Q2) To implement RPC using (Server)

#include <iostream>

#include "../Example1/Example1.h"

// Server function.

void Output(

/\* [string][in] \*/ const char\* szOutput)

{

std::cout << szOutput << std::endl;

}

// Naive security callback.

RPC\_STATUS CALLBACK SecurityCallback(RPC\_IF\_HANDLE /\*hInterface\*/, void\* /\*pBindingHandle\*/)

{

return RPC\_S\_OK; // Always allow anyone.

}

int main()

{

RPC\_STATUS status;

// Uses the protocol combined with the endpoint for receiving

// remote procedure calls.

status = RpcServerUseProtseqEp(

reinterpret\_cast<unsigned char\*>("ncacn\_ip\_tcp"), // Use TCP/IP protocol.

RPC\_C\_PROTSEQ\_MAX\_REQS\_DEFAULT, // Backlog queue length for TCP/IP.

reinterpret\_cast<unsigned char\*>("4747"), // TCP/IP port to use.

NULL); // No security.

if (status)

exit(status);

// Registers the Example1 interface.

status = RpcServerRegisterIf2(

Example1\_v1\_0\_s\_ifspec, // Interface to register.

NULL, // Use the MIDL generated entry-point vector.

NULL, // Use the MIDL generated entry-point vector.

RPC\_IF\_ALLOW\_CALLBACKS\_WITH\_NO\_AUTH, // Forces use of security callback.

RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, // Use default number of concurrent calls.

(unsigned)-1, // Infinite max size of incoming data blocks.

SecurityCallback); // Naive security callback.

if (status)

exit(status);

// Start to listen for remote procedure calls for all registered interfaces.

// This call will not return until RpcMgmtStopServerListening is called.

status = RpcServerListen(

1, // Recommended minimum number of threads.

RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, // Recommended maximum number of threads.

FALSE); // Start listening now.

if (status)

exit(status);

}

// Memory allocation function for RPC.

// The runtime uses these two functions for allocating/deallocating

// enough memory to pass the string to the server.

void\* \_\_RPC\_USER midl\_user\_allocate(size\_t size)

{

return malloc(size);

}

// Memory deallocation function for RPC.

void \_\_RPC\_USER midl\_user\_free(void\* p)

{

free(p);

}