BRADLEY
University

KOMATSU

Senior Project: ECU Communication Performance
Work Log

John Greifzu
Grant Abella

Advised by:
Aleksander Malinowski

Wednesday, September 26th:

Meeting at Komatsu
Bradley attendees: John Greifzu, Grant Abella, Dr. Malinowski
Komatsu attendees: Joshua Rohman, Paul Maynard, Tim Imig, and Jason Schepler
Meeting notes

Wednesday, October 3rd:

Met at library
Worked on problem statement and website
Started up project

Wednesday, October 10th:

Worked in lab room 249
Researched PiCAN2 communication in C

Saturday, October 13th:

Worked at 1515 callender
Followed this tutorial to get piCAN working on fresh OS install:
Got the CAN working between two raspberry pis in C
We found out what types of messages we can send between the pis over can
Wrote a program that takes input for a message, including the ID and message and then sends it

```
int main(int argc, char **argv)
{
        int s; /* can raw socket */
        int nbytes;
        struct sockaddr_can addr;
        struct can_frame frame;
        struct ifreq ifr;

        int i;

        /* CAN message to be sent out */
        unsigned char buff[20];
        unsigned char m_id[3];
        unsigned char mes[16];
        printf("Input message ID:");
        scanf("%s", &m_id);
        printf("\nInput message: ");
        scanf("%s", &mes);
        strcpy(buff,m_id);
        strcat(buff,"#");
        strcat(buff,mes);
        printf("\nBuffer to be sent: %s\n", &buff);
```

Figure 1.1 - Program to enter custom message and ID.

```
pi@raspberrypi: ~/linux-can-utils
pi@raspberrypi:~/linux-can-utils $ ./cantest
Input message ID:5AA

Input message: 12345678

Buffer to be sent: 5AA#12345678
CAN testing
0
pi@raspberrypi:~/linux-can-utils $ ./cantest
Input message ID:5AB

Input message: 9ABC

Buffer to be sent: 5AB#9ABC
CAN testing
0
pi@raspberrypi:~/linux-can-utils $ ./cantest
Input message ID:5AC

Input message: DEF012345678

Buffer to be sent: 5AC#DEF012345678
CAN testing
0
pi@raspberrypi:~/linux-can-utils $
```

Figure 1.2 - Three runs of the program with different message IDs and messages being sent.

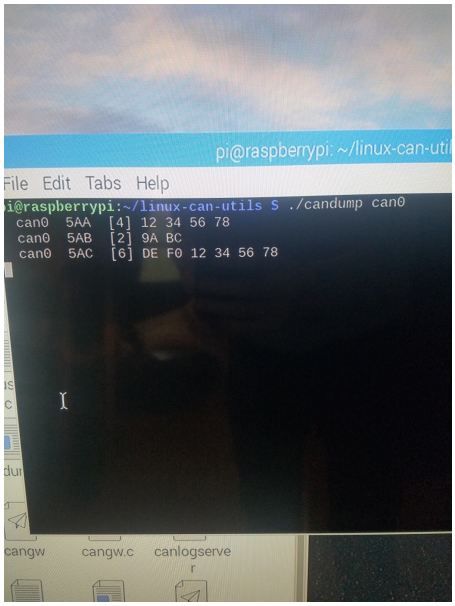

Figure 2.1 - Messages from the above figure being received on the second Pi.

Wednesday, October 17th:

Met at 1515 Callender

Finalized Problem Statement & Functional Requirement document, emailed to Dr. Malinowski and Dr. Lu.

Researched encryption algorithms to potentially implement and measure

SHA, Blowfish, and potentially other encryption: could use cryptlib

Questions to ask on Friday's meeting:

1. Sending multiple messages due to encryption?
2. How will everything be sent in hex?
3. Will we need to use more than 2 raspberry pis? If so, what kind of processes should be performed to incorporate all of them?

Tuesday, October 23rd:

Met at 1515 callender

Made revisions to statement and function requirement document

Added block diagram

Discussed how to send the data through CAN

Monday October 29th:

Met at 1515 Callender

Set up cryptlib on the raspberry pis

Downloaded cryptlib and manual from here

Compiled and tested cryptlib:

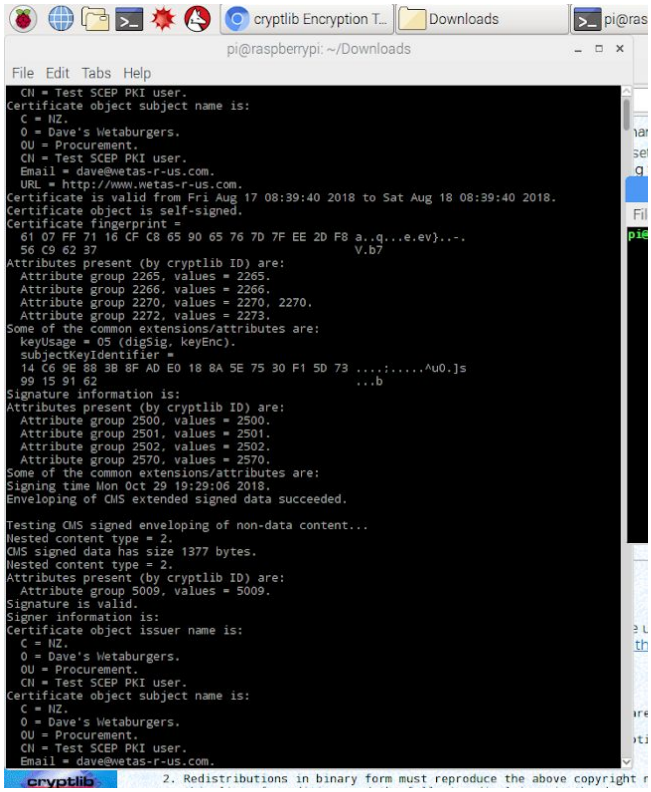

Figure 3.1 - Testing the cryptlib library functions.

Encountered difficulties with cryptlib, might switch to using libsodium.

Wednesday October 31st:

Went with libsodium because of simplicity, process was as follows:

1. Extract the compressed folder
2. Run **./configure**
3. Run **sudo make install**

Wrote a small test program to see if the library worked, test.c, inside the libsodium include folder.

Compiled using the command **gcc -o test test.c -lsodium**

Found a test program to demonstrate different features of the library, made plans for the encryption of messages.

Thursday November 15th:

Worked on Proposal

Installed libsodium on raspberry pi, tried to get it to work

Monday November 19th

Worked on proposal presentation

Wednesday November 21st:

Got libsodium installed on both Raspberry Pis

Merged the secret key encryption program with the cantest program

Created a makefile to compile can_enc and ensure all dependabilities are linked

Below is a screenshot of the makefile:

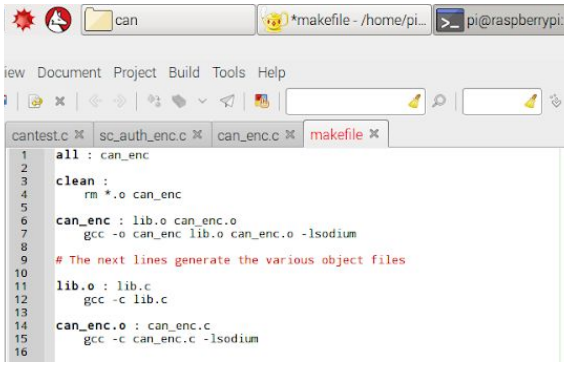

Figure 4.1 - makefile for compiling and linking can_enc.c.

Friday, November 23rd:

Made many modifications to can_enc.c

Successfully achieved the following:

1. Get input from user
2. Encrypt the message into binary
3. Convert cipher text from binary to hex
4. Split the hex string into individual can frames, complete with sequential message ids
5. Send the frames over CAN to the other Raspberry Pi

The following screenshots show this process:

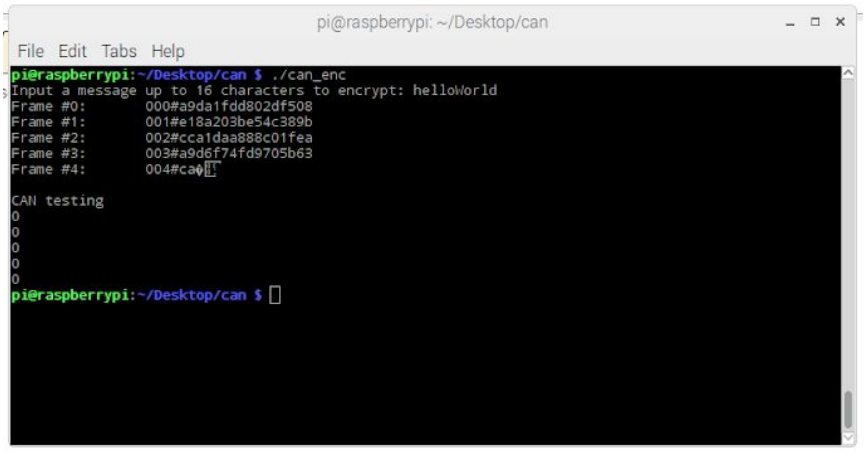

Figure 4.2 - Execution of can_enc from the master Pi.

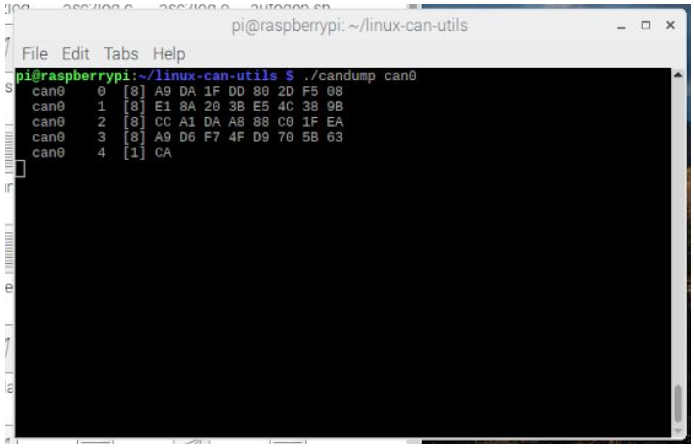

Figure 4.3 - Encrypted message being received by the slave Pi via CAN bus.

Surprisingly, the null characters at the end of the final generated frame did not cause any issues when received.

Things we still must accomplish:

1. Reconstruct the received message into a single hex ciphertext
2. Convert the reconstructed ciphertext back into binary
3. Decrypt the binary ciphertext
    a. This will require the original nonce used to encrypt the message
    b. We will need to get the nonce sent before decryption can occur

Saturday, November 24th:

Began work on a modified candump program to achieve the objectives listed above

Tuesday, November 27th:

Met with Dr. Malinowski to discuss project presentation, discussed things to include:

Talk about other library that we considered
Talk about linking/compiling
How we are connecting to the Pis
Possibly add Gantt chart?

Check sakai/gmail for example presentation by Dr. Lu

Thursday, January 24th:

Met with Malinowski
Need to schedule meeting with Komatsu - possibly 1/31 @ 10:45? Cc Lu and Mali
For ethernet in the future - communication using UDP not TCP

Thursday, January 31st:

Met at Komatsu
Reviewed progress so far
Set direction for future
Find out a way to measure impact of the system
For ethernet we will probably just use a crossover cable

Tuesday, February 5th:

Met at 1515 Callender
Set the raspberry pis back up
Tested them and made sure they still work
Asked Josh and Jason for the library about the blowfish algorithm

Need to ask malinowski about public and secret key

-What should we use -

-How to send over

-What do we need to send

-What do we need to decrypt it

To do: set command to boot and set bitrate for can0, edit candump to save and decrypt message

Thursday, February 7th:

Encryption /decryption key should be hardcoded. No need to send it over. Either within the program or in a text file.

For serial connection: bend the pins using pliers, attach wires.

Possibly connect pins to shield individually

Tuesday, February 12th:

Thought of new ways to connect with the raspberry pis

Tried to just wire over the 6 pins the data sheet said it uses\

Wednesday, February 13th:

Found out the CAN board needed power,Needs both 5V and 3.3V

Uses PIN 1,2 and 9 for power

Uses 15,19,21,22,23,24

Thursday, February 14th:

Discussed getting supplies from ECE surplus

Discussed system monitoring tools:

Sar

Tload

We will have to try different methods to see what works best

Wednesday, February 20th:

Set up the program so the receiver has the data in a format that we can use, also set so it was in the correct

Looked more into detail about how the encryption works

Wednesday, February 27th:

Created key and nonce files with generated values.

Achieved consistent encryption for the same raw message.

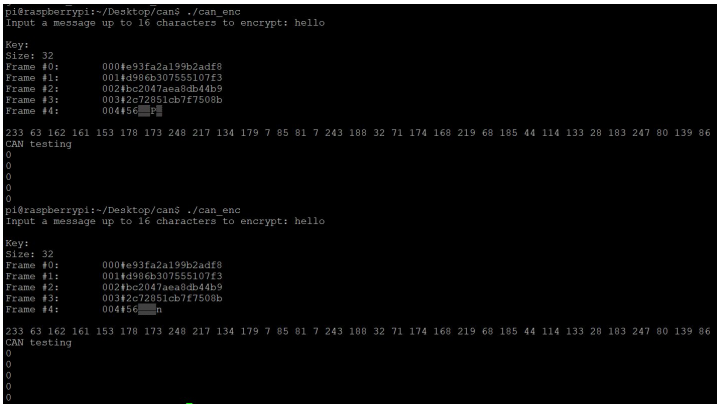

Figure 6.1 - Encrypting the same message twice using hardcoded key and nonce.

Decryption on the receiving end now works

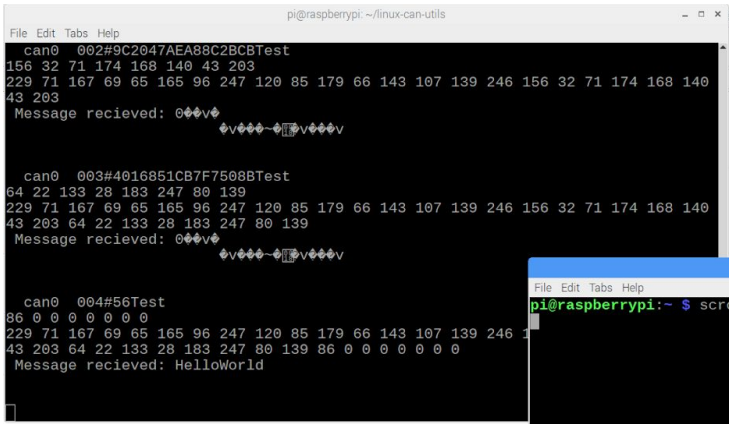

Figure 7.1 - HelloWorld message being decrypted on the receiving Pi

Thursday, February 28th:

Met in Malinowski's office

Discussed measuring performance

Round trip will be best, send 1000s of messages and get average

Send time stamp from the sender, have receiver send back the timestamp and compare

time.h file vs sys/time.h. sys/time.h is more accurate

Start with measuring the time delay in software

Eventually we could move to a hardware measurement

Set a pin high when sent, set low when trip is finished. Measure on oscilloscope.

It will HAVE to be an average measurement because of how the linux can library works

We could use **vmstat** within the program. vmstat 1 > data.txt will run every 1s and log

Tuesday, March 12th:

Met at 1515 Callender

Connected both boards to serial

Used top and ps to do initial performance checks

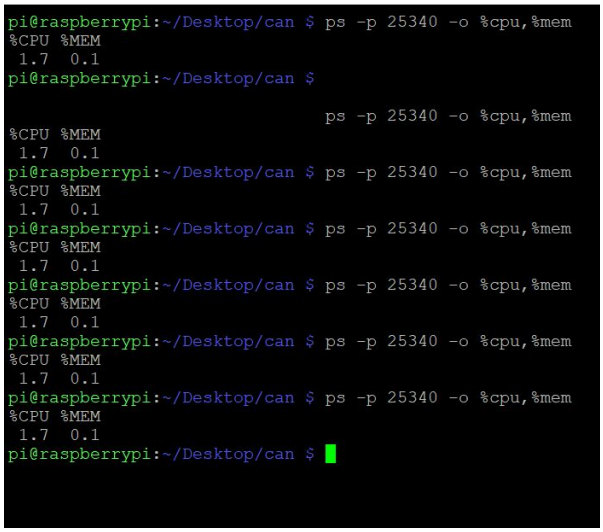

Figure 7.2 - Testing the **ps** command with process ID.

top -b -p PID > nameoffile.txt

This will save all the information from top

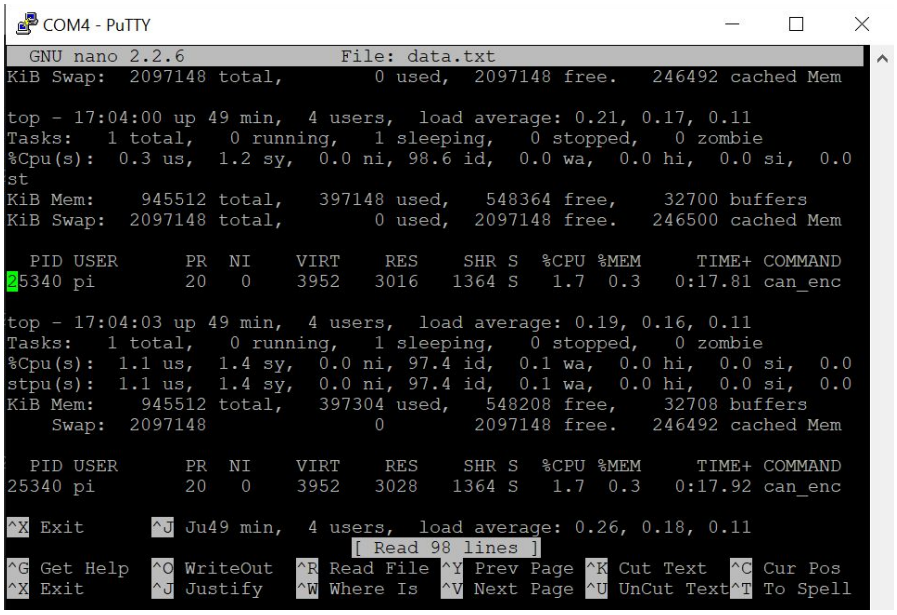

Figure 8.1 - top command output to text file

Will meet tomorrow to fix receiver crashing

- Need to reset the message variable after it decrypts it

Wednesday, March 13th:

We fixed the receiving program

- Now only prints after each full message is received

Next thing to do is to get a program to send and receive messages

Also will try and meet with komatsu after spring break

- Show that we can encrypt,send,receive decrypt messages
- Also show some initial performance metrics

Thursday, March 14th:

Met with Malinowski

Need to compare performance of encrypted vs unencrypted

- Messages should probably be the same length
- Ask Komatsu what their average packet size is

Can use excel, matlab to get the data from the performance output files

For data:

- Average, standard deviation, etc

Schedule meeting with Komatsu for after break, Thursday the 28th at 11?

Bring the boards to Jobst earlier in the week to make sure they will work for demo.

Monday, March 18th:

Fixed color output over serial for both boards

Edited cron tables for both boards so that CAN startup command runs on boot

Wrote a test program working with /sys/time.h

- Program finds the difference between two times
- Similar implementation will be made within the sending/receiving system

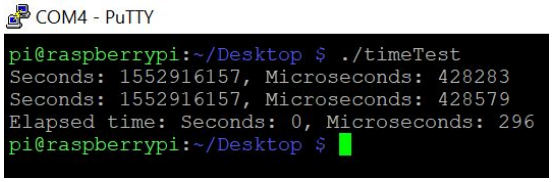

Figure 9.1 - /sys/time.h testing

Thursday, March 21st:

Rewrote sending and receiving programs to both send and receive.
Both programs utilize two sockets
Messages are sent in plaintext
There are issues with both boards synchronizing.

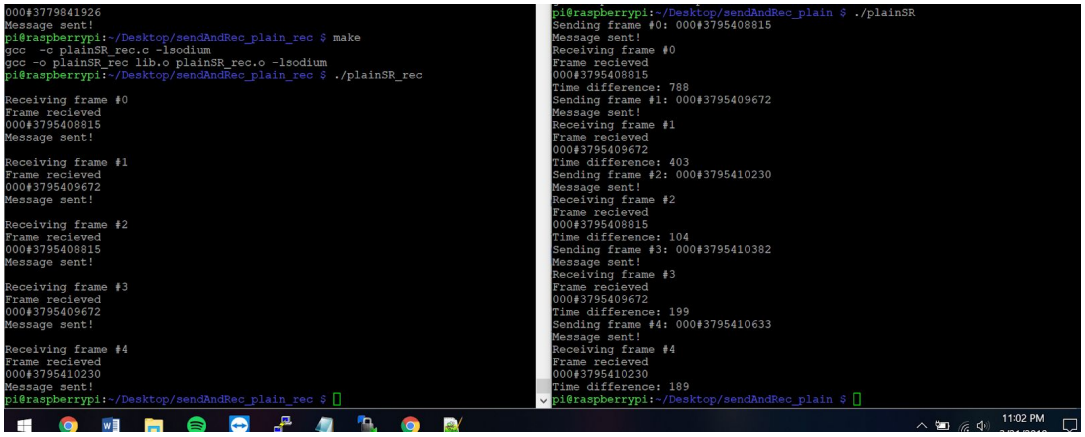

Figure 9.2 - Boards running their respective programs. Some frames are wrongly being repeated

Tuesday, March 26th:

Fixed various bugs in the programs
Recorded data from the program in a .csv

Thursday, March 28th:

Need to change CAN bus speed from 500kbps to 250kbps
Look into bypassing message IDs to save on frame space.
The results are around what should be expected, about .5 byte/us = ~64 us one way, ~128 us both
Make a version of Tuesday's program that sends 4-5 frames and measures time delay
    This will be more comparable to measurements made for encrypted tests
Malinowski will be in Jobst until 5pm tomorrow
Move setup to Jobst 144 - Mali's wifi SSID: malilab, password: dram-portable-lab-key
Network in 144 - ask Jon for password
Possibly reinstall raspbian lite on PIs

Friday March 29th:

Moved everything to Jobst for meeting next thursday with Komatsu
Changed speed of CAN to 250kb/s to match what Komatsu uses
Made the times much more consistent
Added "ifconfig can0 txqueuelen 1000" to run at every boot
Need this or else if a lot of CAN messages are sent it will say no buffer space available
Did analysis on some data
Ask Malinowski about not closing sockets

Ask Malinowski about timeout
Use setsocketopt for reuse address and timeout
Changed to only one socket- fixed a lot of bugs
The program was receiving its own messages

Monday, April 1st:

Fixed bugs in program
Added program parameter for number of messages to send
Added more performance measurement
CPU and memory usage is gathered at intervals during the program using system call

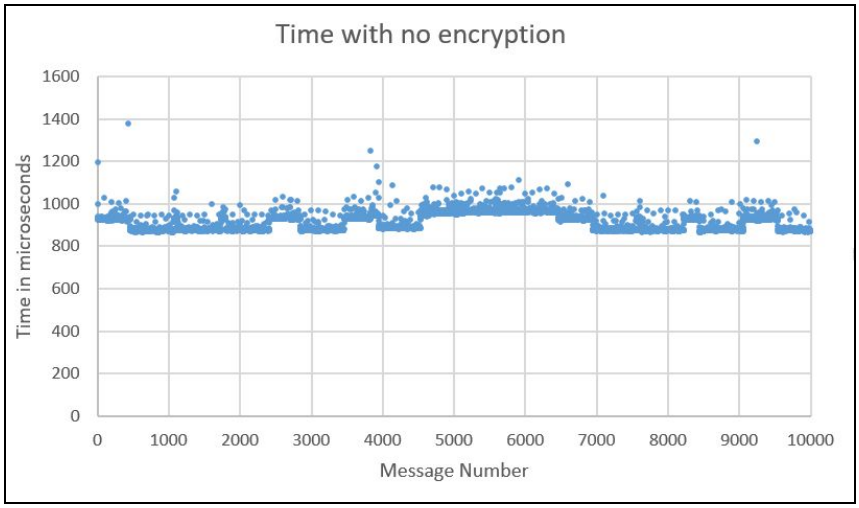

Figure 10.1 - Graph of time delay (in program using system call) for 10000 messages.
Times in above figure fluctuate much more than before, probably a result of using the system call.
May need to call the command differently or incorporate a function like the one described here.

Wednesday, April 3rd:

Added encryption to the sending program
Fixed bugs

Thursday, April 4th:

Josh and Jason met with us in Jobst senior project lab.
Demonstrated implementation and showed them our preliminary results.
Discussed how to move forward.
Place importance on Blowfish implementation first, then ethernet.
For ethernet: send one 32 byte message, do not send 4 separate packets

Sunday, April 7th:

Planned to meet in Jobst senior project lab but it was closed.
Began testing Blowfish algorithm.
Wrote a basic test program to encrypt an 8 byte value and then decrypt the value.

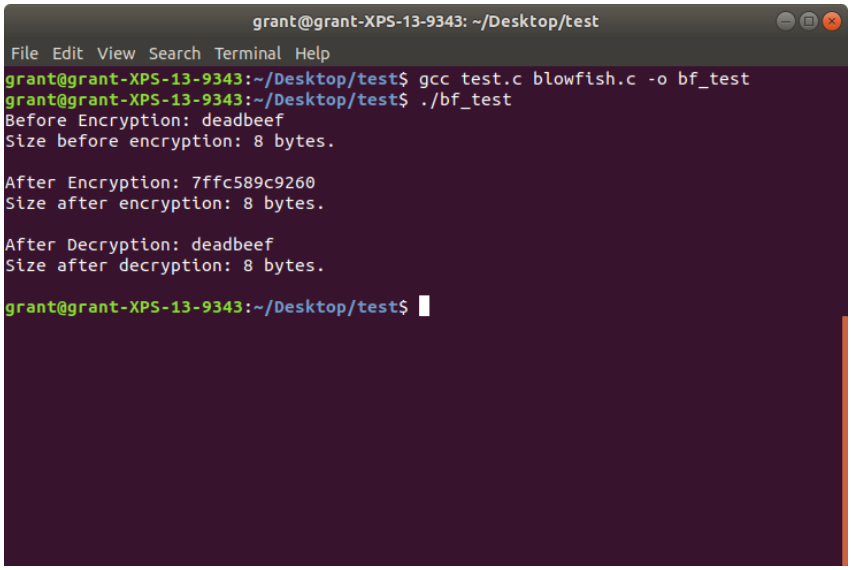

Figure 11.1 - Basic encryption and decryption using Blowfish.

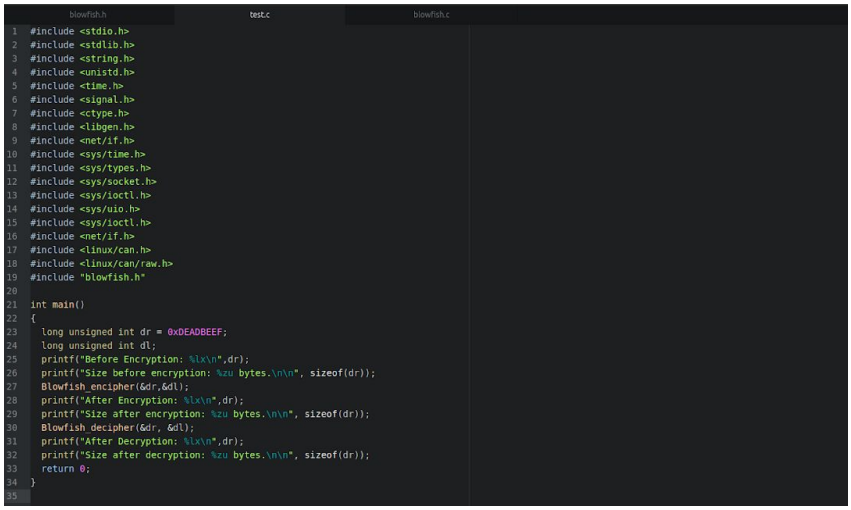

Figure 11.2 - Code for basic encryption and decryption using Blowfish.

Monday, April 8th:

Implemented the blowfish encryption
Still some bugs to work out
Worked on poster

Tuesday, April 9th:

Got the program to encrypt and decrypt both ways
Now just need to add back in the performance measurements
Also use ethernet

Wednesday, April 10th:

Now can record timing data with encryption
Cleaned up the code, reimplemented custom run count
Gathered times, CPU usage, memory usage, and CPU time for encryption with Blowfish
Gathered times, CPU usage, memory usage, and CPU time benchmark values (no encryption)

Created excel sheets for both cases.

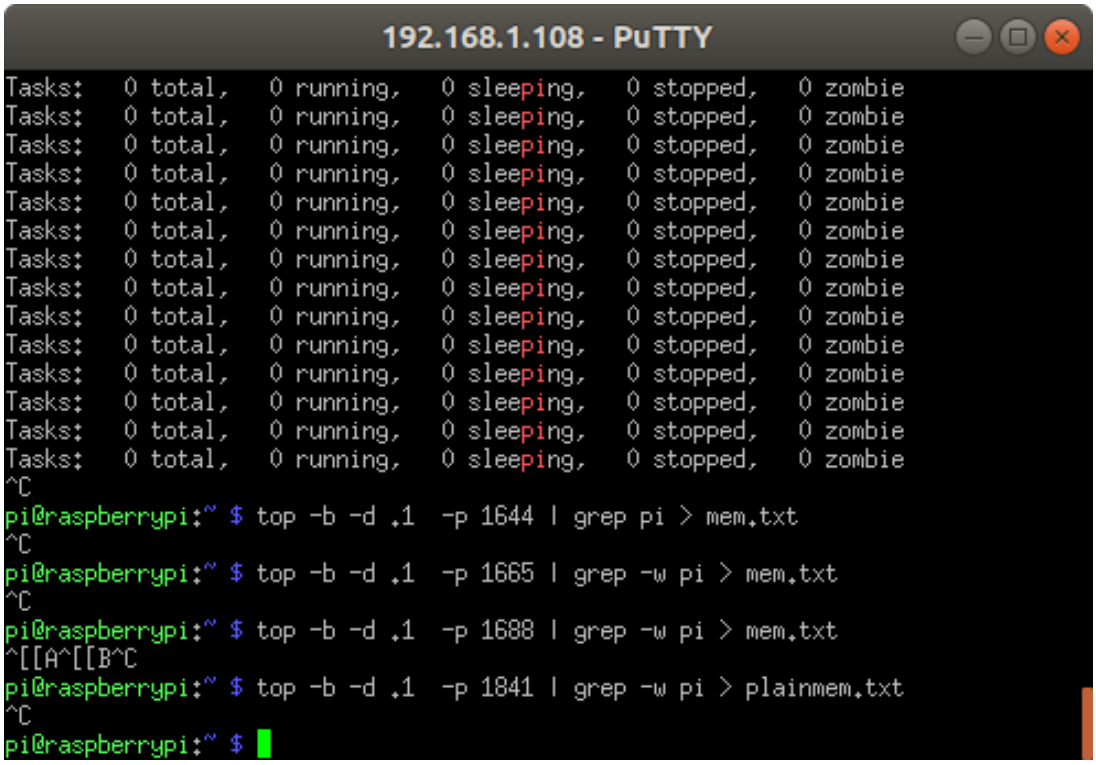

Figure 12.1 - Method for gathering resource usage.

Thursday, April 11th:

Met with Malinowski
Can use examples from 473 for UDP
Rayleigh function for data - possibly use Matlab, raylfit for parameter estimates
Raylstat for mean and variance
Combine both results into single graph
Got ethernet implemented
    Both plaintext and encrypted
    Gathered times, CPU usage, memory usage, and CPU time for encryption with Blowfish
    Gathered times, CPU usage, memory usage, and CPU time benchmark values plaintext

Thursday, May 2nd:

Add distribution graphs, division of labor (agile methodology?), timeline of work, slide numbers
Find out what Ethernet cable was used
Create distribution histograms for each case, with comparable ranges on axes