|  |  |
| --- | --- |
|  | **Wireless -GRVM Application** |
|  | Mateusz Siwoski A00758640  Robin Hsieh A00657820  German Villarreal A00839611  Vincent Lau A00758190 |

|  |
| --- |
| **[Implementation of the (BC) Protocol]** |
|  |



# Abstract

The Wireless-GRVM application is a wireless protocol that functions on serial. Using serial communication to adhere to the protocol introduces issues such as no actual established connection as well as security concerns. The results were as unexpected as usual, as the wireless medium has a lot of interference which brought about complications. Testing occurred on both wired and wireless and, as expected, this application is more reliable when it comes to wired connections.

Contents

[Abstract: 1](#_Toc373718446)

[Introduction: 3](#_Toc373718447)

[Description: 4](#_Toc373718448)

[Features: 5](#_Toc373718449)

[Issues Experienced: 6](#_Toc373718450)

[Gantt Chart: 7](#_Toc373718451)

[State Diagram: 9](#_Toc373718452)

[Improved: 9](#_Toc373718453)

[Original: 10](#_Toc373718454)

[Function Prototypes: 11](#_Toc373718455)

[ErrorCheck 11](#_Toc373718456)

[Main 11](#_Toc373718457)

[Packet 11](#_Toc373718458)

[Physical 11](#_Toc373718459)

[Presentation 11](#_Toc373718460)

[Session 11](#_Toc373718461)

[Transport 12](#_Toc373718462)

[Pseudocode: 12](#_Toc373718463)

[Conclusion 18](#_Toc373718464)

[Test Cases 19](#_Toc373718465)

[Figures 20](#_Toc373718466)

[Test 1: Initial Start Up (PASS) 20](#_Toc373718467)

[Test 2: Opening and Displaying a File (PASS) 20](#_Toc373718468)

[Test 3: Threading (FAIL) 21](#_Toc373718469)

[Test 4: Threading (PASS) 21](#_Toc373718470)

# Introduction

This is a Win32 application that implements a wireless protocol for transmitting text characters from one device to another and displays the information on a computer monitor. This program is fully event driven and will work as a half-duplex.

The protocol used for this program is the (Be Creative) Protocol and will be run on Windows 7. The application will have two states for transmission, Send and Receive. The program will be sending and receiving text files and will display the file upon receiving 5 bytes of data. To ensure correct information is being sent, the program uses the CRCITT

The designers/programmers are Mateusz Siwoski, Robin Hsieh, German Villarreal and Vincent Lau.

# Description

This program is implemented to communicate with other computers using a newly specified wireless protocol. By transmitting data through the serial port to a wireless modem; this program is able to send files to other connected computers. Computers connect to each other and will start listening for another computer; once another computer is found a connection is established and they are able to transmit files.

Users must connect to each other and specify the file to send. If a proper connection was established and the other computer is also strictly following our protocol the file will transmit flawlessly.

The protocol used to transmit files is more thoroughly covered in the *Features* section.

# Features

The features of this program are heavily based on the wireless protocol we have designed. Our program strictly follows the protocol and thus allows us to communicate with others through a half-duplex wireless connection. After establishing a connection and bidding for the line, the file to send is packetized and sent; up to a maximum of 5 packets are sent, the line is dropped by this sender, allowing the other computer to bid for the line and send its 5 packets. This pattern continues until they are both sending.

Our program handles all possible erroneous cases and treats them accordingly depending on the communication medium. When both computers try to send at the same time they are able to properly communicate and let one of them proceed to send first. Sent packets are also verified by the receiver using CRCITT to ensure no packets have arrived with no corrupt bits or out of order. If an error has occurred the erroneous packet must be resent.

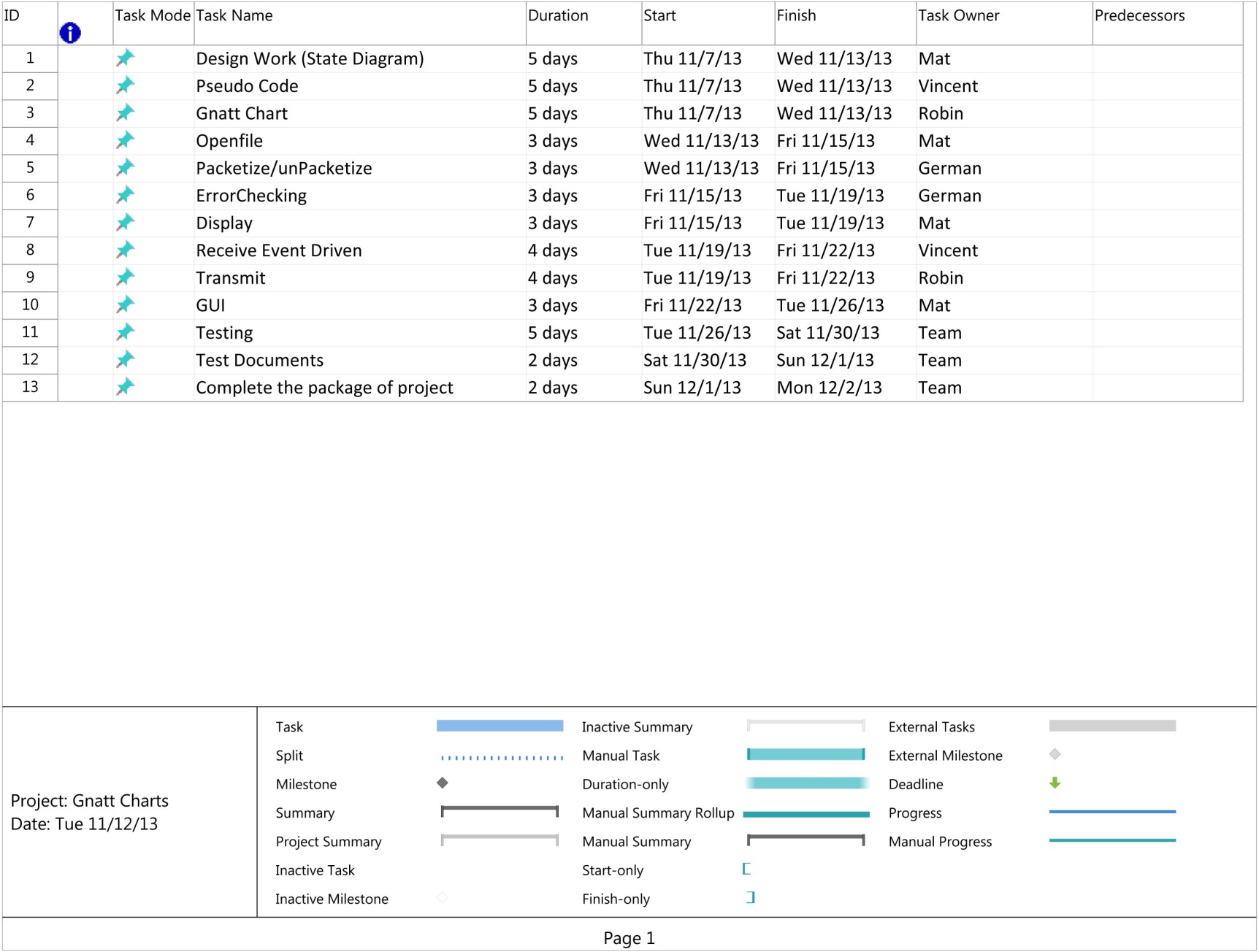
# Issues Experienced

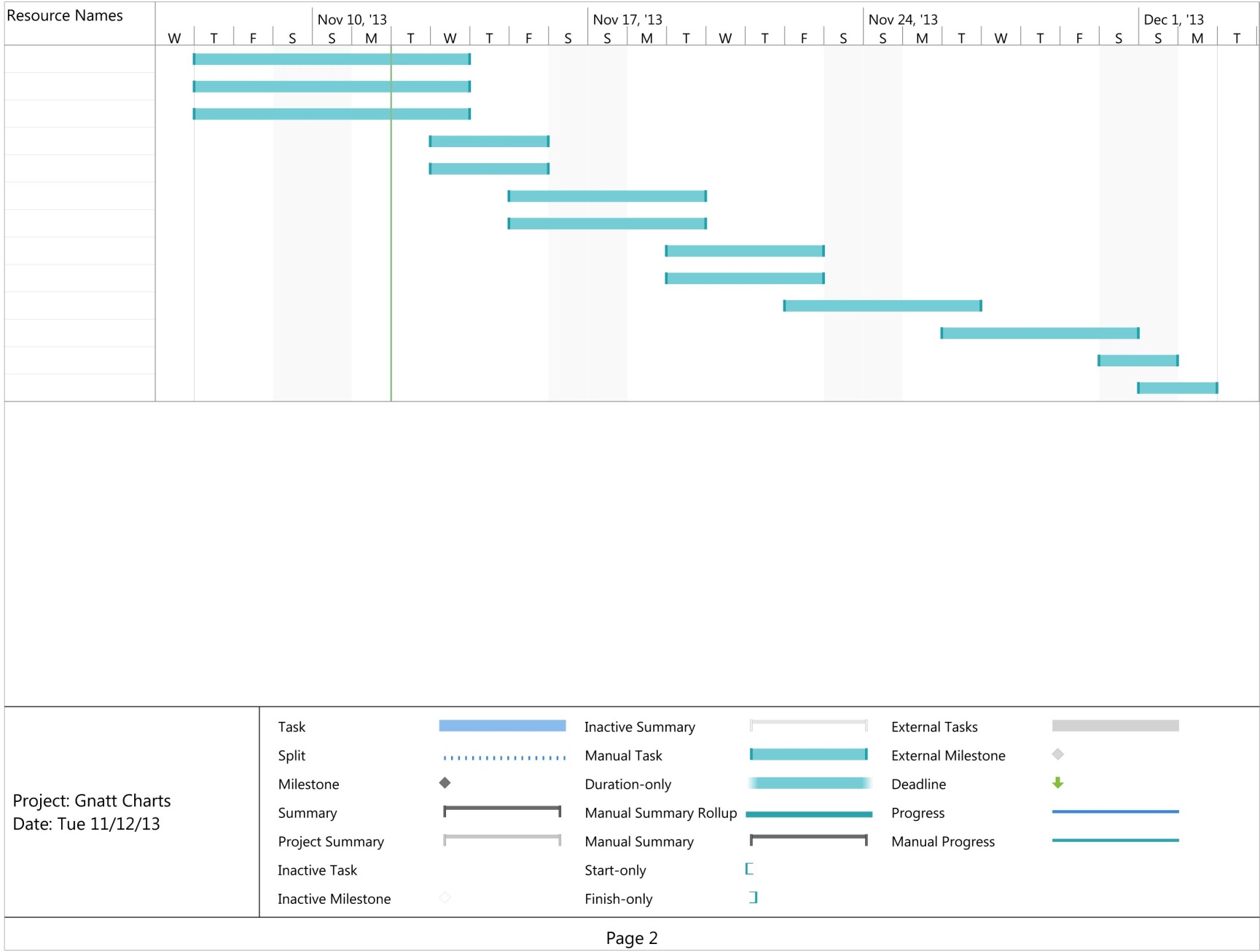
The major issue we have encountered has been corruption over the wireless connection. Packets do not seem to be fully sent and often arrive in with errors due to high interference. When testing through a wireless medium the throughput drops drastically; however, when two computers are connected via serial cable all packets arrive timely and with precision providing optimal throughput and the maximum efficiency while strictly adhering to the established protocol.

We encountered some problems as well when attempting to keep the file to send active in memory. When data was to be sent out on the line there were instances that created problems in attempting to packetize the proceeding data; and then also adjusting for when the CRC failed on the receiver’s side and we were forced to resend the data.

Determining the logic in the receive thread also caused many problems with the sequence in which different packets were expected to arrive and the actual sequence in which they arrived. Sometimes corrupt data was kept in the serial port buffer which conflicted with what was expected causing repetitive NAK’s to be sent.

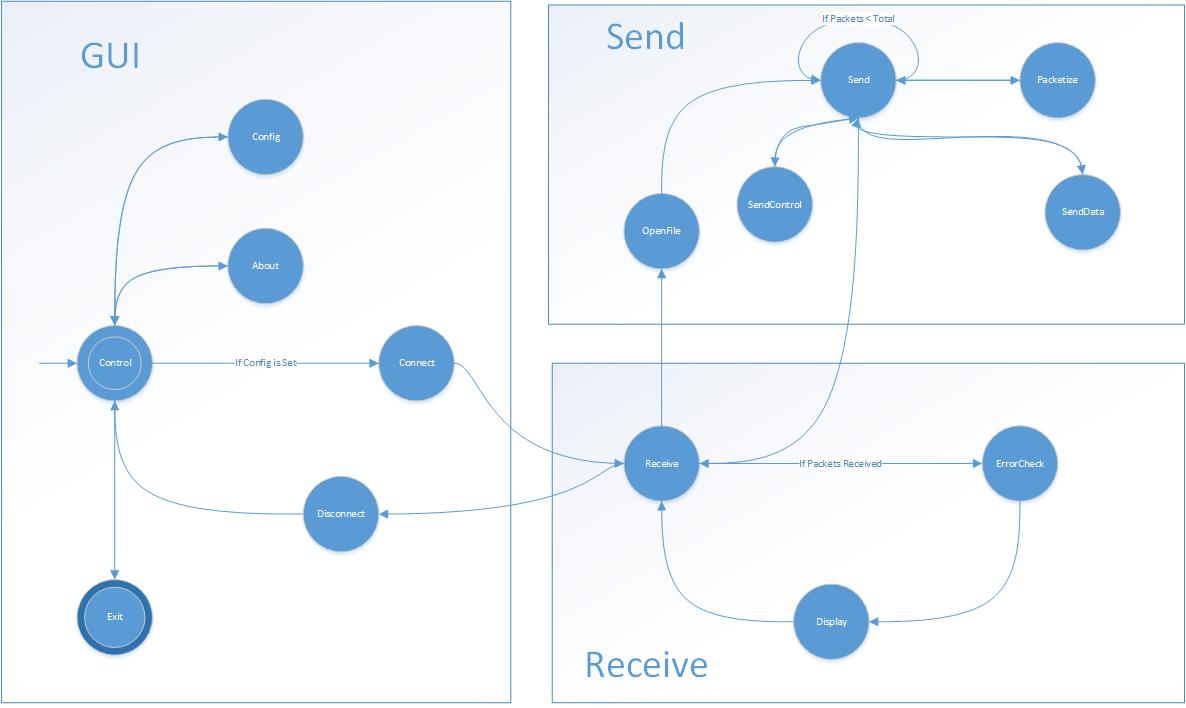
# Gantt Chart



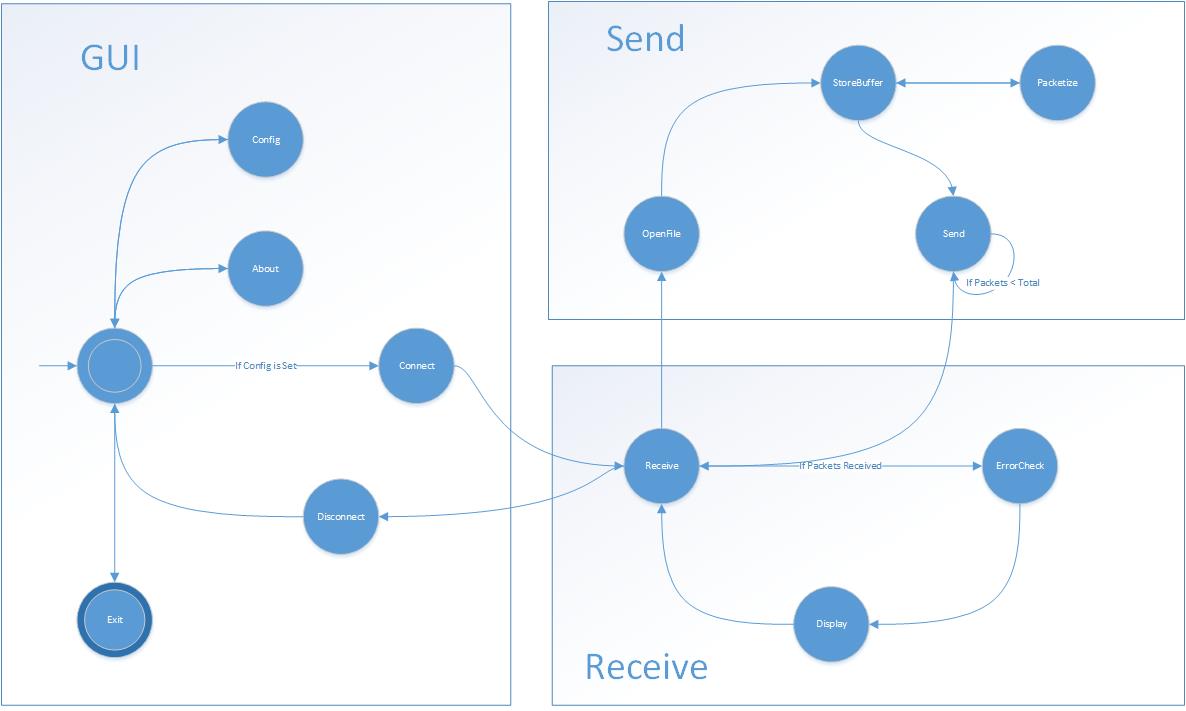


# State Diagram

## Improved:



## Original:



Function Prototypes

### ErrorCheck

unsigned short crc16(char, unsigned short);

void GenerateCRC(char\*, char\*);

BOOL ErrorCheck(char\*);

### Main

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

BOOL Register(HINSTANCE);

HWND Create(HINSTANCE, int);

BOOL Window\_OnCreate(HWND, LPCREATESTRUCT);

void Window\_OnCommand (HWND, int, HWND, UINT);

void Window\_OnDestroy (HWND);

BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM);

void OpenFileInitialize(HWND);

BOOL FileOpenDlg (HWND, PTSTR, LPCSTR);

BOOL FileRead(HWND, const LPCSTR);

void OkMessage(HWND, TCHAR\*, TCHAR\*);

BOOL ErrorCheck(char\*);

void GenerateCRC(char\*, char\*);

void DisplayText(HWND, LPCSTR);

void Window\_OnVScroll(HWND, HWND, UINT, int);

void Window\_OnPaint(HWND);

void Window\_OnSize(HWND, UINT, int, int);

BOOL FileSave(HWND, LPCTSTR);

BOOL FileSaveDlg (HWND, PTSTR, PTSTR);

### Packet

BOOL Packetize(CHAR\*, int);

BOOL PacketCheck(HWND, CHAR\*);

BOOL PacketCheckControl(HWND hwnd, CHAR\* packet);

void GetData(CHAR\*, CHAR\*);

### Physical

BOOL SendControl(HANDLE, int);

LONG\_PTR SendData(HANDLE hComm, char\* packetToSend);

BOOL ReadSerialPortControl(HANDLE hComm, char\* packetBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesRead);

BOOL ReadSerialPortData(HANDLE hComm, char\* packetBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesRead);

### Presentation

BOOL AddToBuffer(const char\*);

### Session

BOOL SetupPort (LPTSTR);

BOOL ConfPort (HWND\*, LPTSTR);

### Transport

DWORD WINAPI TransmitThread(LPVOID);

DWORD WINAPI ReceiveThread (LPVOID);

# Pseudocode

INT Window()

{

Register()

Create()

Set variables for commconfig

MessageLoop()

}  
BOOL Register(){

Register variables for opening main window.

}  
HWND Create{

CreateWIndow

}

LRESULT CALLBACK WndProc(){

Handle different Messages of the window

}  
BOOL Window\_OnCreate(){

OpenFileIntialize();  
 Return True

}  
void Window\_OnCommand(){

Connect()

SendFile()

If FileOpenDialog is True

If FileRead is False

OkMessage (displays file read error)

Disconnect()

Config()

If Com Port was OK and not in use

-Open Receive Thread

About()

Exit()

Window\_OnDestroy

}

Void Window\_OnDestroy{

CloseStream()

CloseThreads()

Exit Program

}

BOOL AboutDlgProc{

Open a DialogBox that displays information about the Program

}

void openFileInitialize{

initialize the parameters for opening a file.

}

FileOpenDialog(){

Initialize types of files to be seen

Open the file dialog

}

Void OkMessage(){  
 print the filename that failed to open

}

BOOL FileRead(){

Create a file

Get the file size

If Filesize does not equal zero

Malloc memory space for the file

Read the file and append two Null characters to the end

}

BOOL disconnect(){

CloseStream()

CloseThreads()

}

BOOL Config ()

{

if ("configured" flag not true)

{

error message : "Port not configured, please configure"

return;

OR

if (!config())

return;

}

clear any port handles or file descriptors that may be in use

get handle to serial port

if (handle is invalid)

{

error message : "Cannot open serial port"

return false;

}

set a "want to read" flag

createReceiveThread();

return true;

}

//RECEIVE

DWORD WINAPI Receive()

{

create temporary packet buffer to save 1024 bytes (1 packet)

set our listening/read parameters for the serial port, we want CHARACTER events (eg SetCommMask)

while (we want to read)

{

if (waiting for event success)

{

if (the event triggered was a CHARACTER event)

{

If there is 1024 chars to read

read 1024 characters into temporary packet buffer

packetcheck()

If there is 2 chars to read

read 2 characters into temporary packet buffer

packetcheck()

}

}

}

}

BOOL PacketCheck (char[1024] packet)

{

switch (char[1])

{

case: ENQ:

send (ACK);

Set "what we're waiting for" flag to PACKET\_DC1

break;

case DC1:

if ("what we're waiting for" is a PACKET\_DC2)

{

send (NAK);

break;

}

if (!ErrorCheck(char[1022], char[1023]))

{

send (NAK);

break;

}

send (ACK);

Display();//read the remaining 1020 characters

break;

case DC2:

if ("what we're waiting for" is a PACKET\_DC1)

{

send (NAK);

break;

}

if (!ErrorCheck(char[1022], char[1023]))

{

sendControlPacket (NAK);

break;

}

send (ACK);

Display();//read the remaining 1020 characters

break;

case NAK:

Set "What we're waiting for" flag to ACK

send (previous packet); //need a way to keep that

break;

case EOT:

// GO back to IDLE state

Set "what we're waiting for" flag to ENQ

break;

}

}

BOOL ErrorCheck(char[1024] packet){

GenerateTable(){

generate a table

calculate the CRC table

}

get(begin, end){

accumulate ()

}

}

VOID display(head)

{

print data

}

BOOL openFile(){

Initialize OpenFile struct

GetFileAttributes();

if (GetOpenFileName()){

Transmitfile();

return true;

}

return false;

}

/\*if data is sent\*/

VOID Transmit (File \*bufferWithFile)

{

create sentPacketCounter = 0

create packetToSend

do

{

// giving the packetize function the sentPacketCounter allows it to skip through the file

// if necessary. It can also determine if the next data packet will be DC1 or DC2 (mod2!=0)

while (packetCounter mod 5 != 0)

{

packetToSend = packetize (File Handle, sentPacketCounter, PKT\_TYPE\_DATA)

sendDataPacket (packetToSend);

decrement semaphore

++packetCounter;

set "what we're waiting for" flag to ACK

waitforSemaphore

if semaphore timed out

resend Packet

}

Wait for ENQ

} while (file not done)

}

CHAR\* packetize(File \*bufferWithFile, int SentPacketCounter)

{

1020 x sentPacketCounter = startingLocation

Read 1020 chars from the file buffer, starting at startingLocation into packet string

If we encounter eof

Pad remains Bytes with null

If (sentPacketBuffer % 2 == 0)

Packet[1] = DC1

Else

Packet[1] = DC2

//create return string returnstr

//add control bytes to returnstr

//if s[i] != eof

// returnstr += s[i]

//

// while i != 1022

// returnstr += '\0'

// returnstr += trailer bytes

// return returnstr

}

VOID sendDataPacket (char[1022] data, char DC\_TYPE?)

{

char[1024] packet

set char[0] to SYN

set char[1] to DC\_TYPE // DC1 or DC2

append data and CRC

write to serial port

}

VOID sendCtrlPacket (char CTRL\_TYPE)

{

char[2] packet

set char[0] to SYN

set char[1] to CTRL\_TYPE

write to serial port

}

# Conclusion

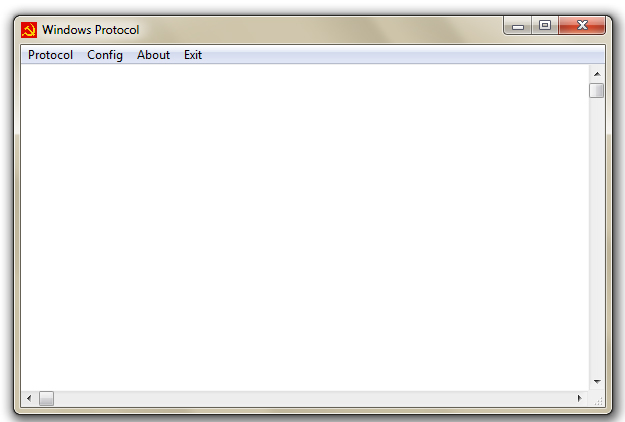
During our hours testing we have concluded that sending over a wireless medium is far less reliable for the receiver to receive the appropriate data; in our tests, retransmission rates were extremely high due to wireless interference and data corruption. A circular buffer could have helped with our results by more appropriately controlling the transmitting data.

# Test Cases

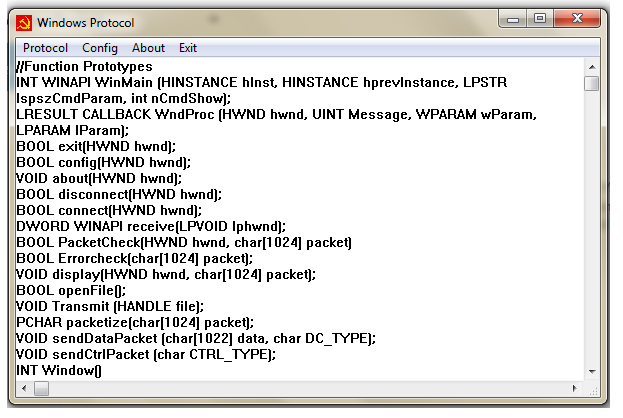
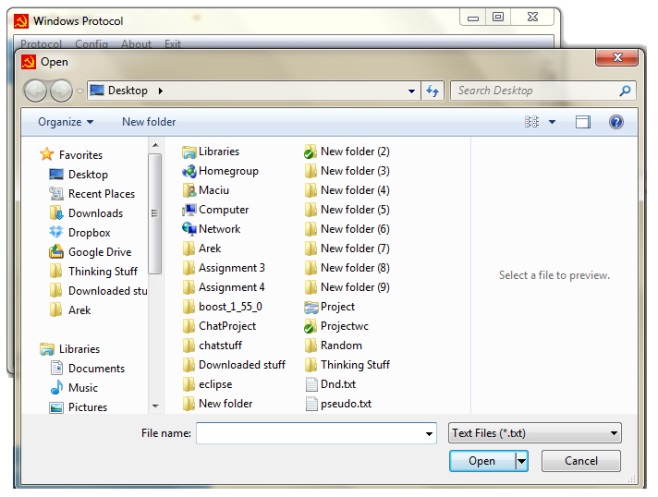
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Tests Description** | **Tools Used** | **Expected Result** | **Pass/Fail** | **Notes** |
| 1 | Opening the GUI | Wireless-GRVM | The program should open with correct menu options | Pass | See Figure 1 |
| 2 | Open a File/Display File | Wireless-GRVM | The program should be able to open a .txt and Display the Text | Pass | See Figure 2 |
| 3 | Threading/CPU usage of program | Wireless-GRVM/Resource Monitor | The program should not use a lot of CPU resources | Fail | See Figure 3 |
| 4 | Threading/CPU usage of program | Wireless-GRVM/Resource Monitor | The program should not use a lot of CPU resources | Pass | See Figure 4 |
| 5 | Packet Padding at EOF Successful | Wireless-GRVM/HyperTerminal | The W’s display the padding after the EOF was reached | Pass | See Figure 5 |
| 6 | Packet Sending | Wireless-GRVM/HyperTerminal | Displays an error sent by GRVM to HyperTerminal | Fail | See Figure 6 |
| 7 | Packet Sending/Receiving | Wireless-GRVM | Received packet is displayed | Pass | See Figure 7 |

# Figures

## Test 1: Initial Start Up (PASS)

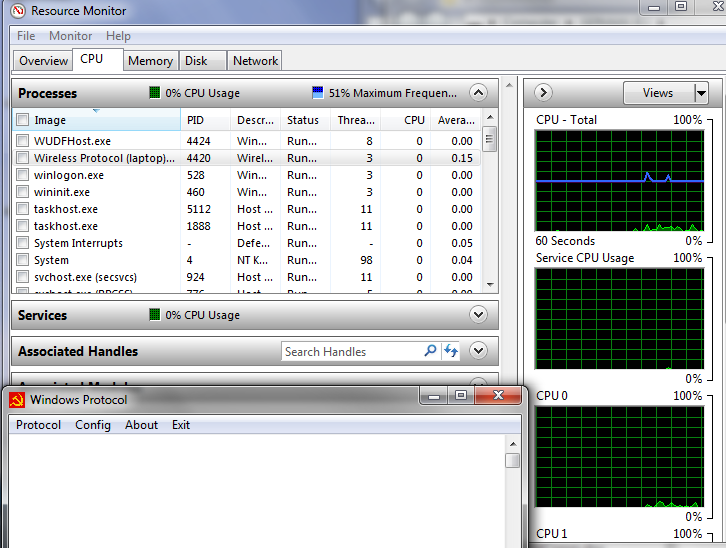


## Test 2: Opening and Displaying a File (PASS)

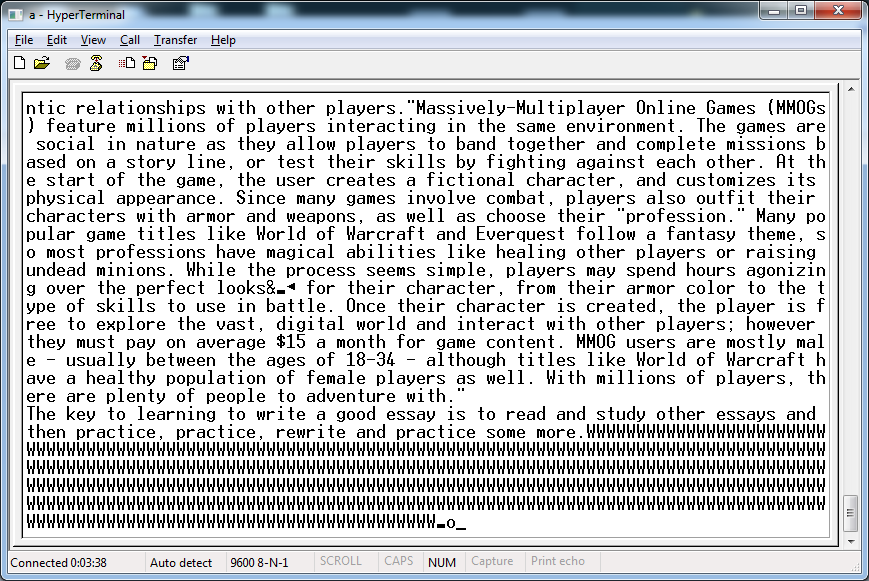


## Test 3: Threading (FAIL)

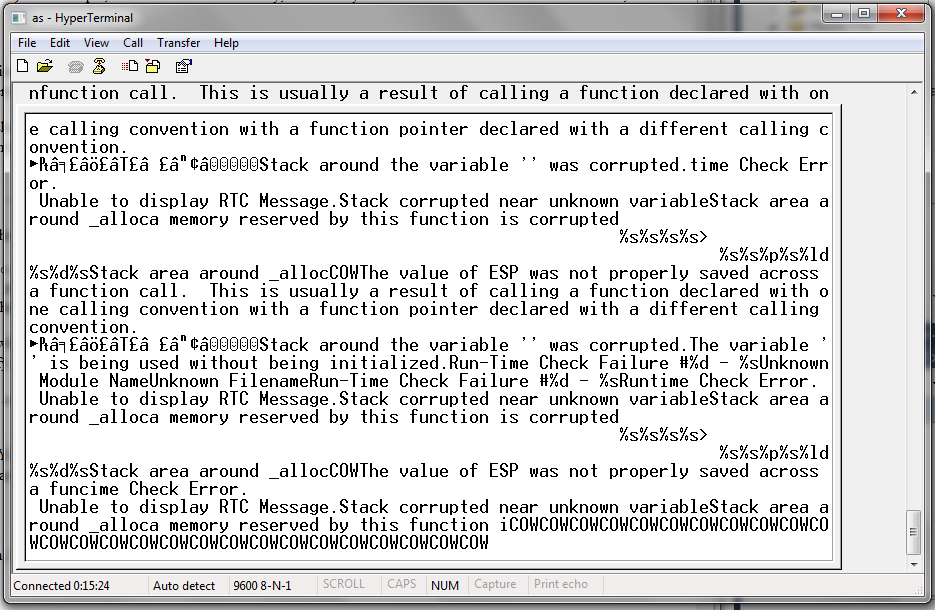
## Test 4: Threading (PASS)



## Test 5: (PASS)



## Test 6: (FAIL)



## Test 7: (PASS)

