
area of rectangle = width * height
width = 1/(no. Of small rectangle)
height we can calculate using this formula x2 + y2 = r2
height = y= sqrt(1 - x2)

```
pragma omp parallel for private(x, i, height) reduction(+:area)
    for (i = 0; i < NumRectangles; i++) {
/*printf("My thread ID is %d\n", omp_get_thread_num());*/
        /* We want to calculate the x-coordinate where the rectangle intersects the
          * circle by counting how many rectangle widths the rectangle is away from
          * the origin */
        x = i * width;

        /* We want to use the x^2 + y^2 = r^2 equation to calculate the height of
          * the rectangle, which is where the rectangle intersects the circle (y).
         * Because the circle is a unit circle, r = 1. */
       height = sqrt(1.0 - x * x);

       /* We want to compute the area of the rectangle and add it to our running
         * total */
       area += width * height;

   }
```

The above loop will be get parallelized with reduction in area

Flowchart:



Program :

```
/* Compute pi by calculating the sum of areas under a quarter unit circle of
* rectangles whose top-left corners intersect the circle.
*
* To compile: gcc -Wall -o pi pi.c
*
* To run: ./pi
*
* Authors:
  * Aaron Weeden, Shodor Education Foundation, 2015
  */

  /* We want to use the printf function to display the final result */
  #include <stdio.h>

  /* We want to use the sqrt function to calculate square roots */
  #include <math.h>
  #include <omp.h>

  /* We want to make the number of rectangles constant for this run of the
  * program, and we want to declare it here so it is easier to find and change
  * later */
  const int NumRectangles = 1000000;

  /* Every C program starts with main() */
  int main() {

  /* We want a variable for the width of each rectangle. The total width is 1
  * because we are working with a quarter unit circle. */
  float width = 1.0 / NumRectangles;

  /* We want a variable for the x-coordinate at which each rectangle intersects
  * the unit circle */
  float x;

  /* We want a variable for the height of each rectangle */
  float height;
/* We want to loop over each of the rectangles and run the same code for
* each */
#pragma omp parallel for private(x, i, height) reduction(+:area)
for (i = 0; i < NumRectangles; i++) {
/*printf("My thread ID is %d\n", omp_get_thread_num());*/
/* We want to calculate the x-coordinate where the rectangle intersects the
* circle by counting how many rectangle widths the rectangle is away from
* the origin */
x = i * width;

/* We want to use the x^2 + y^2 = r^2 equation to calculate the height of
* the rectangle, which is where the rectangle intersects the circle (y).
* Because the circle is a unit circle, r = 1. */
height = sqrt(1.0 - x * x);

/* We want to compute the area of the rectangle and add it to our running
* total */
area += width * height;

}

/* We are ready to compute pi. We need to multiply by 4 because we have found
* the area under only 1/4 of the unit circle. We don't need to multiply by
* radius squared, because the unit circle has a radius of 1. */
pi = 4.0 * area;

/* We want to print the final result */
printf("%f\n", pi);

/* We want to indicate the program has finished successfully using the Unix
* standard value for success */
return 0;
}
```

Conclusion : We successfully written a parallel program to calculate the value of PI/Area of Circle using OpenMP library.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 06	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Write a parallel program to multiply two matrices using openMP library and compare the execution time with its serial version.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 06

Aim : Write a parallel program to multiply two matrices using openMP library and compare the execution time with its serial version.

Theory :

Introduction

Parallel programming is a technique used to improve the performance of applications by splitting the work into multiple threads or processes that can run simultaneously on different CPUs or CPU cores. OpenMP is a popular API for parallel programming in C++, which provides a set of compiler directives, runtime library routines, and environment variables for shared memory parallel programming. This article will demonstrate how to use OpenMP for parallel matrix multiplication in C++.

Matrix multiplication is a fundamental operation in linear algebra, and it involves multiplying two matrices to produce a third matrix. The algorithm for matrix multiplication is computationally intensive and can benefit from parallelization. In our example, we will perform matrix multiplication on two square matrices of size 1000x1000, using serial and parallel approaches, and compare their performance.

Implementation

To begin with, we define three 2D vectors, A, B, and C, of size N x N, where N = 1000. These vectors will represent the two input matrices and the output matrix, respectively. We initialize matrices A and B with random values using nested loops.

```cpp
#include <iostream>
#include <vector>
#include <chrono>
#include <omp.h>

using namespace std;
const int N = 1000;
int main()
{
vector<vector<int>> A(N, vector<int>(N));
vector<vector<int>> B(N, vector<int>(N));
vector<vector<int>> C(N, vector<int>(N));

// Initialize matrices A and B with random values
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        A[i][j] = rand() % 100;
        B[i][j] = rand() % 100;
    }
}
```

Next, we perform matrix multiplication in serial using three nested loops. The outer loops iterate over the rows and columns of the output matrix C, and the innermost loop computes

the dot product of the corresponding row of matrix A and column of matrix B.

```cpp
auto start_serial = chrono::high_resolution_clock::now();
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int sum = 0;
        for (int k = 0; k < N; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
auto end_serial = chrono::high_resolution_clock::now();
auto duration_serial = chrono::duration_cast<chrono::milliseconds>(end_serial - start_
```

In the above code, we measure the time taken to perform matrix multiplication in serial using the C++11 chrono library. We start a timer before the nested loops and stop the timer after the loops are complete. The difference between the two timestamps gives us the elapsed time in milliseconds. We store this duration in a variable duration_serial.

Now, we will use OpenMP to parallelize the matrix multiplication. We will add an #pragma omp parallel for directive before the outer loop to indicate that the iterations of the loop can be executed in parallel. The OpenMP runtime library will automatically distribute the iterations among the available threads.

```cpp
// Perform matrix multiplication in parallel using OpenMP
   auto start_parallel = chrono::high_resolution_clock::now();
   #pragma omp parallel for
   for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            int sum = 0;
            for (int k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
    auto end_parallel = chrono::high_resolution_clock::now();
    auto duration_parallel = chrono::duration_cast<chrono::milliseconds>(end_parallel
```

Finally, we display the time taken for each approach using the chrono library, which provides high-resolution timers for measuring time intervals.

```cpp
// Display the time taken for each approach
cout << "Time taken for serial matrix multiplication: " << duration_serial.count() <<
cout << "Time taken for parallel matrix multiplication: " << duration_parallel.count(
    return 0;
}
```

Note

Enabling OpenMP in Visual Studio can be done in the following steps,

1. Open your C++ project in Visual Studio.
2. Right-click on your project in the Solution Explorer window and select "Properties".
3. In the Properties window, navigate to Configuration Properties -> C/C++ -> Language.
4. Set the "Open MP Support" option to "Yes (/openmp)".
5. Click "Apply" and "OK" to save the changes.

After enabling OpenMP, you can use OpenMP directives in your code to parallelize loops and other tasks. Remember that not all compilers support OpenMP, so it's essential to check that your compiler supports it before using it in your code.

Conclusion : Successfully written a parallel program to multiply two matrices using openMP library and compare the execution time with its serial version.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 07	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Implement a program to demonstrate balancing workload on MPI platform.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 07

Aim : Implement a program to demonstrate balancing workload on MPI platform.

Theory :

MPI (Message Passing Interface) is a standard specification for message-passing libraries used in parallel computing. It is commonly used for distributed memory systems, where multiple processors or nodes communicate and collaborate by passing messages to each other. Here's a brief introduction to MPI:

1. Purpose:

MPI is designed to facilitate communication and coordination among processes in a parallel computing environment.

It allows programs to be written in a parallel and distributed manner, where tasks can be executed simultaneously across multiple processors or nodes.

2. Features:

Point-to-point communication: MPI supports sending and receiving messages between individual processes.

Collective communication: MPI provides operations for collective communication, such as broadcasting, scattering, gathering, and reducing data across groups of processes.

Process management: MPI allows for dynamic process creation and management within a parallel application.

Datatypes: MPI supports defining and using custom data types for efficient communication and data manipulation.

Error handling: MPI provides mechanisms for error detection and handling in parallel applications.

3. Usage:

MPI is widely used in high-performance computing (HPC) and parallel computing environments, such as supercomputers and clusters.

It is used for developing parallel applications that can take advantage of distributed memory architectures and scale across multiple processing units.

4. Programming Models:

MPI follows a message-passing programming model, where processes communicate by explicitly sending and receiving messages.

MPI programs are typically written in languages like C, C++, or Fortran, using MPI library functions and constructs.

5. Implementation:

MPI implementations are available from various vendors and organizations, such as MPICH, Open MPI, and Intel MPI.

These implementations provide libraries, compilers, and runtime environments for developing and running MPI-based applications.

Overall, MPI is a powerful and widely-used standard for parallel computing, providing programmers with tools and techniques to develop efficient and scalable parallel applications for distributed memory systems.

Certainly! Below is an example of a program in C using MPI that demonstrates workload balancing by dividing a workload (represented as an array) among MPI processes and showing the output of each process.

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

#define ARRAY_SIZE 100

int main(int argc, char *argv[]) {
    int rank, size;
    int workload[ARRAY_SIZE];
    int local_workload[ARRAY_SIZE];
    int i, sum = 0, local_sum = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Generate random workload on rank 0
    if (rank == 0) {
        for (i = 0; i < ARRAY_SIZE; i++) {
            workload[i] = rand() % 10;  // Random workload between 0 and 9
            sum += workload[i];  // Calculate total workload
        }
    }

    // Broadcast total workload to all processes
    MPI_Bcast(&sum, 1, MPI_INT, 0, MPI_COMM_WORLD);

    // Calculate local workload for each process
    int chunk_size = ARRAY_SIZE / size;
    MPI_Scatter(workload, chunk_size, MPI_INT, local_workload, chunk_size,
MPI_INT, 0, MPI_COMM_WORLD);

    // Calculate local sum
    for (i = 0; i < chunk_size; i++) {
```

```
        local_sum += local_workload[i];
    }

    // Reduce local sums to get the final result
    int total_sum;
    MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

    // Print the result on each process
    printf("Process %d local sum: %d\n", rank, local_sum);

    // Print the total sum on rank 0
    if (rank == 0) {
        printf("Total workload: %d\n", sum);
        printf("Balanced workload result: %d\n", total_sum);
    }

    MPI_Finalize();
    return 0;
}
```

In this program:

- Each MPI process calculates a local workload based on the total workload, which is initially generated on rank 0.
- The total workload is broadcasted to all processes using MPI_Bcast.
- The total workload is divided evenly among processes using MPI_Scatter.
- Each process calculates its local sum.
- The local sums are reduced to get the final result using MPI_Reduce.

Conclusion : We have successfully **i**mplemented a program to demonstrate balancing workload on MPI platform.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 08	
Subject : High Performance Computing	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Install MPICH library and write a "Hello World" program for the same.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 08

Aim : Install MPICH library and write a "Hello World" program for the same.

Theory :

MPICH, which stands for Message Passing Interface for Computing in Heterogeneous environments, is a portable and widely-used implementation of the MPI (Message Passing Interface) standard. Here's a brief introduction to MPICH:

1. Purpose:
   - MPICH is designed to facilitate communication and coordination among processes in parallel and distributed computing environments.
   - It allows developers to write parallel applications that can execute on distributed memory systems, such as clusters, supercomputers, and multi-core processors.

2. Features:
   - MPI Compliance: MPICH adheres to the MPI standard, which defines a set of functions, data types, and communication protocols for parallel programming.
   - Portability: MPICH is highly portable and can run on a variety of platforms, including Linux, macOS, Windows, and various Unix-based systems.
   - Scalability: MPICH is scalable and can efficiently handle parallel applications with a large number of processes or nodes.
   - Performance Optimization: MPICH provides optimizations for communication and data movement to enhance performance in parallel computing tasks.
   - Customization: MPICH offers flexibility for customization and configuration to meet specific requirements of parallel applications.

3. Usage:
   - MPICH is widely used in scientific computing, engineering simulations, data analytics, and other domains that require parallel processing.
   - It is utilized by researchers, developers, and organizations to develop and run high-performance parallel applications.

4. Components:
   - MPICH includes libraries, compilers, and runtime environments for developing and executing MPI-based parallel programs.
   - It provides command-line tools such as mpicc (MPI C Compiler), mpiexec (MPI Process Manager), and mpirun (MPI Program Launcher) for compiling and running MPI applications.

5. Implementation:
   - MPICH is an open-source project developed and maintained by a community of contributors, including researchers and developers from academia and industry.
   - It supports various MPI versions, including MPI-1, MPI-2, MPI-3, and MPI-4, providing compatibility with different MPI-based applications.

Overall, MPICH is a robust and versatile MPI implementation that enables efficient and scalable parallel computing on distributed memory systems. It provides a standardized approach to message passing and parallel programming, making it a valuable tool for high-performance computing and scientific research.

To install the MPICH library and write a "Hello World" program using MPICH, you can follow these steps. These instructions assume you are using a Linux-based system.

1. **Install MPICH**:
   - Open a terminal window.
   - Update your package list to ensure you have the latest package information:

     ```sql
     sudo apt update
     ```

   - Install MPICH using the package manager (in this case, `apt` for Debian-based systems like Ubuntu):

     ```
     sudo apt install mpich
     ```

2. **Verify Installation**:
   - After installation, you can verify that MPICH is installed correctly by checking the version:

     ```
     mpichversion
     ```

3. **Write a "Hello World" Program**:
   - Create a file named `hello_world_mpi.c` and add the following code using a text editor of your choice:

     ```c
     #include <stdio.h>
     #include <mpi.h>

     int main(int argc, char *argv[]) {
         int rank, size;

         MPI_Init(&argc, &argv);
         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
         MPI_Comm_size(MPI_COMM_WORLD, &size);

         printf("Hello world from process %d of %d\n", rank, size);

         MPI_Finalize();
         return 0;
     }
     ```

4. **Compile the Program**:
   - Open a terminal and navigate to the directory containing your `hello_world_mpi.c` file.
   - Compile the program using the MPICH wrapper compiler (`mpicc`):

```
mpicc -o hello_world_mpi hello_world_mpi.c
```

5. **Run the Program**:
   - After successful compilation, run the program using the MPICH launcher (`mpiexec` or `mpirun`):

```
mpiexec -n 4 ./hello_world_mpi
```

   Replace `4` with the number of MPI processes you want to run. In this case, `./hello_world_mpi` will run with 4 MPI processes, each printing its rank and the total number of processes.

6. **Verify Output**:
   - You should see output similar to the following (the exact output may vary depending on the number of processes you specify):

```
Hello world from process 0 of 4
Hello world from process 1 of 4
Hello world from process 2 of 4
Hello world from process 3 of 4
```

That's it! You have successfully installed MPICH and run a "Hello World" program using MPICH on your Linux system. Adjust the number of MPI processes (`-n`) in the `mpiexec` command to observe different outputs with varying numbers of processes.

Conclusion : We have successfully installed MPICH library and written a "Hello World" program for the same.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 01	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Analysis of Data Source Using Autopsy.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 01

Aim : Analysis of Data Source Using Autopsy.

Theory :

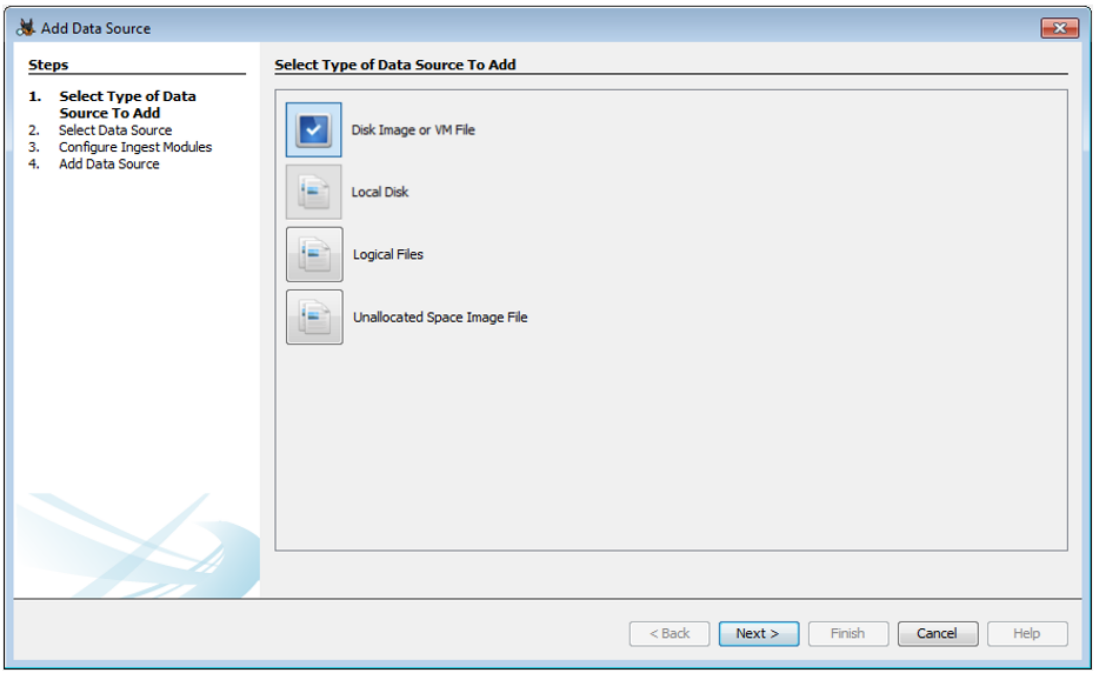
The Sleuth Kit is a library and a collection of command-line tools used to investigate disk images. Autopsy is the GUI program for TSK. The results of the forensic search carried over the images are displayed here. These results help the investigator to locate relevant sections of data in their investigation. It is used by law enforcement, military, and corporate examiners to investigate the actions taken place on the evidence computer, however, it can be used to recover deleted data from digital devices too. Autopsy performs operations onto disk images which can be created using tools like FTK Imager. Here an already created image is used. You may download Autopsy from here .

1. **Getting Started**
   Open Autopsy and create a new case.

Click on Finish after completing both the steps.

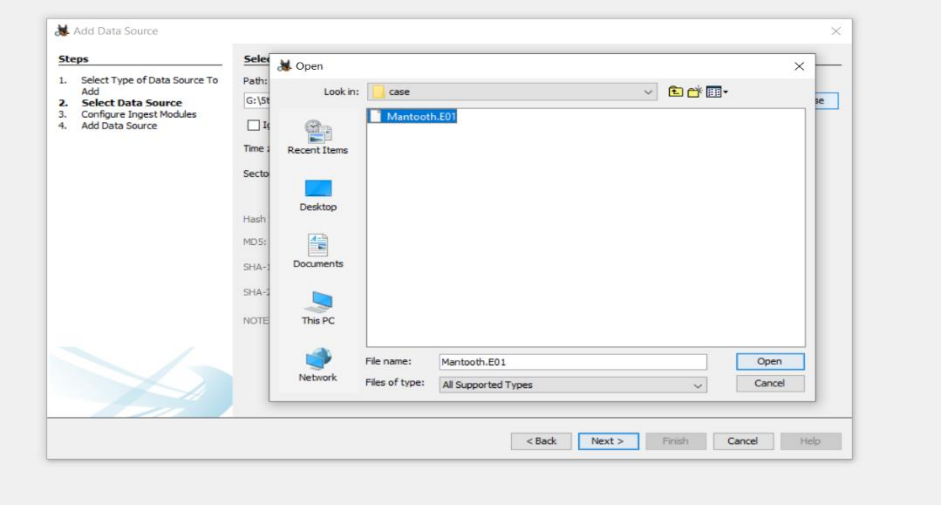
2. **Add a data source.**

Select the appropriate data source type.

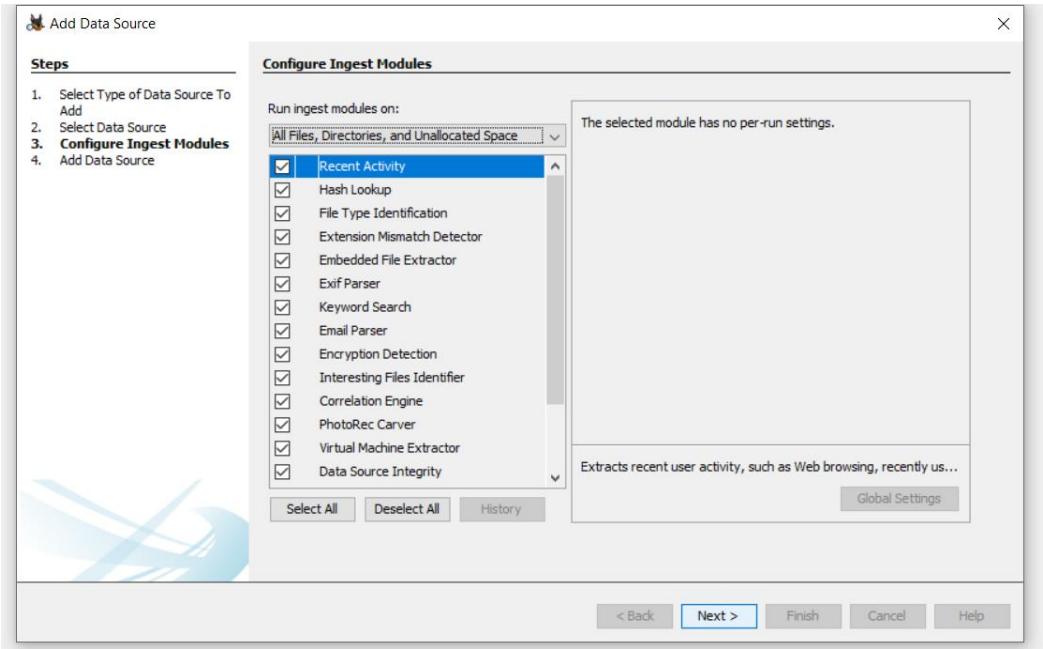


- Disk Image or VM file: Includes images that are an exact copy of a hard drive or media card, or a virtual machine image.
- Local Disk: Includes Hard disk, Pendrive, memory card, etc.
- Logical Files. : Includes local folders or files.
- Unallocated Space Image File: Includes files that do not contain a file system but need to run through ingest.

The data source used here is a disk image. Add the data source destination.



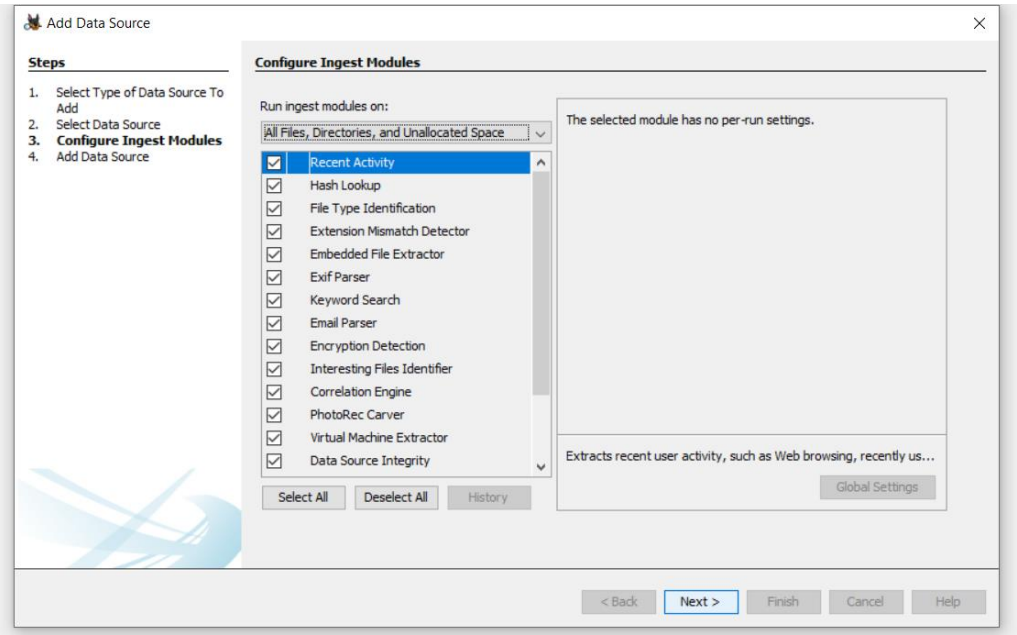
Configure ingest modules.



The ingest modules determine factors for which the data in the data source is to be analyzed. Here is a brief overview of each of them.

- Recent Activity: Discover the recent operations performed on the disk, for example, the files that were last viewed.
- Hash Lookup: Identify files using hash values.
- File Type Identification: Identify files based on their internal signatures rather than just file .extensions.
- Extension Mismatch Detector: Identify files whose extensions are tampered with/changed possibly to hide evidence.
- Embedded File Extractor: It extracts embedded files such as .zip, .rar, etc. and uses the derived file for analysis. Another example could be a PNG image saved inside a doc to make it appear as a document and thus hide crucial information.
- EXIF (Exchangeable Image File Format) Parser: It is used to retrieve metadata about the files, for example, date of creation, geolocation, etc.
- Keyword Search: Search for a particular keyword/pattern in the data source.
- Email Parser: If the disk holds any form of email database, for example, pst/ost files of outlook then information from these files can be extracted using an email parser.
- Encryption Detection: Detects and identifies encrypted / password-protected files.
- Interesting File Identifier: Let’s set custom rules regarding the filtering of data. Examiner is notified when results pertaining to these rules are found.

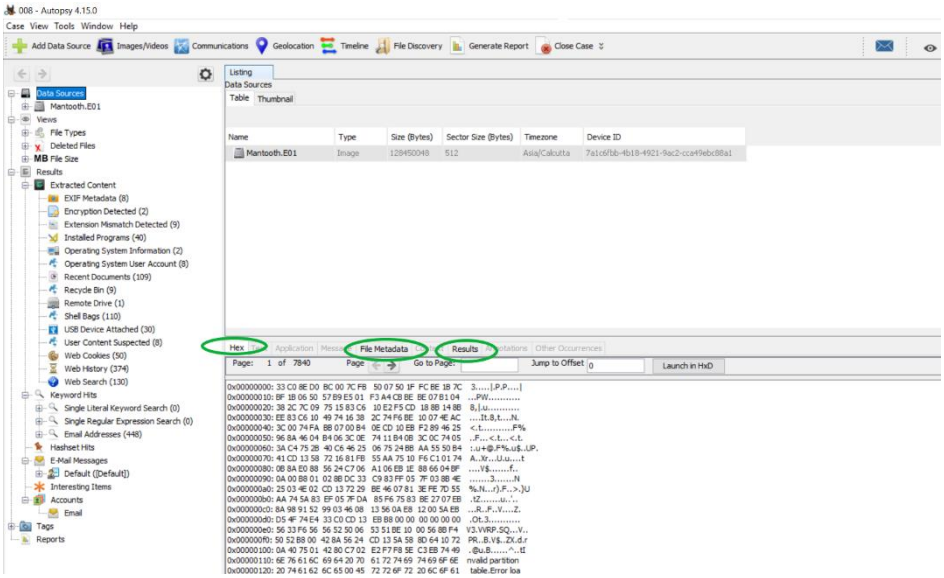
- Correlation Engine: Allows saving properties in and then retrieved from the central repository. It helps in displaying correlated properties.
- PhotoRec Carver: Recover files, photos, etc. from the unallocated space.
- Virtual Machine Extractor: Extract and analyze any Virtual machine found on the data source.
- Data Source Integrity: Calculates the hash values and stores them in the database in case they aren't already present. Otherwise, it will verify the hash values associated with the database.
- Plaso: Extract timestamp for various types of files.
- Android Analyzer: Analyze SQLite and other files retrieved from an Android device.



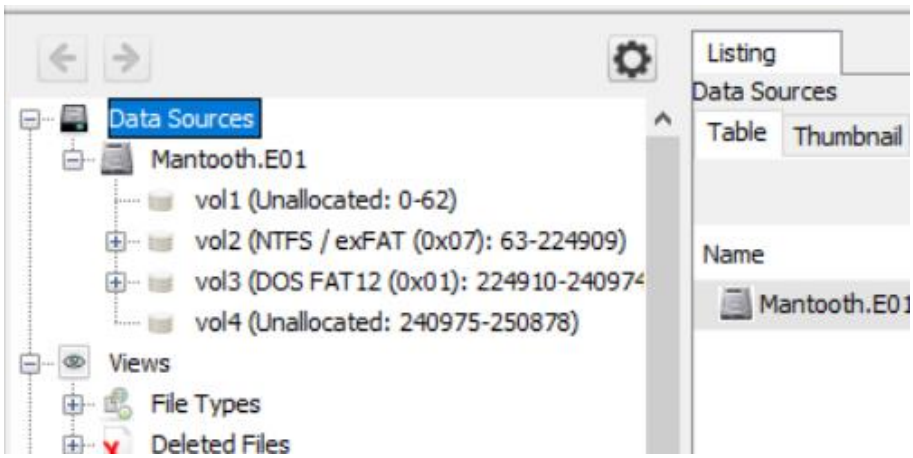
Select all that will serve the purpose of your investigation and click Next. Once the data source is added, click Finish. It will take some buffer time to extract and analyze the data depending upon the size of the Data Source.

3. Exploring the data source:

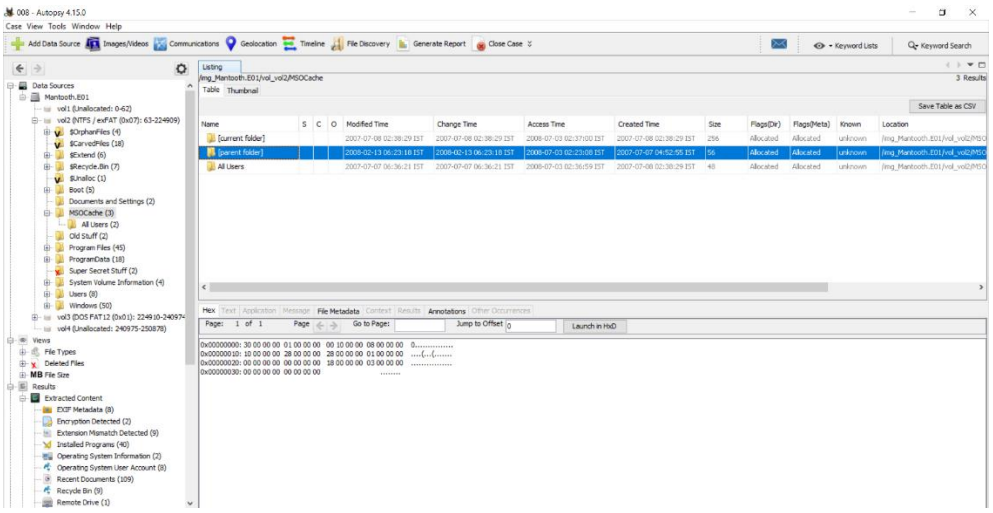
The Data Source information: Here the basic metadata is shown. A detailed analysis is displayed in the bottom section. These details can be extracted in the form of Hex values, Results, File Metadata, etc.



The disk image is then broken down based upon its volume partitions.

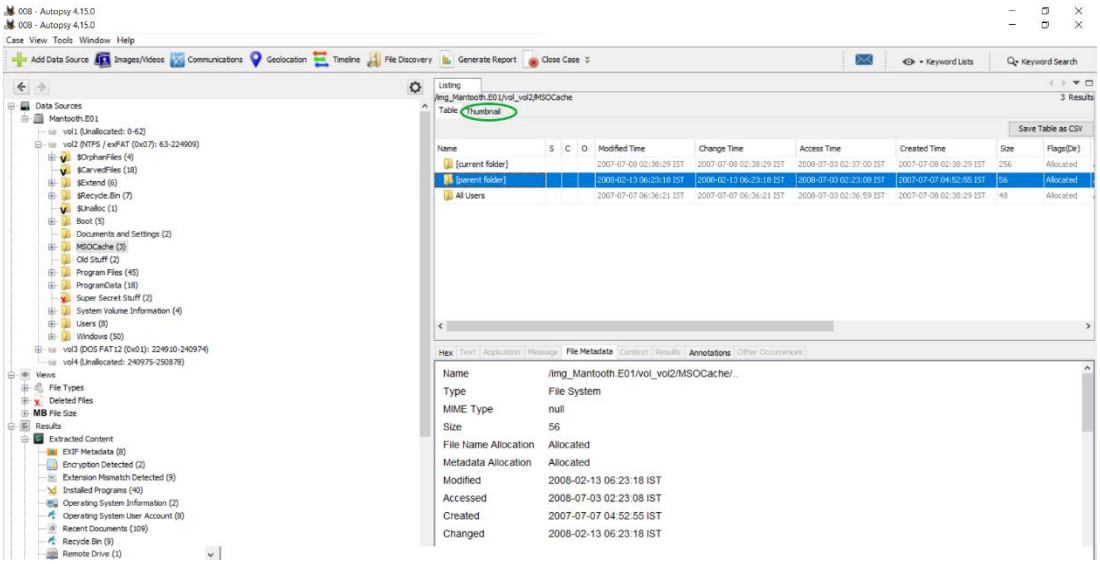


Each volume can be browsed for its contents, results for which are displayed in the section at the bottom. For example, the content shown below belongs to Data Sources -> Mantooth.E01 -> MSOCache-> [Parent Folder].

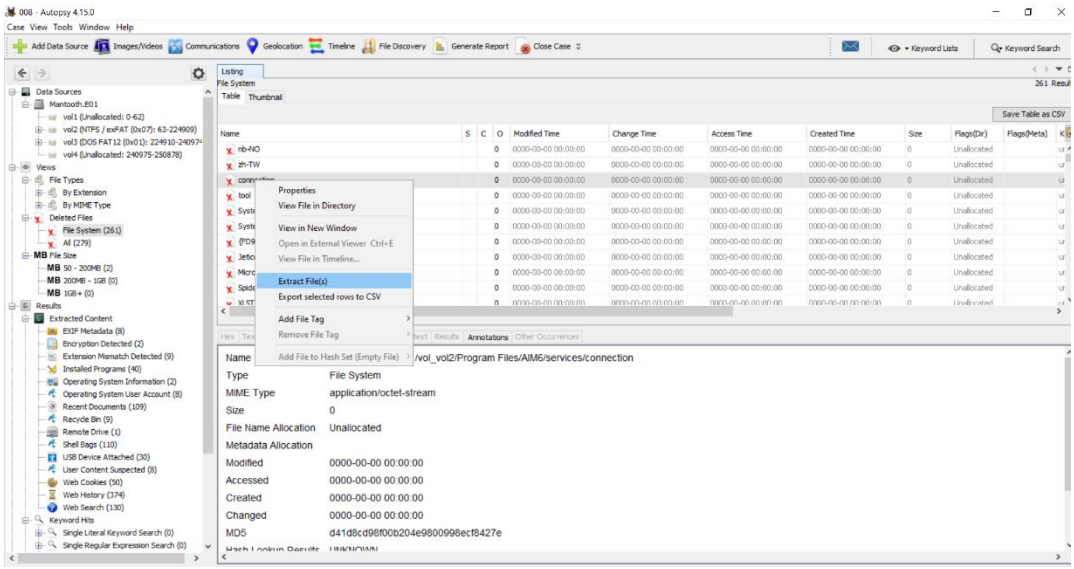


Views (Determines the factor of file classification)

- File Type: Here the files are categorized based upon their type. The classification can be done either on the basis of file .extension or MIME type. While both of these provide a hint about how to deal with a file, file extensions are commonly used by the OS to decide what program shall be used to open a file and MIME types are used by the browser to decide about how to present the data (or by the server on how to interpret the data received). Files displayed here also include the deleted files.



- Deleted Files: Here information about the files that were specifically deleted can be found. These deleted files can be recovered as well: Right-click on the file to be recovered -> click on Extract File(s). -> Save the file in an appropriate destination.



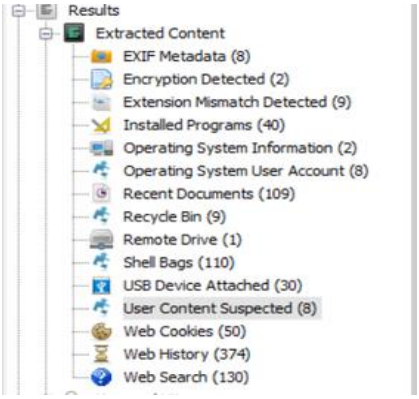
- MB Size Files: Here files are classified based upon their size. The range starts from 50MB. This enables the examiner to determine exclusively large files.

Note: It is usually advised to not scan or extract any suspected files/ disks such as payload files, etc. in the main system, rather scan them in safe environments such as a virtual machine, and then extract the data, as they hold the possibility of being corrupt and may infect the examiner's system with viruses.

Results:

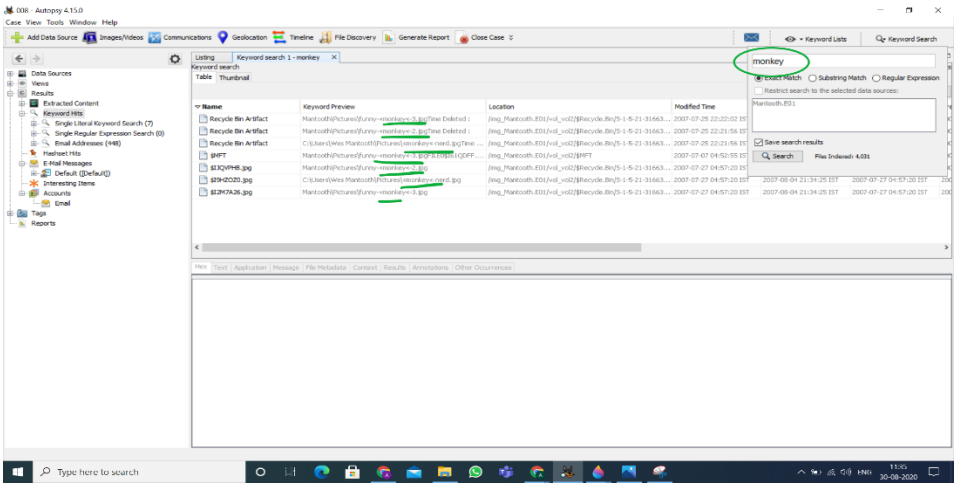
All the extracted data is viewed in Views/ Data Source. In Results, we get the information about this data.

- Extracted Content: Each Extracted Content displayed below can be further explored. The following briefly explains each of them.

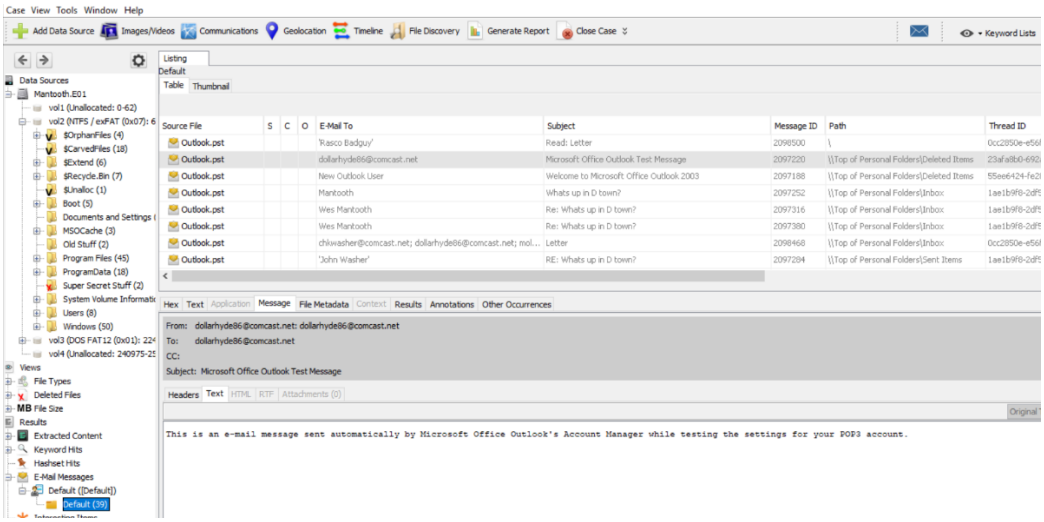


- EXIF Metadata: It contains all the .jpg images that have EXIF Metadata associated with them, this Metadata can be analyzed further.
- Encryption Detection: It detects files that are password protected/ encrypted.
- Extension Mismatch Detection: As explained above, it Identifies the files whose extensions do not match their MIME types and thus they may be suspicious.
- Installed Programs: It gives details about the software used by the user. This information is extracted with the help of the Software Registry hive.
- Operating System Information: It gives information about the OS with the help of the Windows Registry hive and the Software Registry hive.
- Operating System User Account: It lists information about all the user accounts, for example, accounts belonging to the device are extracted from the Software Hive and the accounts associated with the Internet Explorer using index.data files.
- Recent documents: Lists all the documents that were accessed nearby the time the disk image was captured.
- Recycle Bin: Files that are temporarily stored on the system before being permanently deleted are visible here.
- Remote Drive: Shows information about all the remote drives accessed using the system.
- Shell bags: A shell bag is a set of registry keys that stores details about a folder being viewed, such as its position, icon, and size. All the Shell bags from the system can be viewed here.

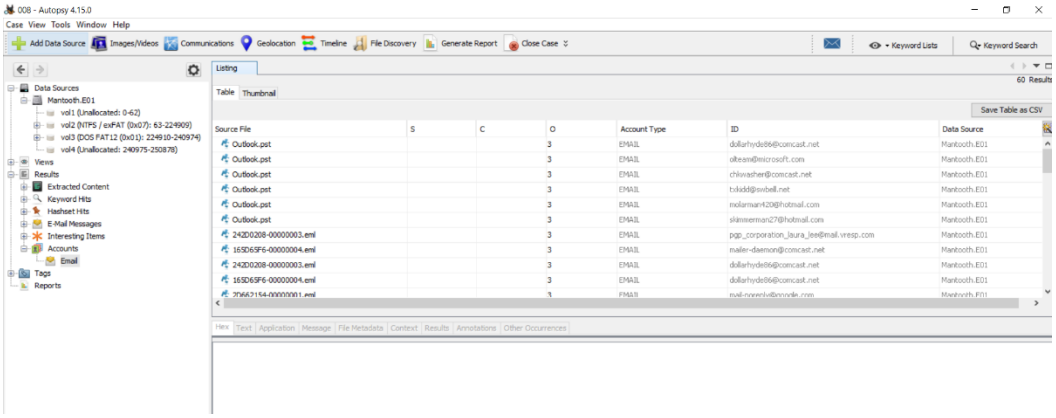
- USB Device attached: All the information about the external devices attached to the system is displayed here. This data is extracted from Windows Registry which is actually a maintained database about all the activities taking place on the system.
- Web Cookies: Cookies saves the user information from the sites and thus provide a lot of information about the user's online activities.
- Web History: All the details about the browser history is shown here.
- Web Searches: Details about the web searches made are displayed here.
- Keyword Hits: Here specific keywords can be looked for in the image of the disk. Multiple data sources can be selected for the lookup. The search can be restricted to Exact match, Substring match and Regular expression, for example, emails/ IP Addresses, etc.



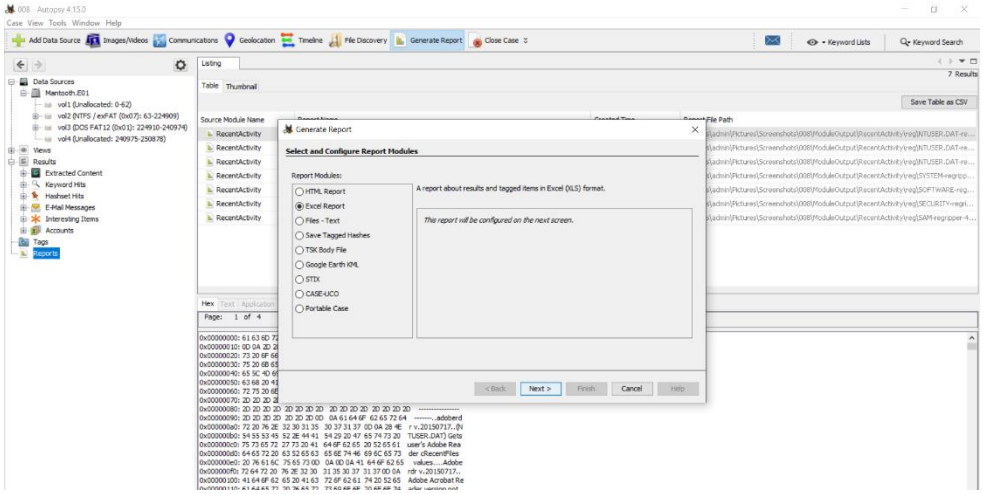
- HashSet Hits: Here the search can be made using hash values.
- E-mail Messages: Here all the outlook.pst files can be explored.

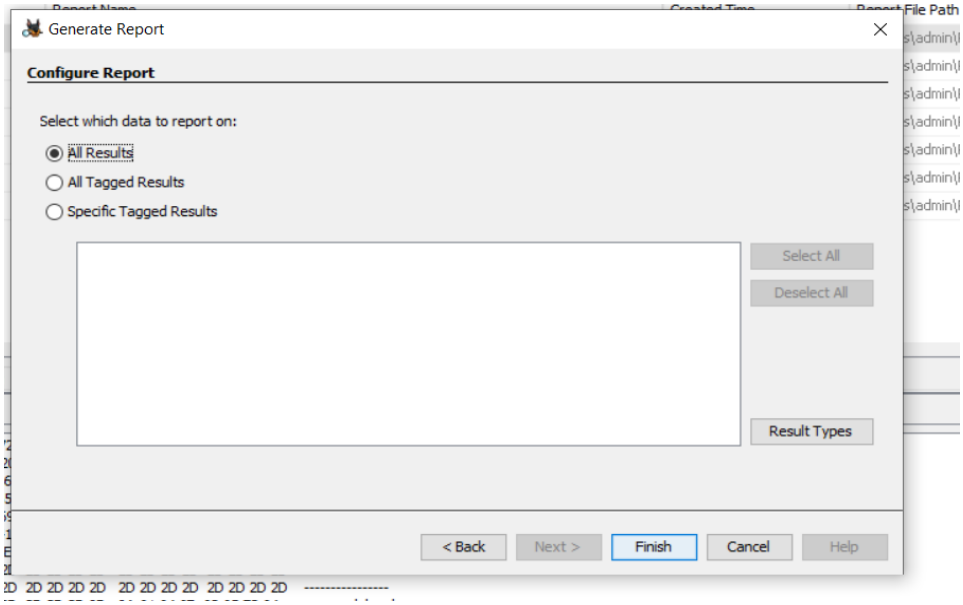


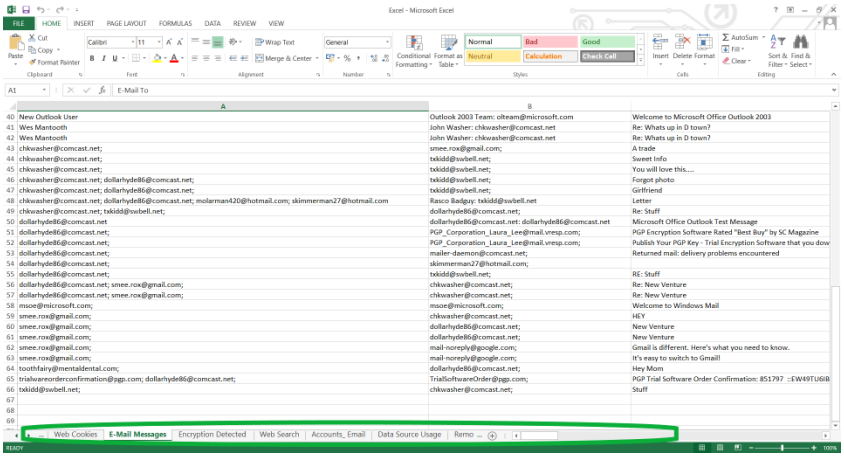
- Interesting Items: As discussed before, these are the file results based upon the custom rules set by the examiner.
- Accounts: Here all the details regarding the accounts present on the disk are shown. This disk has the following EMAIL accounts.



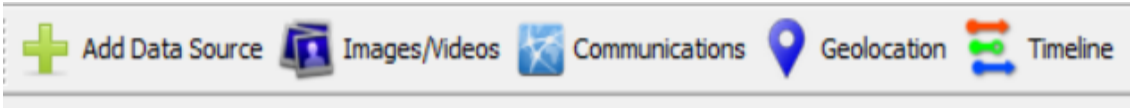
- Reports: Reports about the entire analysis of the data source can be generated and exported in many formats.



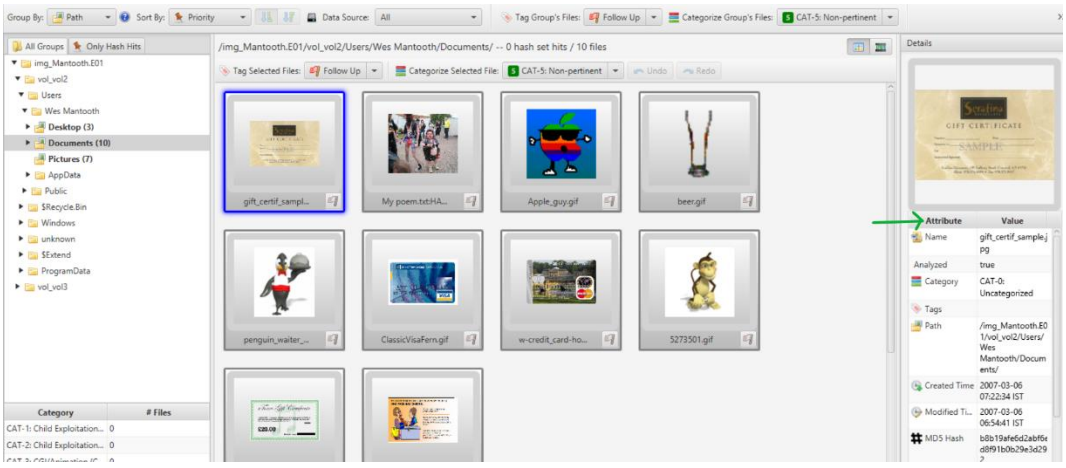




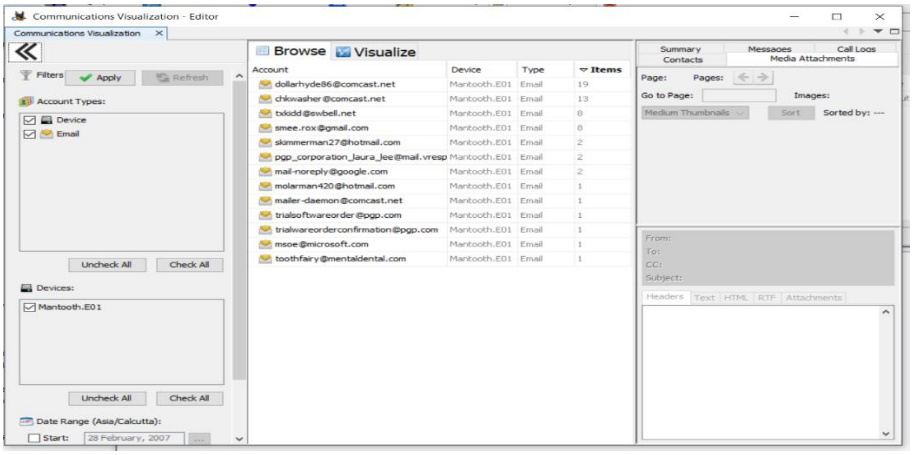
Additional Features:



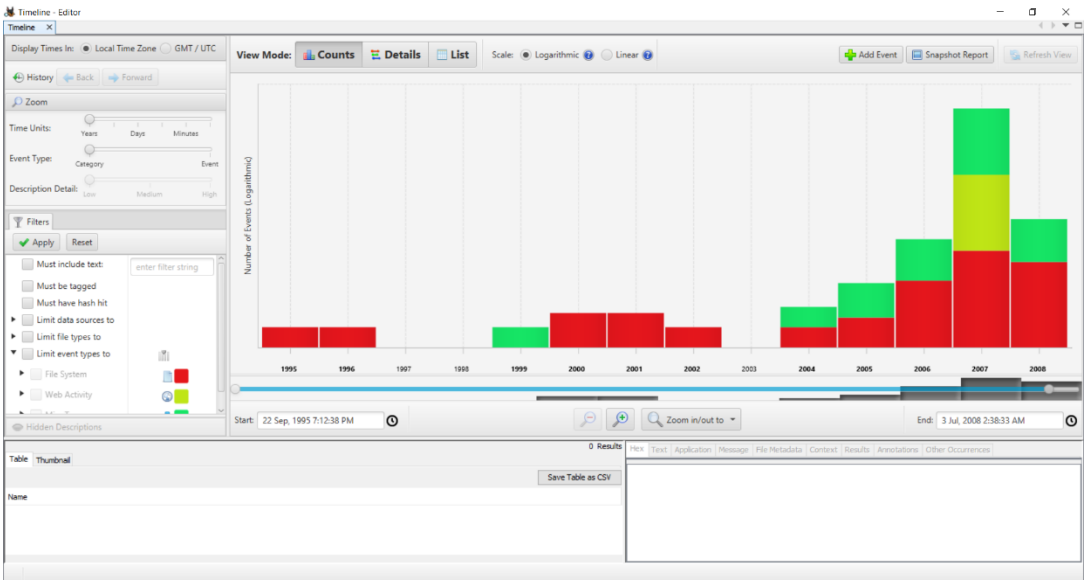
- Add a Data Source: Each case can hold multiple Data Sources.
- Images/Videos: Images/ Videos in the data source can be viewed in Gallery View. The information here is displayed in the form of attribute-value pairs.



- Communications: All the communications made using the source device are displayed here. This device had communications only in the form of emails.



- Geolocation: This window displays the artifacts that have longitude and latitude attributes as waypoints on a map. Here the data source has no waypoints.
- Timeline: Information about when the computer was used or what events took place before or after a given event can be found, this greatly helps in investigating events near about a particular time.



Almost all the basic features and how actually Autopsy works have been discussed in this article. However, it is always recommended to go through different sample data sources to explore even more.

Conclusion : We have successfully studied Analysis of Data Source Using Autopsy.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 02-A	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Analysis of forensic images using FTK Imager.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 02-A

Aim : Analysis of forensic images using FTK Imager.

Theory :

FTK Imager is a data preview and imaging tool; it is used to create forensic images of target computer data without making any changes to the original evidence. By using this tool, you can create forensic images of local hard drives, floppy diskettes, zip disks, CDs, DVDs, entire folders, or individual files from various places within the media. FTK Imager can also be used to perform jobs other than acquiring images, like the following:

1. Mounting an image for read-only.
2. Previewing the contents of forensics images.
3. Exporting files/folders from forensics images.
4. Acquiring Windows registry.
5. Recovering deleted files.

The tool can be either installed locally on where it will be used or run from within a USB thumb drive connected to a machine in the field (the latter is preferred when conducting live forensics on running systems).

Capture RAM with FTK Imager

Before we begin using this tool, we will demonstrate how to install it on a portable media device (e.g., USB thumb drive):

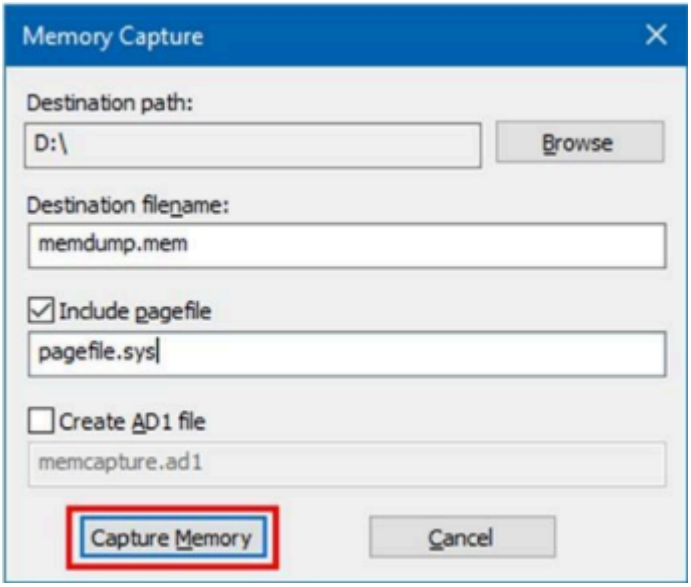
1. Go to https://accessdata.com/product-download and select which version of "FTK Imager" you want to download. You need to fill in a registration form (enter your full name, e-mail address, and job title, among other details); after that, a download link will be sent to your specified e-mail address to download the tool.
2. Execute the installer. Now you have two options to install it on your USB thumb drive:

a. Run the installation on a local computer, then copy the FTK Imager folder from the [Drive Letter]:\Program Files\ AccessData\FTK Imager to the USB thumb drive.

b. Install the FTK Imager files directly to the thumb drive, avoiding installing to a local computer first. The installer will unzip the downloaded files to the portable drive; after then, that USB can be connected to any computer running a Windows OS, and the program file (FTK Imager.exe) can be executed from the portable USB device.

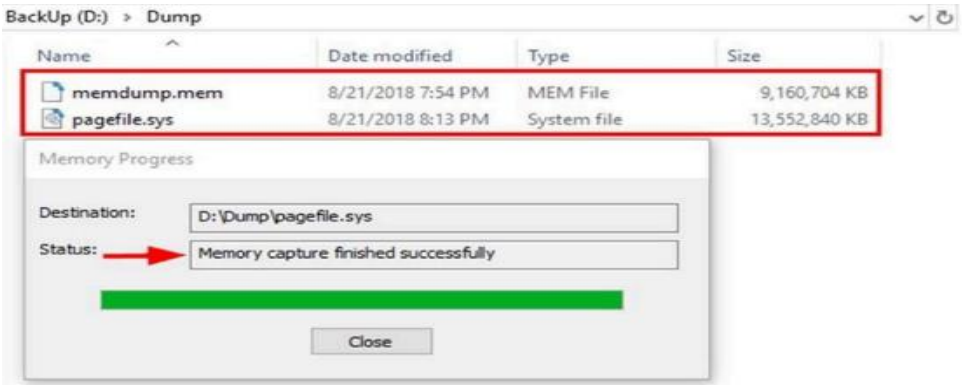
To use this tool for RAM capture, do the following:

Launch FTK Imager from the USB thumb drive (if you select to install it on a USB as we already demonstrated). Navigate to File ➤ Capture Memory. A new window appears showing options for capturing the RAM memory of the current machine



1. The opened window will prompt the user to select the destination where s/he wants to store the resulting RAM image; you can also select whether you want to include the pagefile (pagefile.sys). When everything is set, click the "Capture Memory" button.
2. Now, the capture will begin and a progress bar will appear showing the capture progress. Once the capture is complete, the open window will announce whether or not the memory dump completed successfully.

Browse to the folder where you saved the memory dump. You should find two files, memdump.mem (or whatever you named it) and pagefile.sys, if you select to acquire it (see Figure 5-5). These two files contain the entire contents of RAM memory when the dump was processed.



RAM and pagefile.sys (virtual memory) dump acquired by FTK Imager

The captured RAM image can be further analyzed to extract important information like passwords, temporary Internet files, deleted files,

Using FTK Imager to Capture Hard Drive

We have already used this tool to capture RAM memory; capturing hard drive with it is similar.

1. If you did not download and install the tool yet, go to this chapter's section titled "Capture RAM with FTK Imager" and read the relevant instructions.
2. Launch AccessData FTK Imager and go to File ➤ Create Disk Image…
3. A new window appears (); here you will need to select the source evidence type. You have five options:
    - Physical Drive: This is the most commonly used option. A physical drive will allow you to capture all data (bit by bit) on a hard drive; unallocated space and deleted files will get captured too.
    - Logical Drive: Select a specific partition within a drive to capture; for example, capture the D:\ drive only.
    - Image File: Here you can select an image file as a source.
    - Contents of a Folder: Select a folder as a source.
    - Fernico Device: Restore forensics images from multiple sources (multiple CD/DVDs).

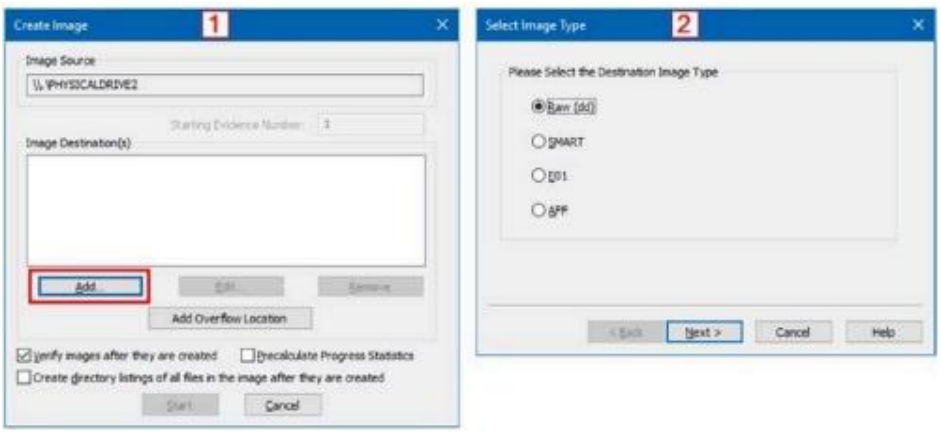
      Out of given 4 options choose logical drive to capture/aquire E:/ drive of machine

FTK Imager "Select Source" window allows you to select the type of acquisition you want to perform

Out of given 4 options choose logical drive to capture/aquire E:/ drive of machine

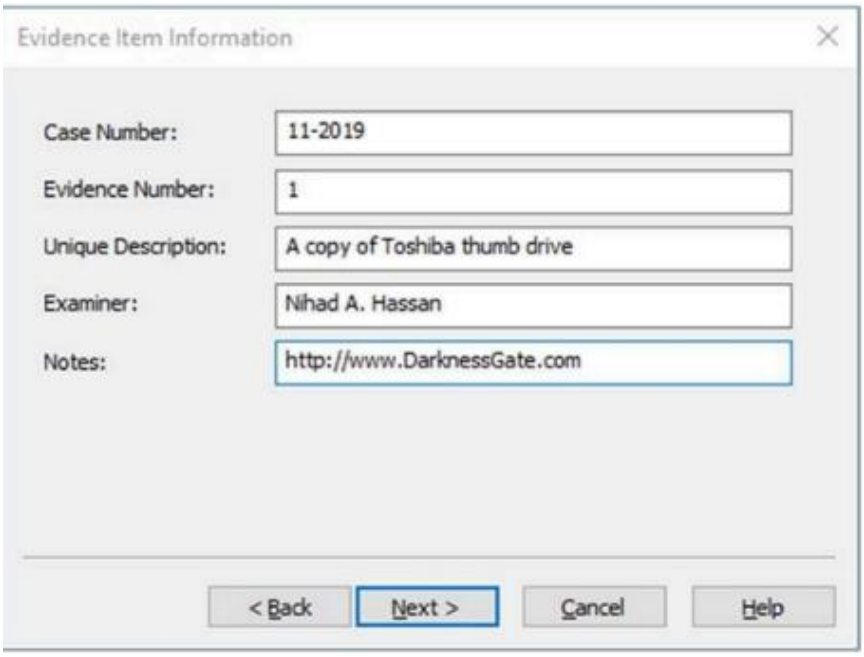
Select which physical drive you want to image

4. Click the "Finish" button to move to the next window; here you will be asked where you want to save the drive image. Click the "Add" button and a new dialog will appear asking you to choose the image format



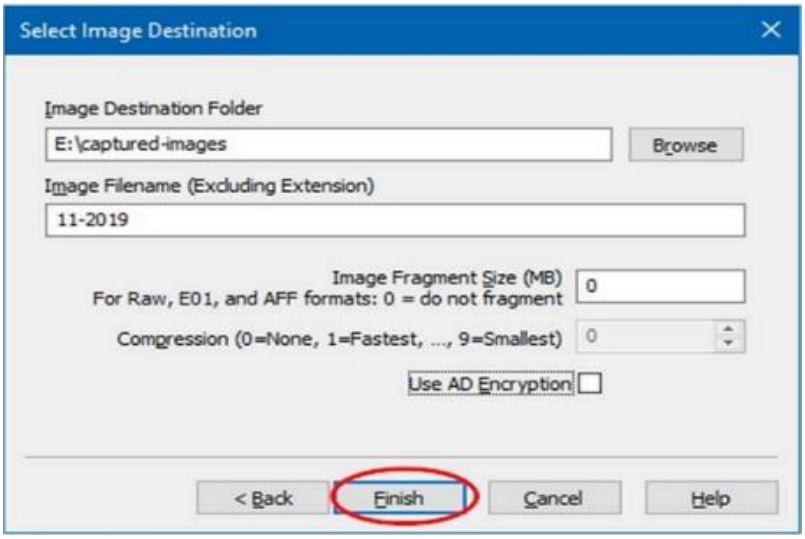
FTK Imager “Create Image” window allows you to select image location and image type

5. In our case, we will select “Raw (dd)”; click the “Next” button and a new window will appear asking you to enter evidence information, which includes case number, evidence number, unique description, examiner, and notes. All fields are optional (see Figure 5-9). Click “Next” to continue.



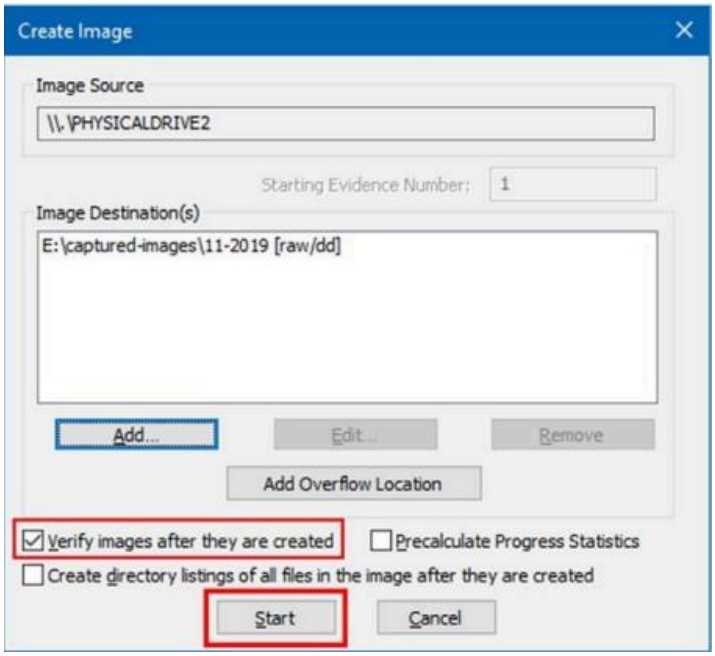
Enter evidence item information in FTK Imager

The next window will allow you to select where you want to store this image; you can also specify the file name to use and whether or not you want to split the image up into multiple fragments (fragment size is measured in megabytes). Remember that you cannot split a forensic image that has a Raw file format. If you want to secure the image with a password, tick the option “Use AD Encryption” and enter a password twice after clicking the “Finish” button. FTK Imager uses AES-256 encryption algorithm to secure the image.



The “Select Image Destination” window allows you choose where to store the acquired image, its name, whether you want to fragment it or compress it, and whether you want to encrypt it (protect it with a password)

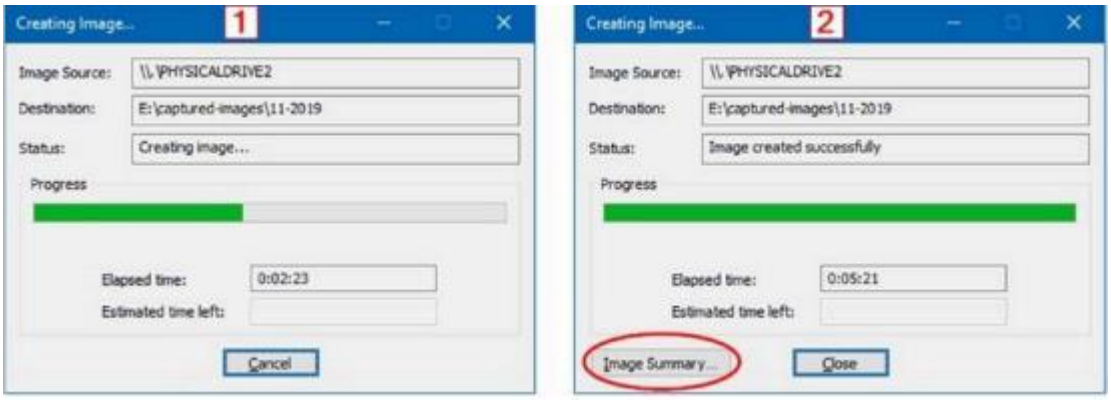
In our case, we do not need to secure the image with a password, so we will click the “Finish” button. This will bring us back to the original “Create Image” screen with the “Start” button enabled



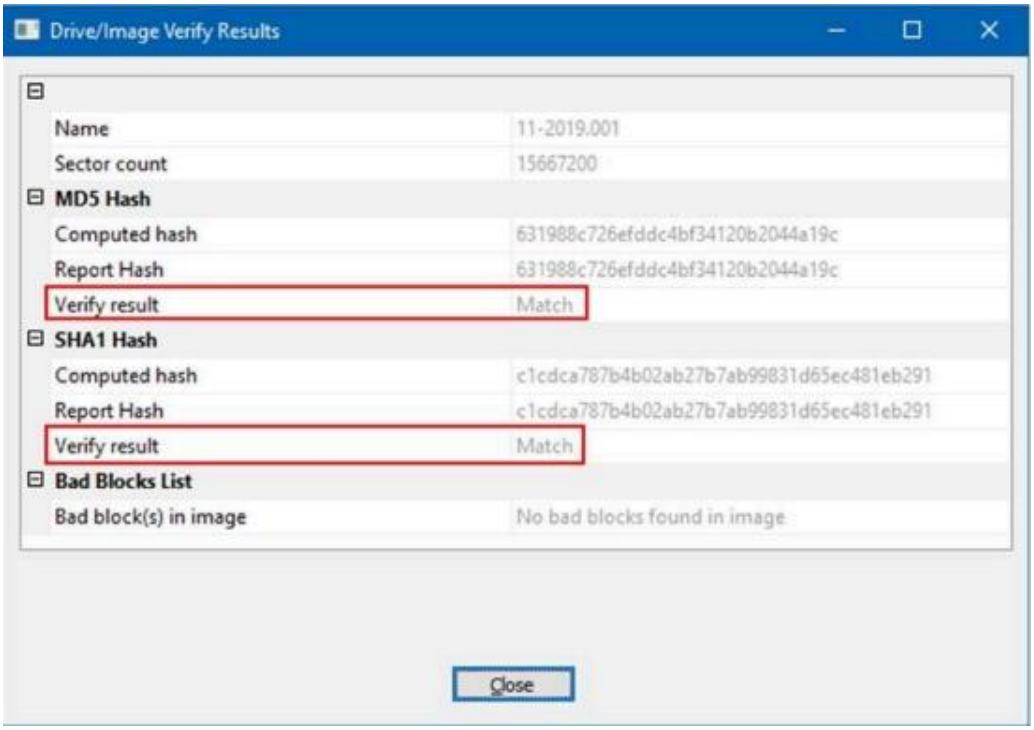
FTK Imager “Create Image” window ready to start the acquisition process

1. Make sure the option “Verify images after they are created” is checked so that you can ensure that the source drive and the resultant image are 100% equal. Now, after everything is settled, press the “Start” button to begin the acquisition process.

A window with a progress bar will appear showing you the acquisition progress. Once the image has been created, the verification process begins.

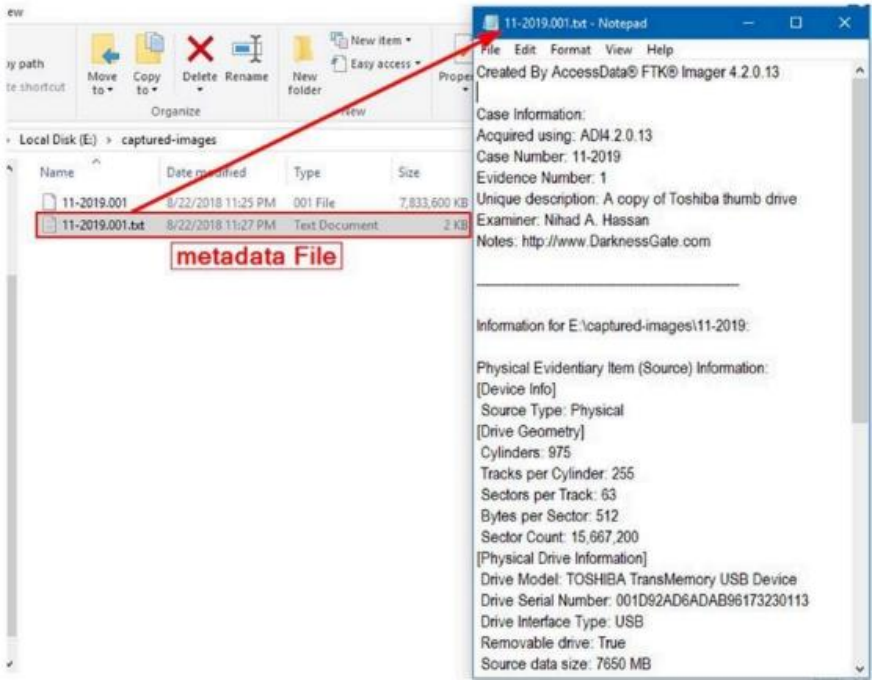


1. After finishing the verification process, the “Verify Results” window will appear (see Figure 5-13).



Verify Results window: Hash value of the captured data is identical to the resultant image file hash value.

Go to the folder where you choose to store the acquired image. There, you should find two files the image file and the associated metadata file (text file). FTK Imager has created a separate metadata file for the image because we selected to store the image in Raw format.



Separate metadata file is associated with the captured image when using the Raw file format for acquiring the image file.

To restore deleted images from captured image, follow these steps:

1. Go to file menu.
2. choose mount image option.
3. choose captured image file source from c: drive location Chose mount type as logical mount
4. click on mount option
5. close mount window
6. go to file menu again
7. select show all drives options.
8. select newly mounted drive.

explore currently available and deleted data using tree control.

Conclusion : We have successfully studied Analysis of forensic images using FTK Imager.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 02-B	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- AUTOPSY -Digital Forensics Investigation using Autopsy In Kali Linux.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 02-B

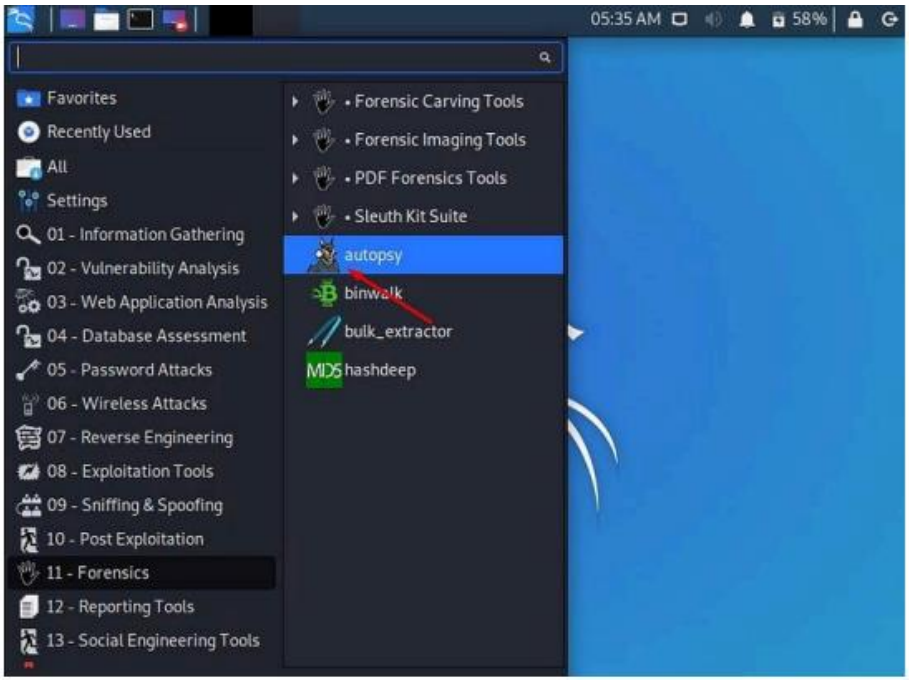
Aim : AUTOPSY -Digital Forensics Investigation using Autopsy In Kali Linux.

Theory :

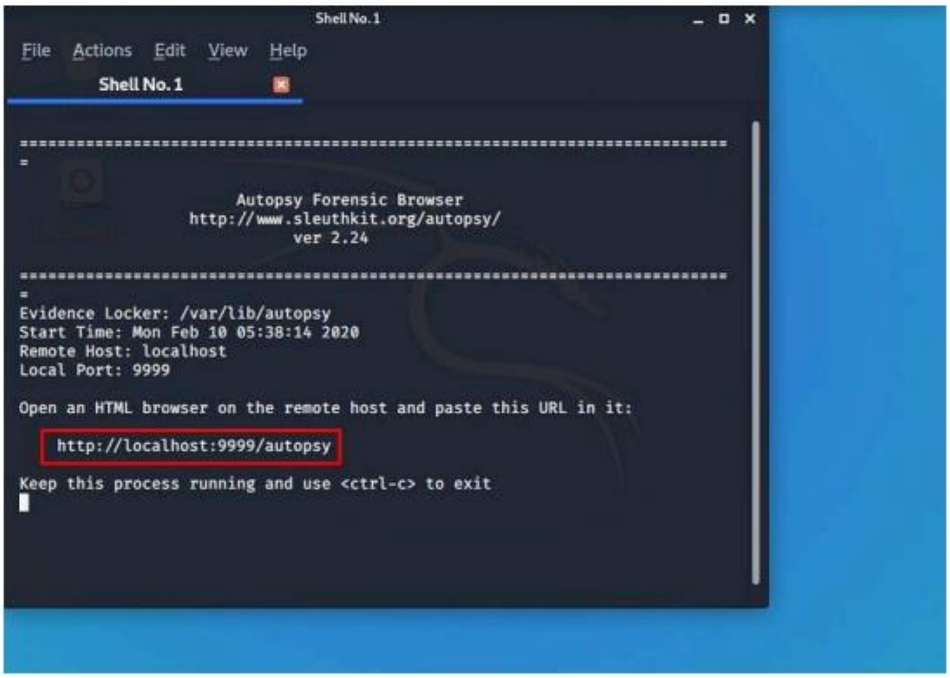
Autopsy is one of the digital forensics tools use to investigate what happened on a computer. It offers a GUI access to variety of investigative command-line tools from The Sleuth Kit including image file hashing, deleted file recovery, file analysis and case management. Autopsy produces results in real time, making it more compatible over other forensics tools.

It comes preinstalled in Kali Linux so Lets start the Kali Virtual Machine. You will find the option ‘forensics’ in the application tab. Select ‘autopsy’ from the list of forensics tools.

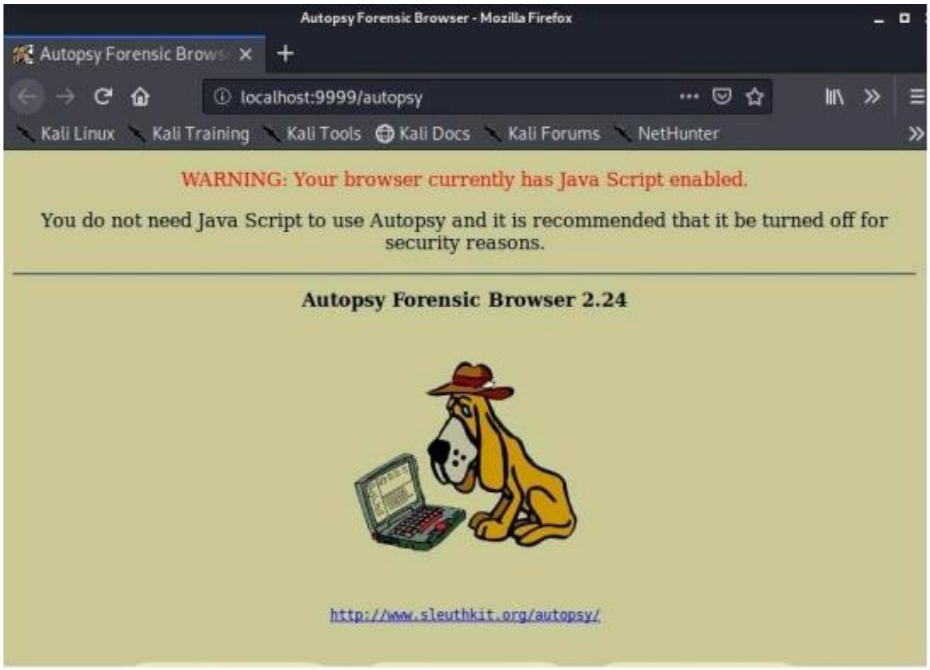
Open Autopsy



When you select autopsy, it will open a prompt where you see a program information, the version number listed as 2.24 with the path to the Evidence Locker folder as /var/lib/autopsy and an address http://localhost:9999/autopsy to open it on a web browser.



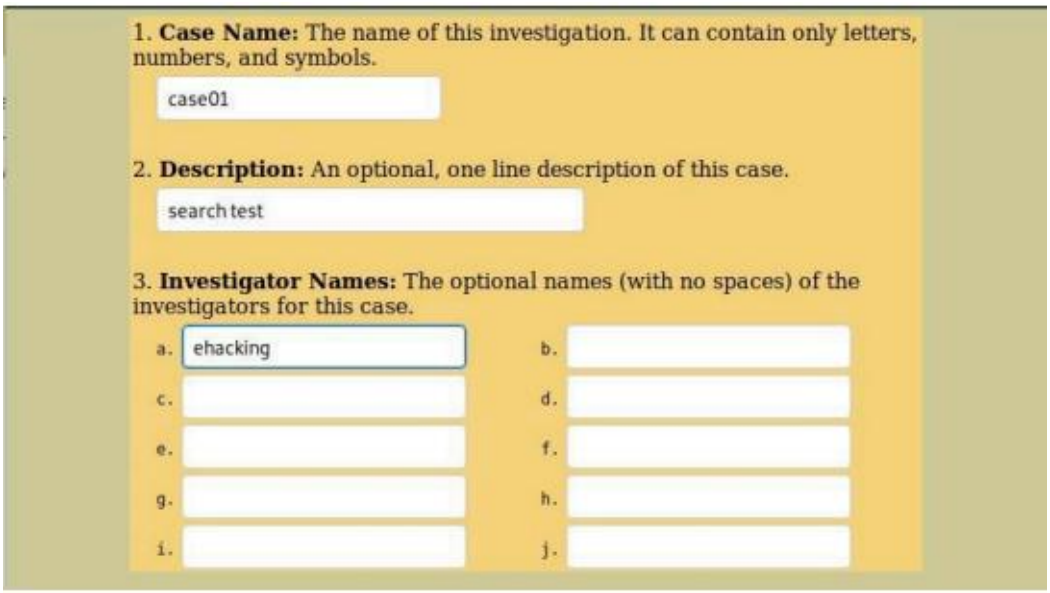
Click on that link and open it in your Kali web browser, you will be redirected to the home page of autopsy. This tool is running on our local web server accessing the port 9999.



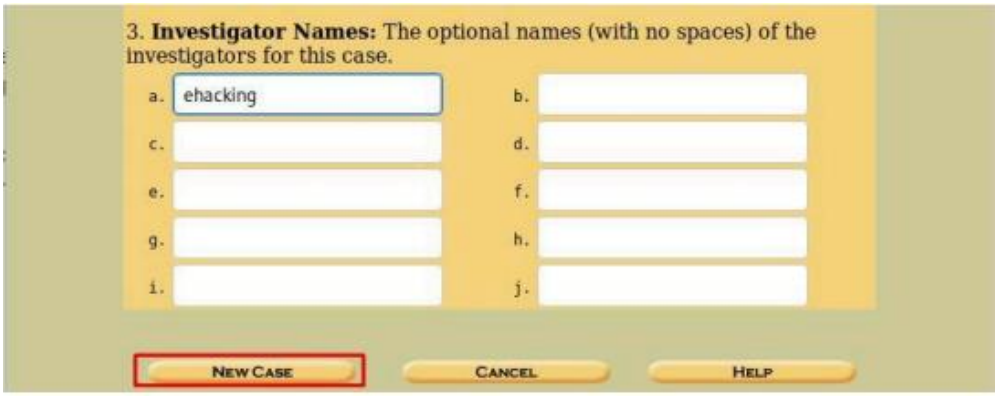
Create a New Case There will be three options on the home page: ‘OPEN CASE’, NEW CASE’, ‘HELP’

For forensic investigation, we need to create a new case and arrange all the information and evidences. Select ‘NEW CASE’

It will direct you to a page where you have been asked to add case name, description and investigator names. Note that you can add more than one investigator name because in these scenarios usually a team of forensic investigators work on a single case.



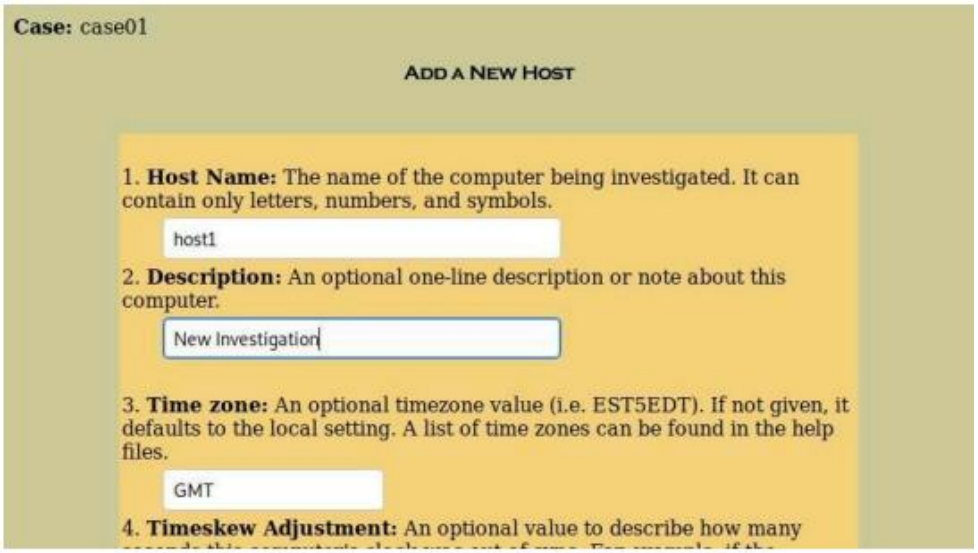
After adding all the required information, select ‘NEW CASE’

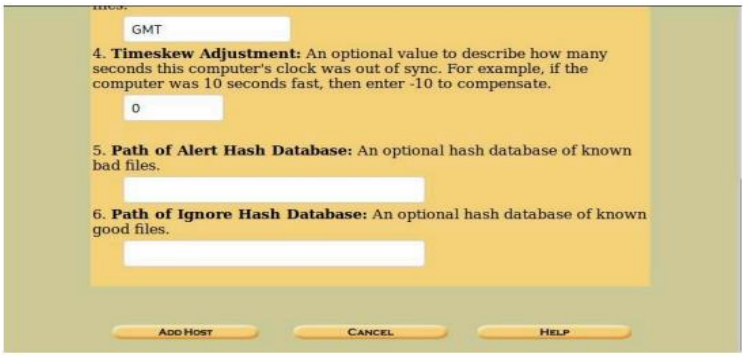


This simply showing us the name of the case, the destination where it will be stored i.e. /var/lib/autopsy/case01/, and the destination where its configuration file will be stored i.e. /var/lib/autopsy/case01/case.aut

Select ‘ADD HOST’ option below

Now you will be asked to enter the name of the computer you are investigating and the description of the investigation. After that it will ask you the time zone (leaving it blank will select the default setting), timeskew adjustments means a value in seconds to compensate for differences in time, path of alert hash means a path to the created database of bad hashes and a path of ignore hash database means specifying a path to the database of good hashes. Select ‘ADD HOST’ to continue.





Select ‘ADD IMAGE’ here.

Creating a Image File

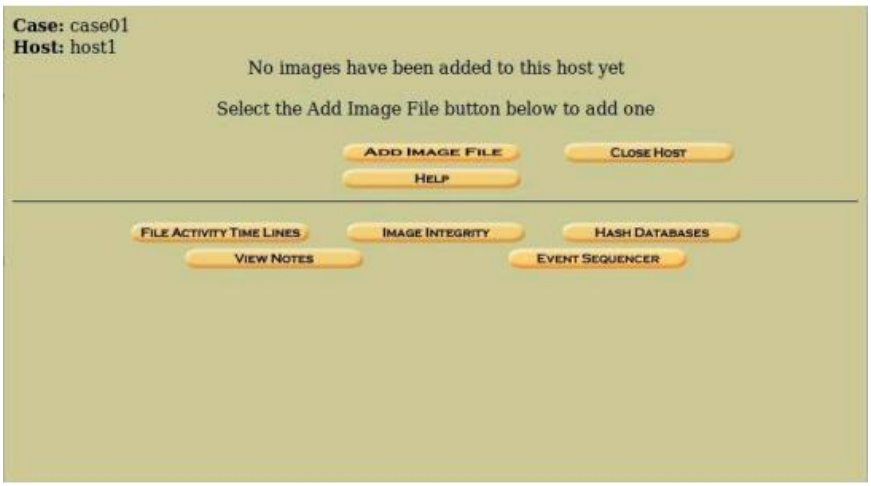
We need to import an image file of the system we want to investigate. Creating this image file is the first step of forensic investigation. The reason for doing this is analysis cannot be conducting on an original storage device. A disk Image can be defined as a file that stores the contents and structure of a data storage device such as a hard drive, CD drive, phone, tablet, RAM, or USB. This image file can be taken locally or remotely.

There are several ways to get the image file. You can get this by different tools such as FTK imager or guymager. Or you can use CLI to acquire your image by using dd (disk-to-disk) command:

dd if=/dev/sda of=ehacking.img

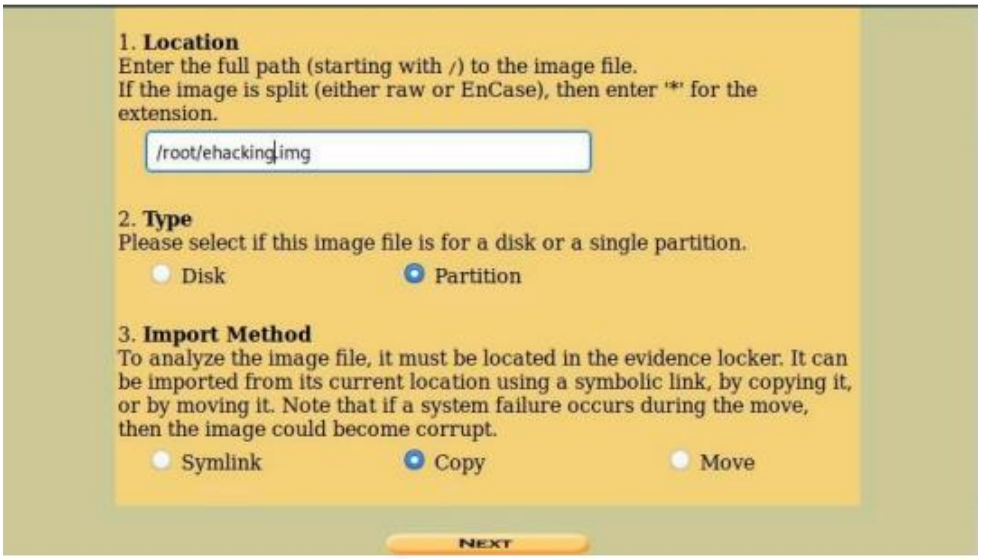
Where /dev/sda is the source and ehacking.img is the destination file.

Once you get an image file, select ‘ADD IMAGE’ option here

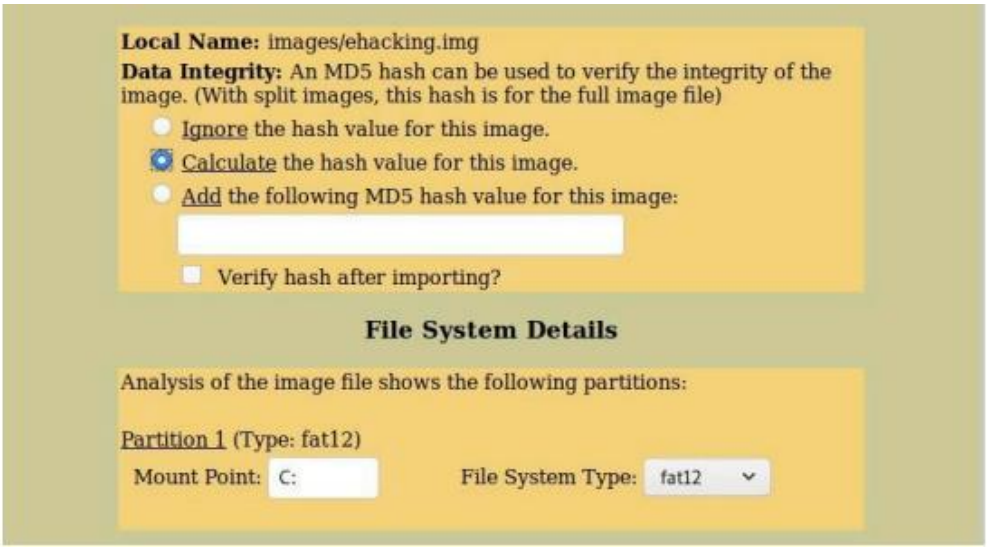


Import the image to autopsy by specifying the location of the file and selecting the type whether it is Disk or Partition.

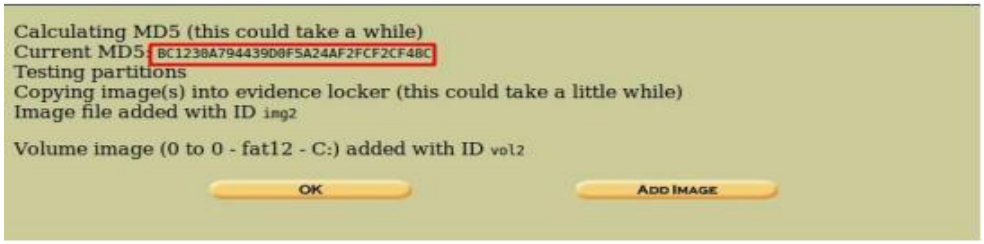
Select the import method ‘Copy’ to copy it into the evidence locker and click on ‘NEXT’



To maintain the integrity of the image file we must calculate its Hash value. It is important to calculate the Hash so that we may be able to prove that the file has not been tampered.



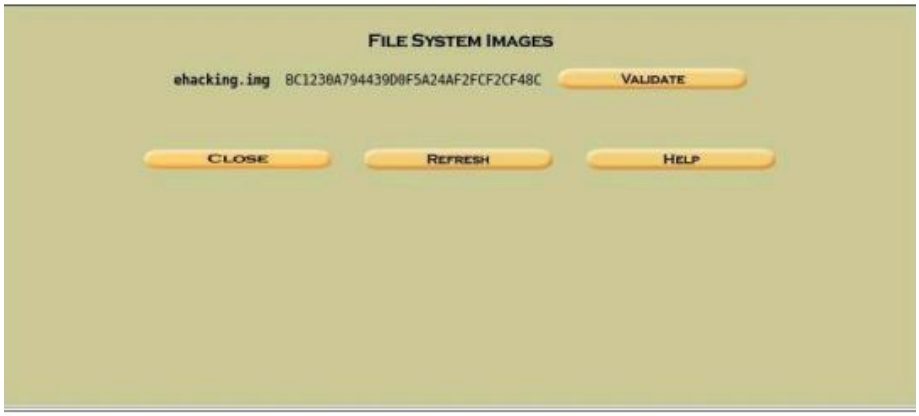
This showing the hash value of the image file and links the image into the evidence locker. Select ok to continue.



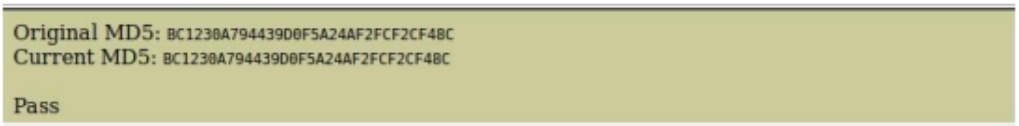
The Case Management Prompt

Now we have successfully imported the file for investigation. Let's check the integrity by selecting an option 'IMAGE INTEGRITY'.

This showing the name and the hash value of the file. Select 'VALIDATE'.



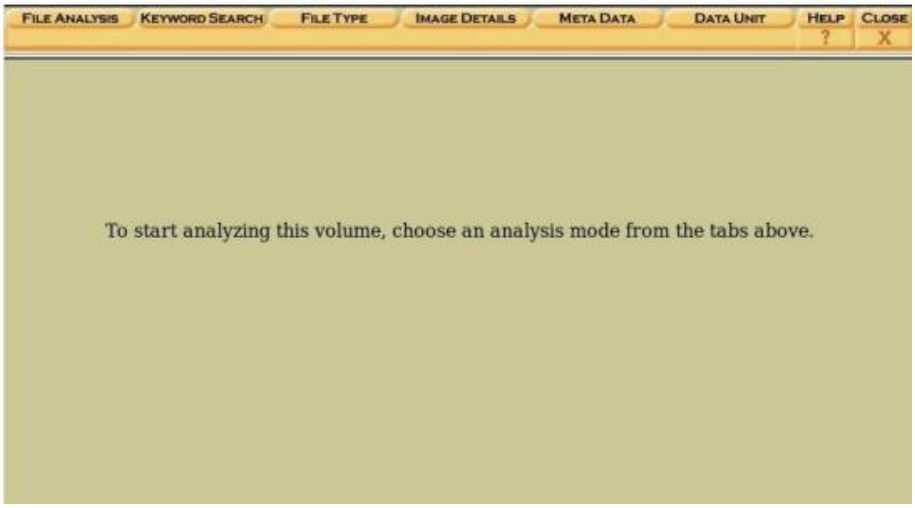
The validation is successful, displaying the same MD5 hashes in the bottom.



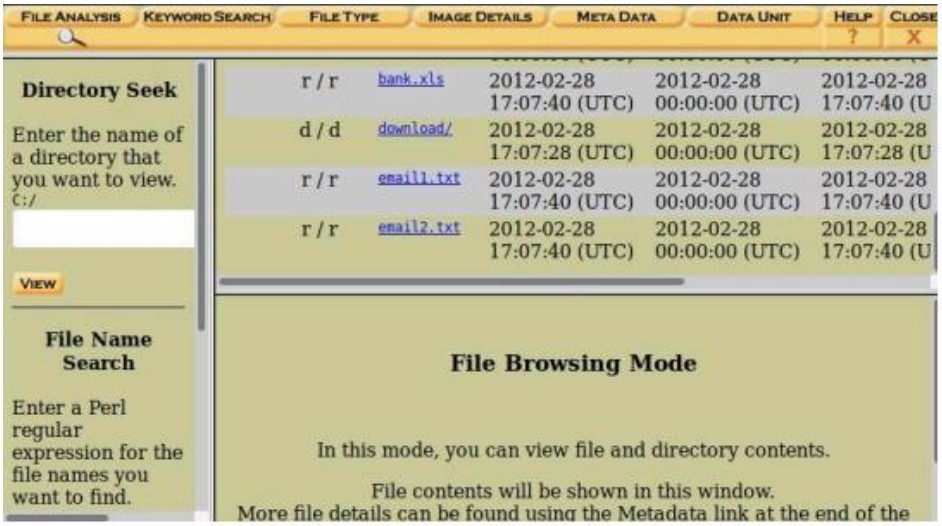
File Analysis

Let's click on 'ANALYZE'.

It will ask which type of analysis I want. Select ‘FILE ANALYSIS’.



It gives me the list of files and directories that are inside in this file. From here you can analyze the content of the target image file and conduct the required investigation.



In this article we have learned how to use a forensic tool Autopsy to investigate an image file and analyze the contents inside that file. We also calculated the hash value of the image file so that in future if there is a need to prove the integrity of the image file you can easily validate it by matching the hash values to maintain evidence integrity.

Conclusion : We have successfully studied AUTOPSY -Digital Forensics Investigation using Autopsy In Kali Linux.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 03	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To study the steps for hiding and extract any text file behind an image file/ Audio file using Command Prompt.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 03

Aim : To study the steps for hiding and extract any text file behind an image file/ Audio file using Command Prompt.

Theory :

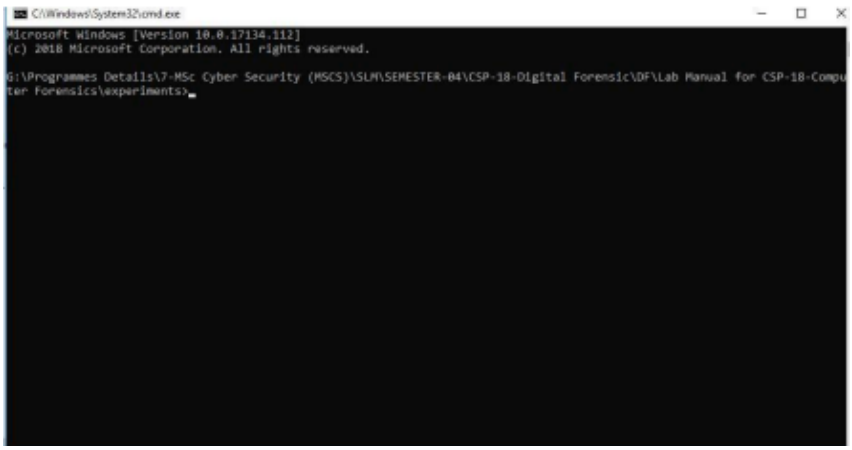
Any file like .rar .jpg .txt or any file can be merged inside another file. In a simple way, we shall learn how to hide a text file inside an image file using the Command Prompt.

How to Hide the FILE?

Suppose you have to hide a text file “A.txt” with the image file “B.jpg” and combine them in a new file as “C.jpg”.Where “C.jpg” is our output file which contains the text hidden in the image file.

Follow the steps:

1. copy the file,u need to hide, to desktop(for our tutorial let us assume the file to be "A.txt")
2. copy the image, within which you need to hide the file, to desktop (let it be "B.jpg")
3. now open the cmd: >ctrl+r >type: cmd and hit enter



4. in cmd first type the code as follows:

   >cd desktop

   NOTE: this code is for assigning the location on cmd to desktop

5. Now type the following code:

   > copy /b B.jpg + A.txt C.jpg

```
G:\Programmes Details\7-MSc Cyber Security (MSCS)\SLM\SEMESTER-04\CSP-18-Digital Forensic\DF\Lab Manual for CSP-18-Co
ter Forensics\experiments>copy /b B.jpg + A.txt  C.jpg
```

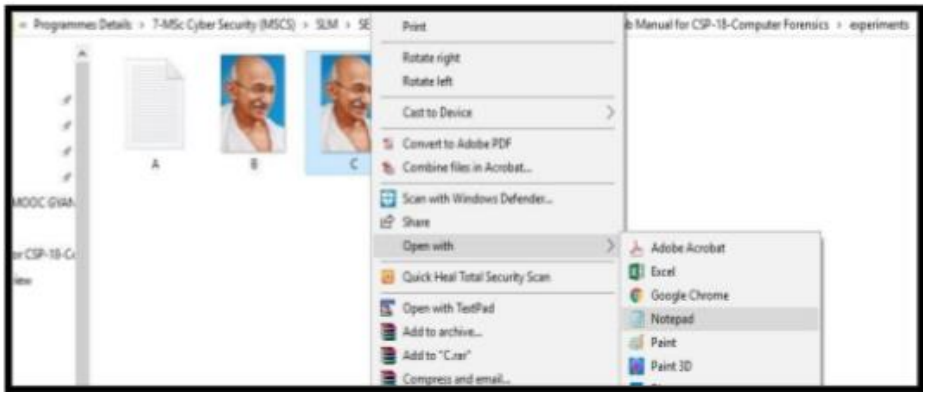
Syntax: copy /b Name-of-file-containing-text-you-want-to-hide.txt + Name-of-initialimage.jpg Resulting-image-name.jpg

```
G:\Programmes Details\7-MSc Cyber Security (MSCS)\SLM\SEMESTER-04\CSP-18-Digital Forensic\DF\Lab Manual fo
ter Forensics\experiments>copy /b B.jpg + A.txt  C.jpg
B.jpg
A.txt
       1 file(s) copied.

G:\Programmes Details\7-MSc Cyber Security (MSCS)\SLM\SEMESTER-04\CSP-18-Digital Forensic\DF\Lab Manual fo
ter Forensics\experiments>
```

"C.jpg" is the output image inside this out image our file is hidden

How to retrieve the file?

1. locate C.jpg file from where you want to retrieve text data
2. Right-click and open with notepad



Done! Successfully opened! In the last of the notepad, you'll find the content of the text file.

Hide A Message Into Image:

Open Run command window by pressing **win + r.**

Open command prompt by typing **cmd** and press **OK**

Enter the directory where you have your files. Then type the command :

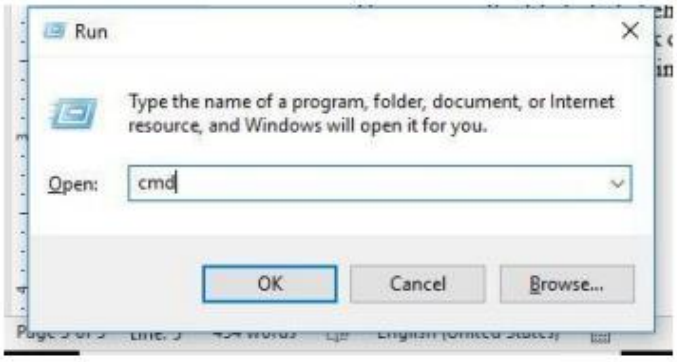
echo "Your Message">>"image.jpg"

Now the message is successfully hidden in the image file.

To view the message: Open with Notepad, at last, you'll find the Your Message

Another Method

1. Open Run command window by pressing **win + r.**
2. Open command prompt by typing **cmd** and press **OK**



3. Enter the directory where you have your files.
4. Then type the command :

   >> copy /b B.jpg + A.rar C.jpg

Here a.rar is the file to hide behind the image file (b.jpg) and the output file is **c.jpg**.

To view the RAR file: right-click on the output image (here, c.jpg) and open with WinRAR. You'll find the file inside the image.

Hide File and text behind Audio File

Firstly get hold of a sound file you want to hide the data in (example sound.mp3), then gather all your files you want to hide and put them in a ZIP (example secret.zip).

Our chosen Sound and zip file:

Windows 7/10:

Shift+right click in the folder containing the files will open the command prompt in that directory Windows: Open command prompt (start->run cmd), then use cd to get to the folder where the files are stored.

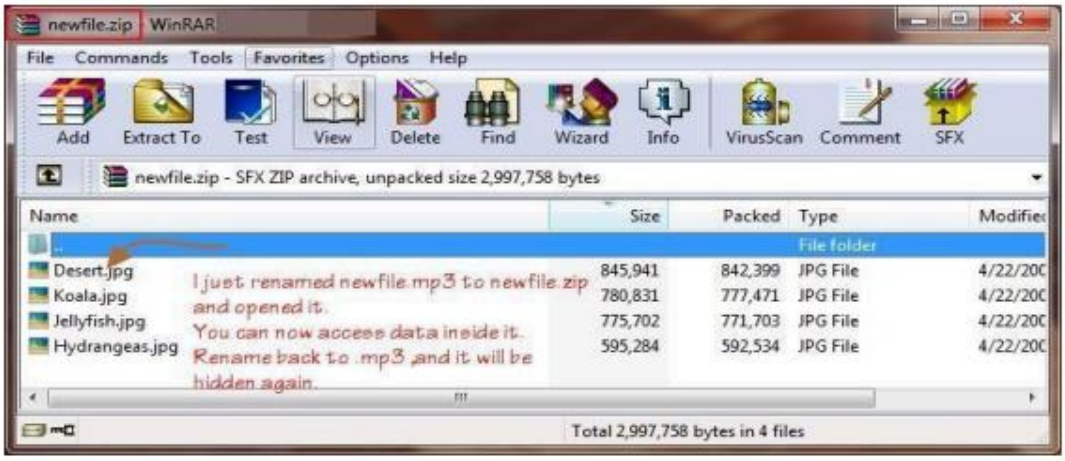
Linux: You know what to do, open terminal and move to the directory containing files. We now need to merge these files together, but we want to use a binary merge to keep the two files intact. With Windows copy command this uses the /B switch. (Binary Data)

Windows Code: copy /b secret.zip + sound.mp3 newfile.mp3 Linux

Code:

cat sound.mp3 secret.zip > newfile.mp3

You should now have gained a new file called newfile.mp3. This should look identical to the sound you started with when opened with a media player, but with a secret payload hidden within. Here is the example sound containing a ZIP:



The two simplest ways to get your data back out of these files is to either change the extension from .mp3 to .zip or to open your chosen ZIP program and open newfile.mp3 within that. You should now be presented with your original files.

Conclusion : We have successfully studied the steps for hiding and extract any text file behind an image file/ Audio file using Command Prompt.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 04	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To perform Network forensics using network miner.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 04

Aim : To perform Network forensics using network miner.

Theory :

NetworkMiner

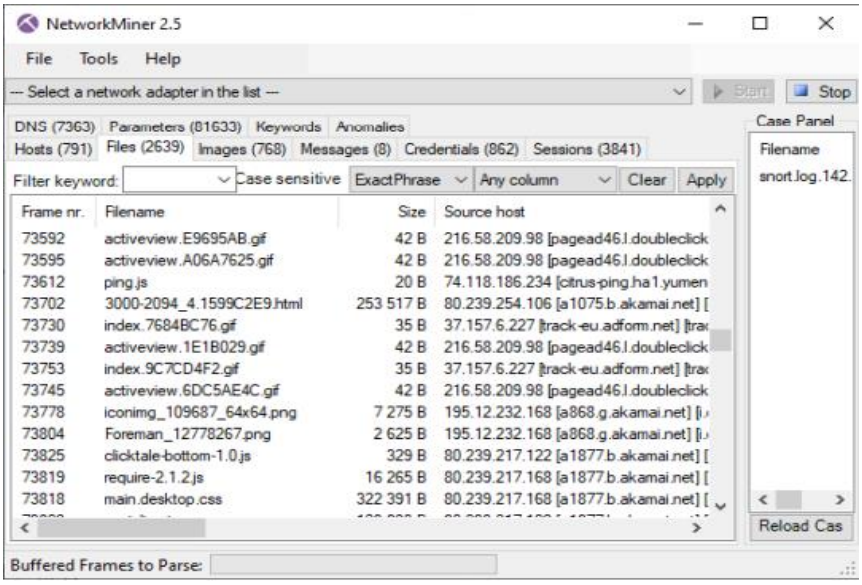
NetworkMiner is an open source network forensics tool that extracts artifacts, such as files, images, emails and passwords, from captured network traffic in PCAP files. NetworkMiner can also be used to capture live network traffic by sniffing a network interface. Detailed information about each IP address in the analyzed network traffic is aggregated to a network host inventory, which can be used for passive asset discovery as well as to get an overview of which devices that are communicating. NetworkMiner is primarily designed to run in Windows, but can also be used in Linux.

Steps :

1. Download and install network miner software in windows 10.
2. Download sample pcap file from internet. (https://forensicscontest.com/contest05/infected.pcap Here is the network capture file for this puzzle)
3. Run network miner as administrator.
4. Go to file menu and open pcap file.

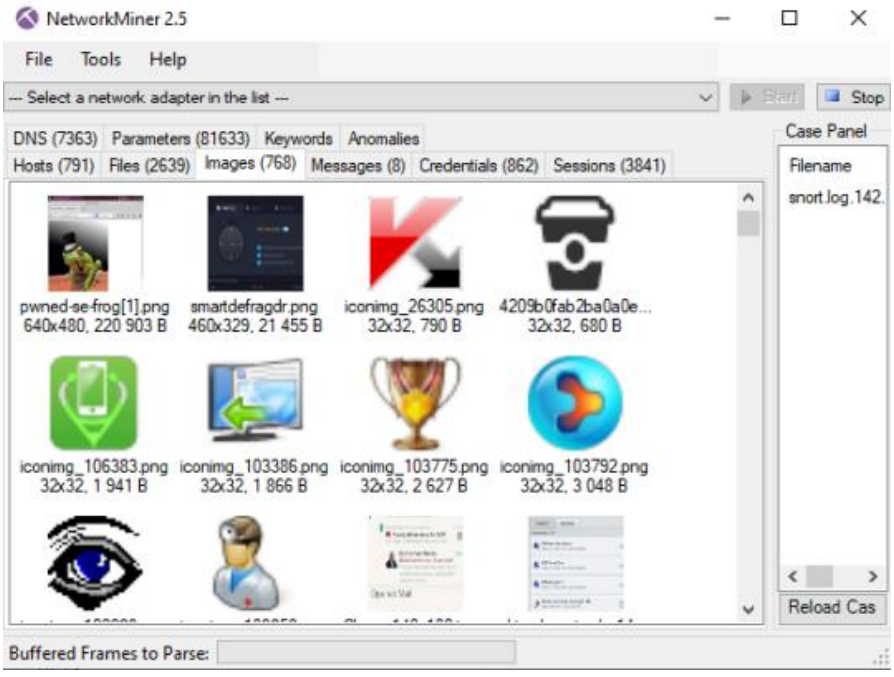
Extracting Files from PCAP Files

Many users turn to NetworkMiner when it comes to extracting artifacts, such as files or credentials from pcap files. You can solve such tasks with Wireshark too, but NetworkMiner will save you time and spare you some tedious manual work. In fact, NetworkMiner automatically extracts files from protocols like FTP, TFTP, HTTP, HTTP/2, SMB, SMB2, SMTP, POP3, and IMAP as soon as a pcap file is opened.



Files extracted by NetworkMiner 2.5

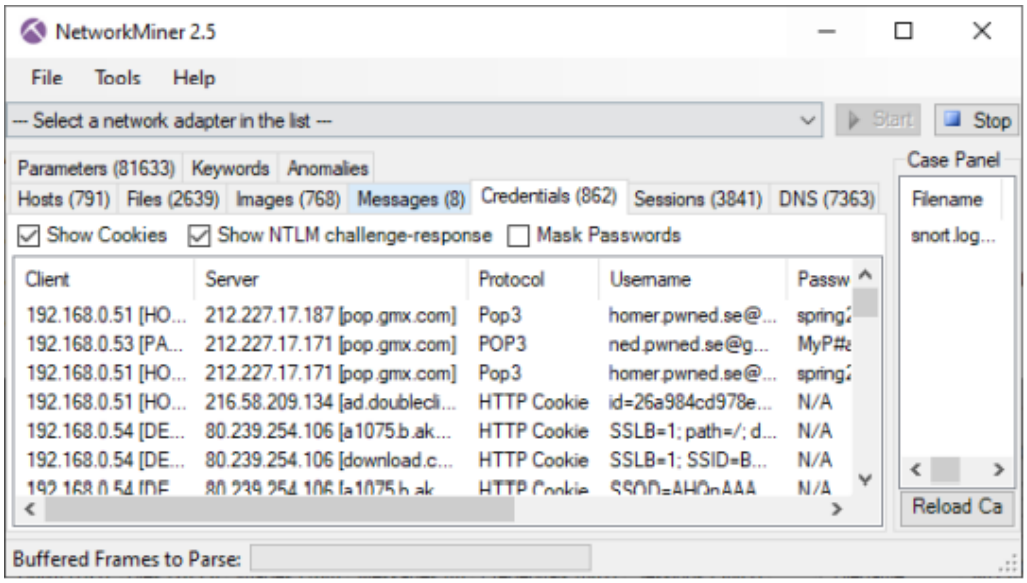
Extracted files that are recognized as images are also shown as thumbnails on the images tab. This image list can give a quick overview of what is going on in the capture file.



Thumbnails of extracted images in NetworkMiner 2.5

Usernames and Passwords

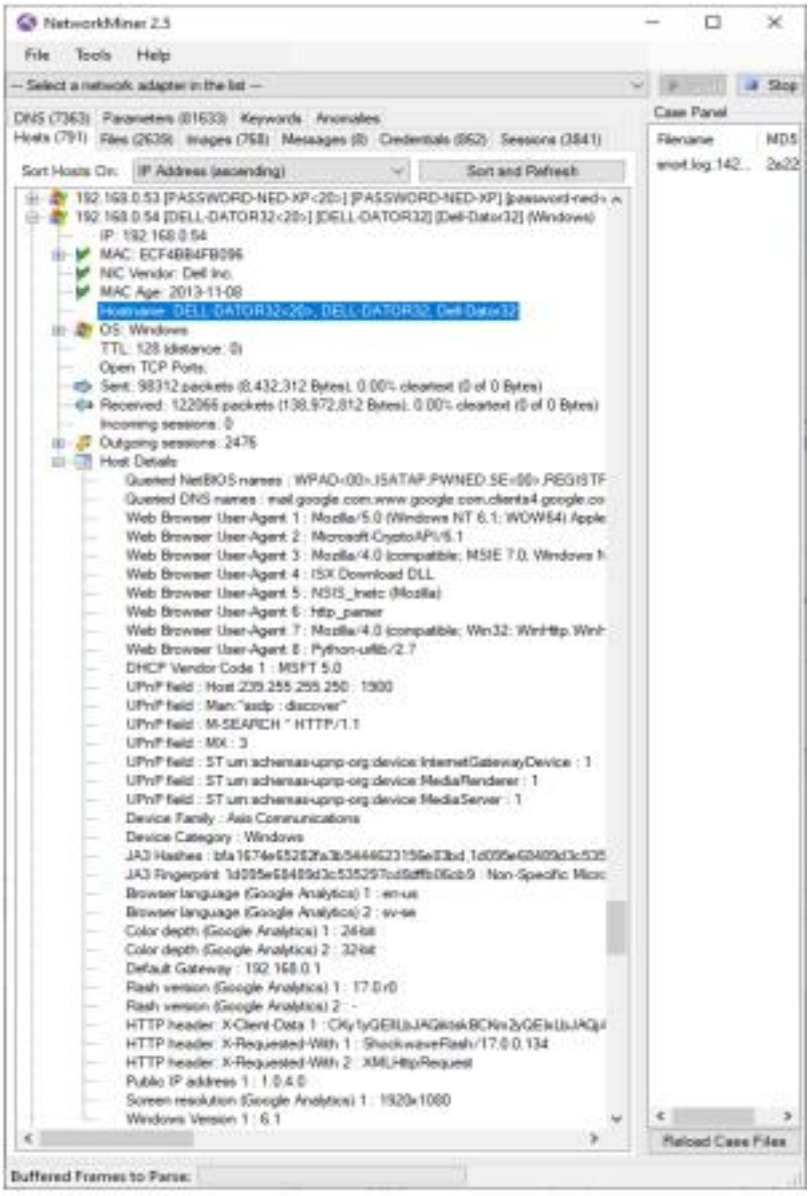
User credentials, such as usernames, passwords and hashes that NetworkMiner detects are all placed under the “Credentials” tab. The protocols and data structures from which NetworkMiner can extract credentials include FTP, HTTP cookies, HTTP POST requests, IMAP, Kerberos hashes, MS SQL, NTLM hashes, POP3, RDP cookies, SMTP, SOCKS and a few more.



Usernames, passwords, hashes and cookies extracted by NetworkMiner

Passive Host Inventory.

Another popular feature in NetworkMiner is the “Hosts” tab, which provides an overview of which devices have been observed in the loaded capture files. NetworkMiner aggregates data based on IP address in the Hosts tab, which is useful if you want to create a host inventory list without actively scanning a network or if you want to learn more about a particular IP address.



Details about 192.168.0.54 extracted by NetworkMiner

The Hosts tab is great if you want to find out what hostname(s) an IP address has since it doesn't just rely on captured DNS traffic to perform passive hostname resolution. NetworkMiner also extracts and aggregates hostname info from the CIFS Browser Protocol, DHCP, HP Switch Protocol, HTTP/2 authority headers, HTTP host headers, HTTP User-Agent strings, Kerberos, NetBIOS Datagram Service, NetBIOS Name Service, NTLMSSP, TLS SNI, X.509 certificates, and a few additional protocols and data structures.

The Hosts tab is also often used in order to find out which operating system a host is running as well as details like what web browser User-Agents or open SMB file shares it has.

Conclusion : We have successfully studied To perform Network forensics using network miner.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 05	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- Recycle Bin Forensics.	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 05

Aim : Recycle Bin Forensics.

Theory :

Recycle Bin Forensic

Lets begin with a small history of recycle bin.

Windows recycle bin was first introduced in windows 95 which contains files that have been deleted by users. For instance when a user deletes the file the file directly goes into recycle bin without deleting it permanently. This is the default behaviour of windows.

Besides this you can also delete permanently by pressing the shift key it will delete the file without moving into recycle bin. Recycle bin can hold artifacts which are consider valuable source for digital evidence.

Different version of windows have different recycle bin file name and location. In windows 95/98/me there was single file called info2.

C:\RECYCLED\INFO2

It contains metadata about each deleted files like it's original path, file size and date/time when it was deleted.

In windows NT/2000/XP the INFO2 file was still present. But now in recycler folder it's with a sub folder called SID(Security Identifier)

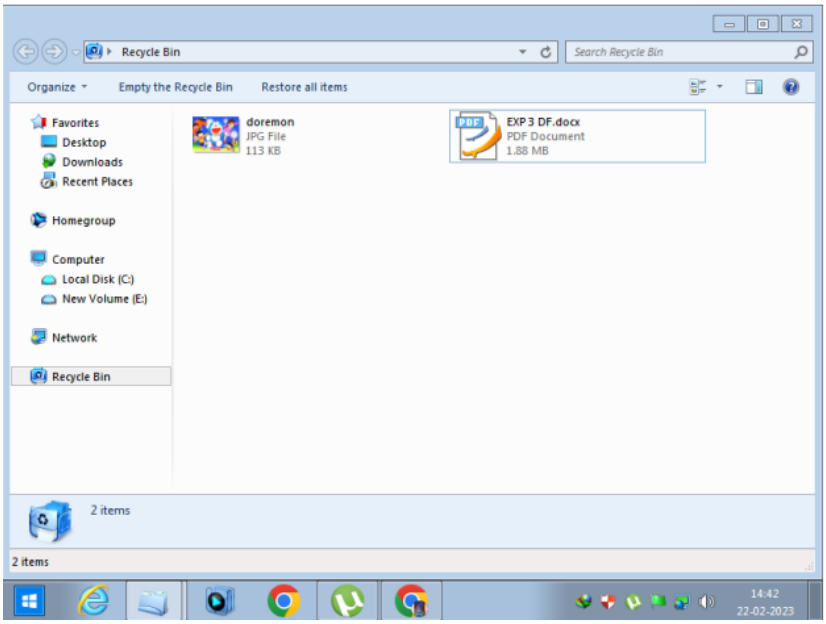
C:\RECYCLER\SID*\INFO2

So whenever user deletes a file it creates SID file. Each SID subfolder contains its own info2 file

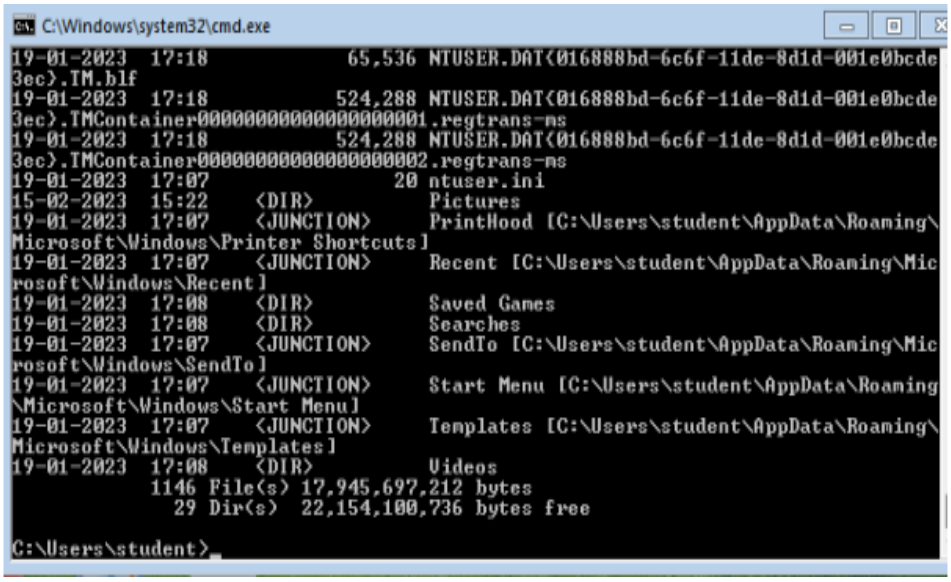
From windows Vista and later the recycler path was renamed as C:\$Recycle.Bin\SID*\$I and C:\$Recyecle.Bin\SID*\$R where $I contains meta data of the deleted file and $R contains actual deleted file. Now it discards the need of INFO2 which was in older versions of windows.

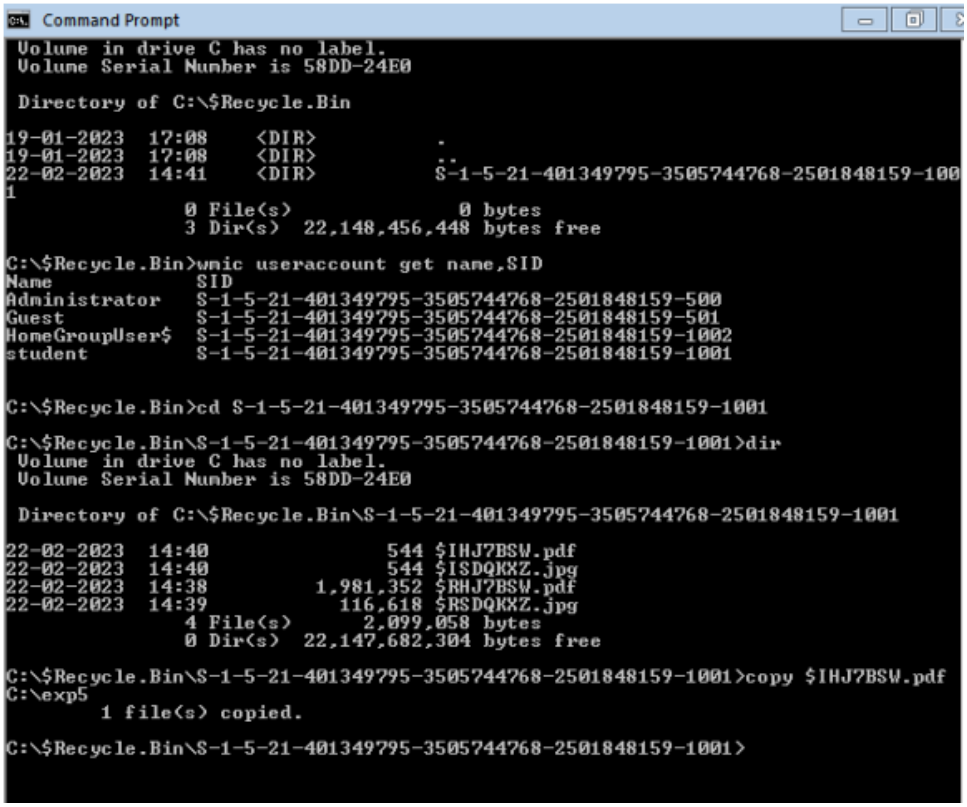
Lets see how it works :

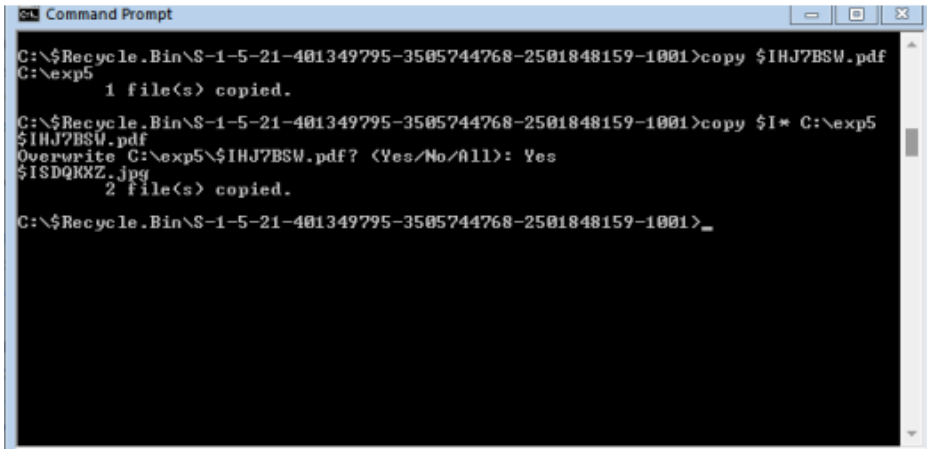
- Now here I have deleted two files from my desktop which is in now recycle bin here you can see Original location, Date Deleted ,Size, File Type and Date Modified. This all are metadata which is consider as $I as we discussed previously.



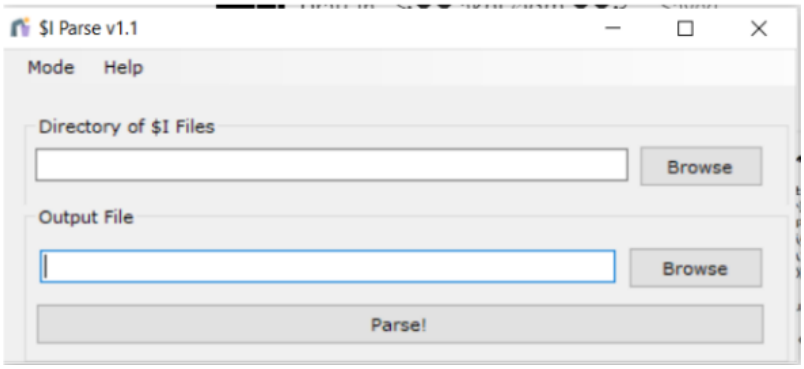
- Now let's open Command Prompt (CMD) and go to your drive C: and type the following command dir /a as shown below it will display the files in that directory including the hidden once.

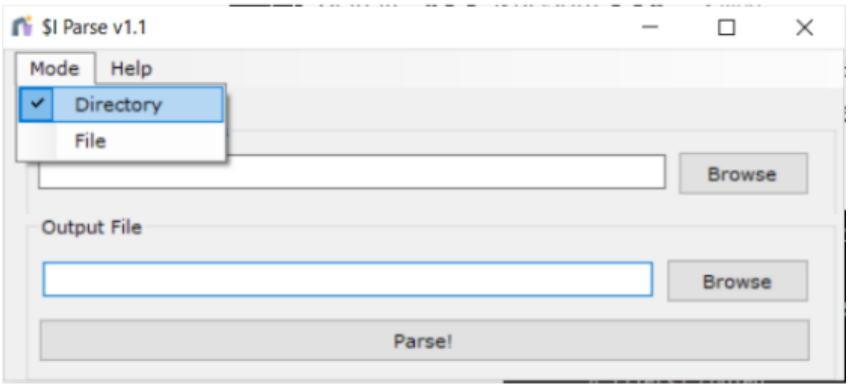






- Once all the files are copied we shall start the tool $I Parse. Click on Mode select Directory where the $I files are kept in my case I have moved on to desktop so i will give the desktop path as mentioned below it will automatically take it from desktop.





After provide the output where you want to get the output of the file i have provided the output of my desktop. And then click on Parse.

Once the parsing is done it will provide the output in .tsv format which can be opened in excel.

And this will be the final output as you can see we got the required metadata in a readable format.

Deleted Date	File Name	File Size (bytes)	Version
08-22-2021 05:26:09 UTC	C:\Users\Akhil\Downloads\WhatsApp Image 2021-	31440	Windows 10
09-07-2021 06:46:28 UTC	C:\Users\Akhil\Desktop\Tree.jfif	5406	Windows 10
09-07-2021 06:46:30 UTC	C:\Users\Akhil\Desktop\Splunkfun.pdf	12319438	Windows 10
09-07-2021 07:40:57 UTC	C:\Users\Akhil\Desktop\$R2UW2VL.jfif	5406	Windows 10

Conclusion : We have successfully studied Recycle Bin Forensics.

JAYAWANTI BABU FOUNDATION'S
METROPOLITAN INSTITUTE OF TECHNOLOGY AND MANAGEMENT (MITM)
AT POST-SUKALWAD, TAL-MALVAN, DIST-SINDHUDURG, PIN-416534

Department of Computer Engineering

Experiment No:- 06	
Subject : Digital Forensic	
Semester :- VIII	**Roll No:-**
Name of the Student :-	
Aim :- To perform data carving using following open source tools 1)Foremost 2)Scalpel	

Submission Details	
Given Date	**Submission Date**

Practical Conduction [03]	
Attendance [02]	
Result [03]	
Oral [02]	
Total [10]	

Faculty Signature with Date:-

Experiment No. 06

Aim : To perform data carving using following open source tools

1)Foremost

2)Scalpel

Theory :

Data carving :

Data/file Carving is a process to recover or reconstruct the deleted or formatted files in the computer. It is the process of searching a file in a data stream and carve out deleted files.

This process is very important in Digital Forensics, as the forensics expert has to investigate all the system files and they also have to check for any deleted or formatted files for further investigation. To recover these deleted files the forensics expert uses certain software and programs to carve out these files.

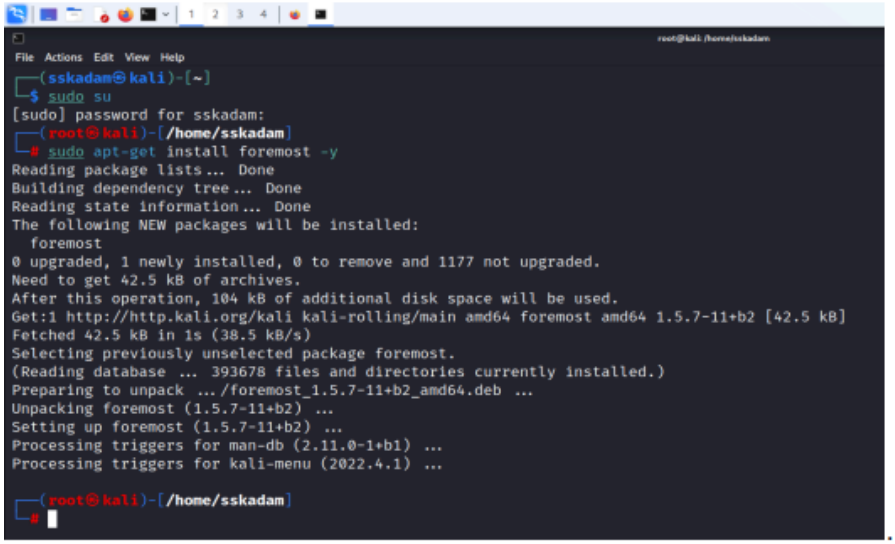
Data carving is one of the best methods for recovering the files if the entire directory is corrupt or missing. It's especially used in cybercrimes to collect and restore evidence at the crime.

1) Foremost:

   Foremost is a program that is used to carve data from disk image files, it is an extremely useful tool and very easy to use. Foremost can carve data out of incomplete disk images as well. We have used Kali Linux but if you want you can install Foremost on pretty much any distro of Linux.We will use captured image(.dd) file of usb drive as input file carving will be performed on this file.

Steps-

1. install foremost on kali linux terminal. .



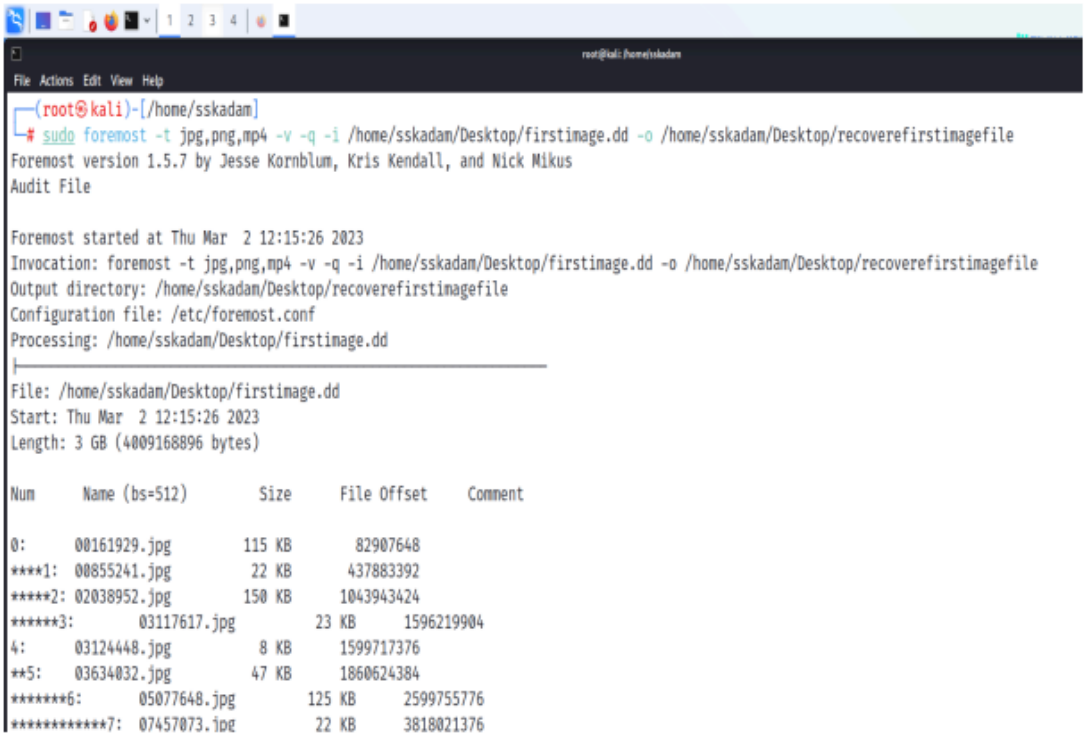
Use command following command

To break this down **"-t"** is setting the file types we want to carve out of the disk image, here those are .jpeg and .png.

"-i" is specifying the input file, the **"/home/sskadam/Desktop/firstimage.dd"** that is placed on the desktop.

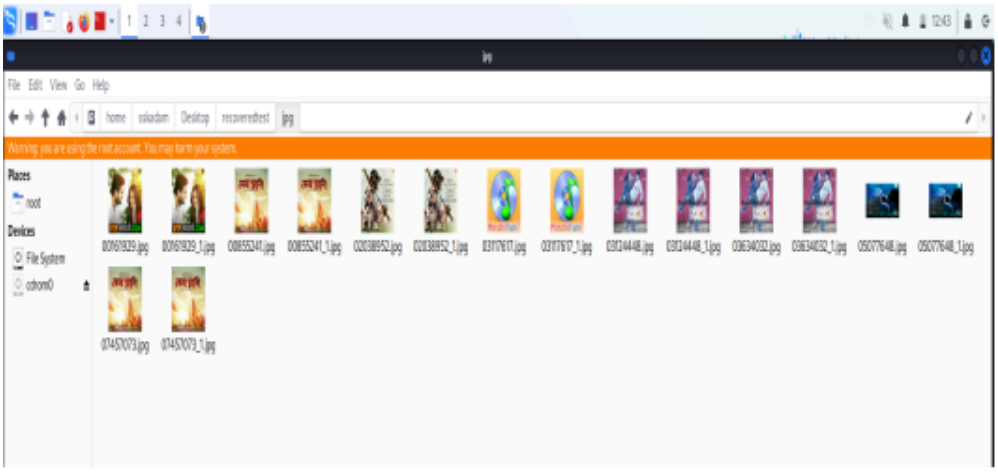
"-o" is telling Foremost where we want the carved files to be stored, for that we have the **"/home/sskadam/Desktop/recoverfirstimagefile"** folder on the desktop that we made earlier.

"-v" is to tell Foremost to log all the messages that appear on screen as the file is being carved into a text file in the output folder **(recoverfirstimagefile)** as an audit report.



That's all it takes for Foremost to start digging into the disk image. The process looks like this.

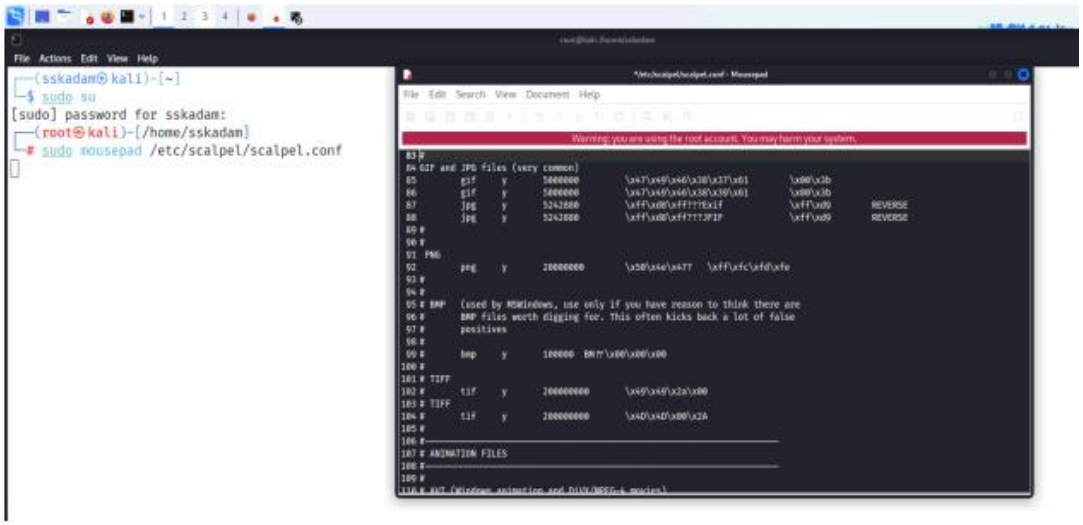
Now you can check **‘’ recoverfirstimagefile”** folder present on desktop. To open this folder you need to right click on that then choose **“Open as root”** option. You will see an audit report and Sub folders which will be named by the file types we invoked Foremost to carve for us. Open sub folders to see carved files.



2) Scalpel:

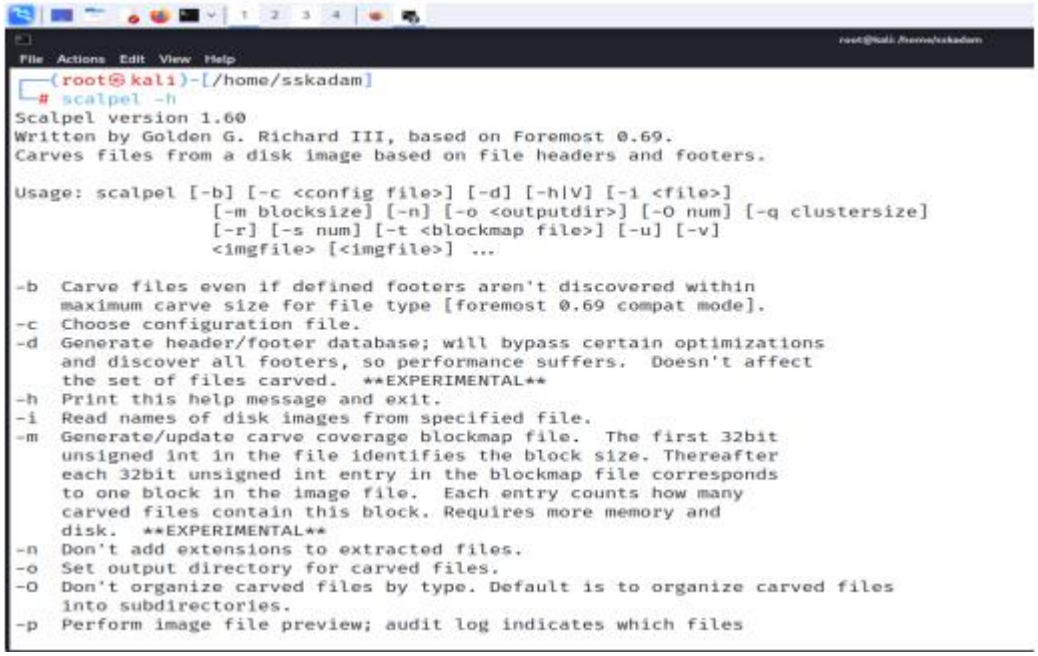
It is a very well-known tool for file carving and a reprogrammed version of the “foremost”. Scalpel is available for both Windows and Linux. In Kali Linux, scalpel comes pre-installed and can be directly used from the terminal by typing scalpel.

For using scalpel in Linux we have to change the configuration file of the scalpel which is located at **/etc/scalpel/scalpel.conf** and remove the hashes from the line where our desired file type is written.



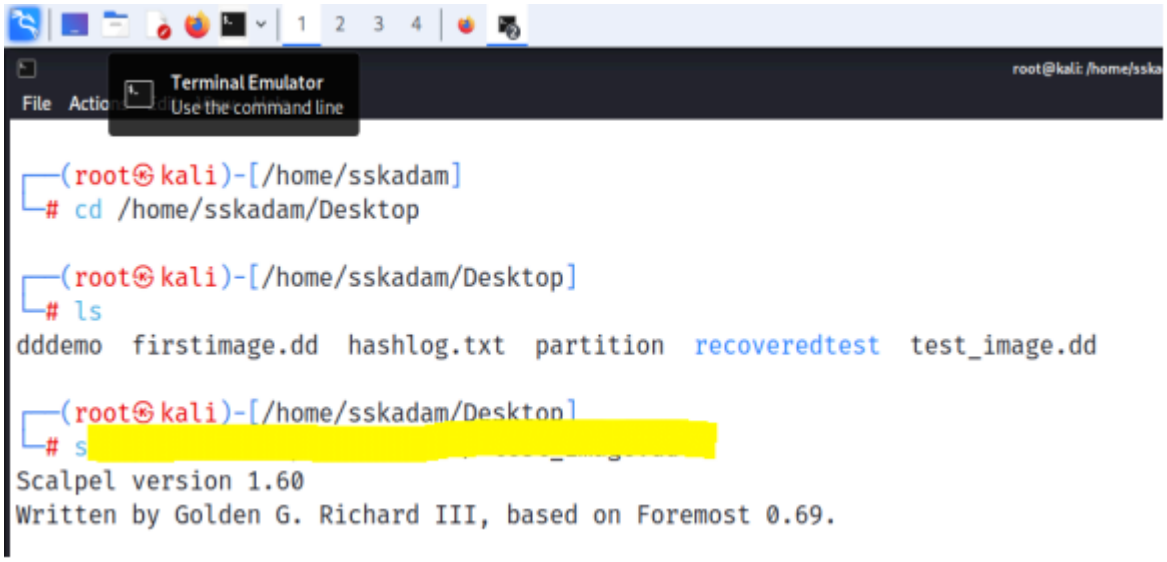
For eg: If I want to find a JPG file I will remove has or Uncomment the line where JPG is written. Save changes.

Then you can check all the options by typing scalpel -h or just scalpel.



```
File  Actions  Edit  View  Help
┌──(root㉿kali)-[/home/sskadam]
└─# scalpel -h
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.
Carves files from a disk image based on file headers and footers.

Usage: scalpel [-b] [-c <config file>] [-d] [-h|V] [-i <file>]
               [-m blocksize] [-n] [-o <outputdir>] [-O num] [-q clustersize]
               [-r] [-s num] [-t <blockmap file>] [-u] [-v]
               <imgfile> [<imgfile>] ...

-b  Carve files even if defined footers aren't discovered within
    maximum carve size for file type [foremost 0.69 compat mode].
-c  Choose configuration file.
-d  Generate header/footer database; will bypass certain optimizations
    and discover all footers, so performance suffers.  Doesn't affect
    the set of files carved.  **EXPERIMENTAL**
-h  Print this help message and exit.
-i  Read names of disk images from specified file.
-m  Generate/update carve coverage blockmap file.  The first 32bit
    unsigned int in the file identifies the block size. Thereafter
    each 32bit unsigned int entry in the blockmap file corresponds
    to one block in the image file.  Each entry counts how many
    carved files contain this block. Requires more memory and
    disk.  **EXPERIMENTAL**
-n  Don't add extensions to extracted files.
-o  Set output directory for carved files.
-O  Don't organize carved files by type. Default is to organize carved files
    into subdirectories.
-p  Perform image file preview; audit log indicates which files
```

Use following highlighted command



```
File  Actio   Terminal Emulator
              Use the command line
                                                    root@kali: /home/sska
┌──(root㉿kali)-[/home/sskadam]
└─# cd /home/sskadam/Desktop

┌──(root㉿kali)-[/home/sskadam/Desktop]
└─# ls
dddemo  firstimage.dd  hashlog.txt  partition  recoveredtest  test_image.dd

┌──(root㉿kali)-[/home/sskadam/Desktop]
└─# s
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.
```

In this command -O is followed by name of folder in which scalpel will recover data. And next to folder name there is image file name on which we will perform data carving.

Now you can check **‘’ scalpelrecoverd”** folder present on desktop. To open this folder you need to right click on that then choose **“Open as root”** option.

You will see an audit report and Sub folders which will be named by the file types we invoked Foremost to carve for us. Open sub folders to see carved files.

Conclusion : We have successfully studied To perform data carving using following open source tools

1)Foremost

2)Scalpel