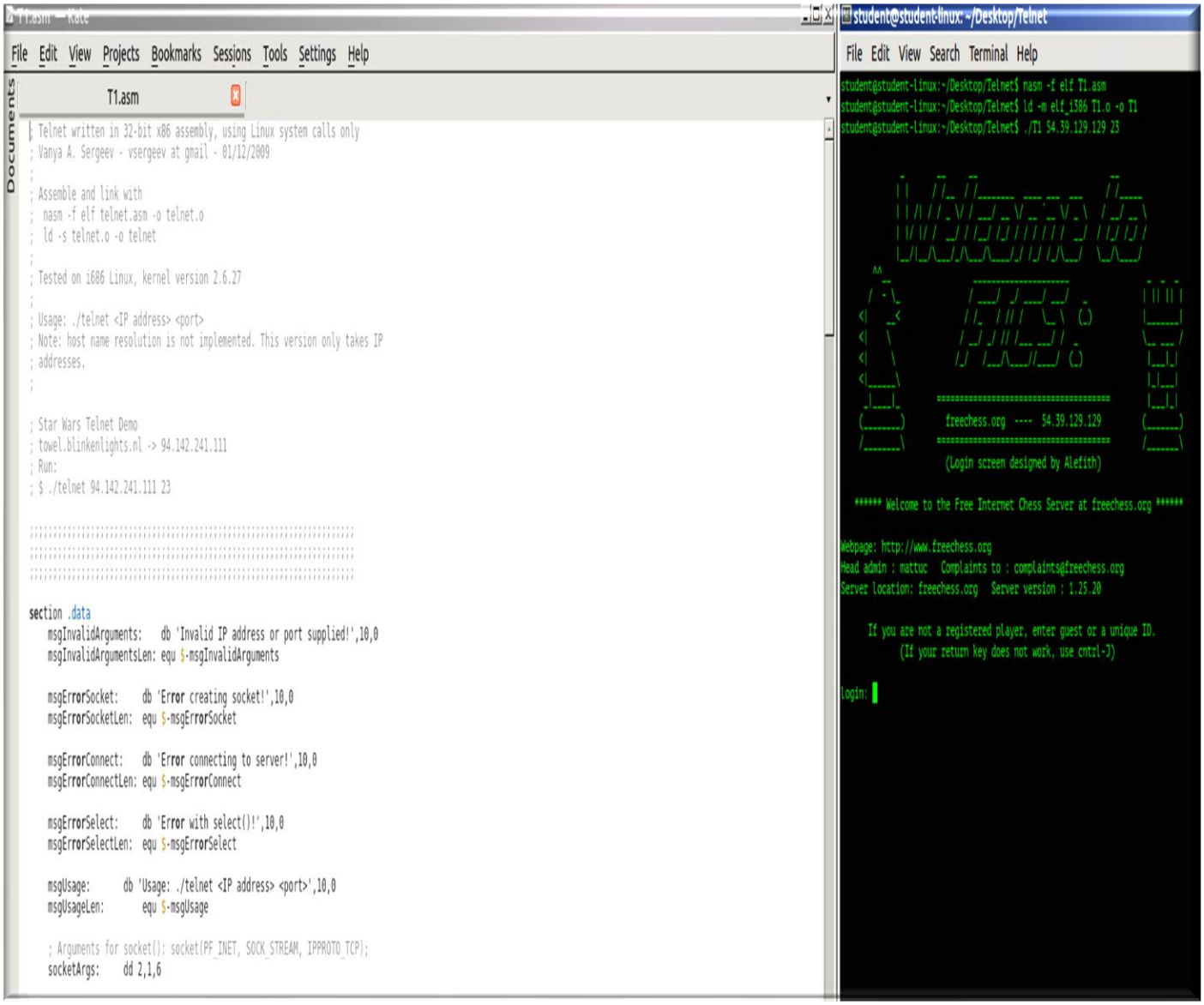
Vivek Madala

12/10/2023

CYBV 471

Professor Dr. Meky

CYBV 471 Telnet Final



```
section .data
    msgInvalidArguments:    db 'Invalid IP address or port supplied!',10,0
    msgInvalidArgumentsLen: equ $-msgInvalidArguments

    msgErrorSocket:     db 'Error creating socket!',10,0
    msgErrorSocketLen:  equ $-msgErrorSocket

    msgErrorConnect:    db 'Error connecting to server!',10,0
    msgErrorConnectLen: equ $-msgErrorConnect

    msgErrorSelect:     db 'Error with select()!',10,0
    msgErrorSelectLen:  equ $-msgErrorSelect

    msgUsage:       db 'Usage: ./telnet <IP address> <port>',10,0
    msgUsageLen:        equ $-msgUsage

    ; Arguments for socket(): socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    socketArgs:     dd 2,1,6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

section .bss
    ; Socket file descriptor returned by socket()
    sockfd:         resd    1

    ; Storage for the 4 IP octets
    ipOctets        resb    4

    ; Storage for the connection port represented in one 16-bit word
    ipPort          resw    1

    ; Arguments for connect():
    ;   connect(sockfd, serverSockaddr, serversockaddrLen);
    connectArgs     resd    3

    ; The read file descriptor array for select()
    masterReadFdArray   resb    128
    checkReadFdArray    resb    128
    readFdArrayLen      equ 128

    ; sockaddr_in structure that needs to be filled in for the
    ; connect() system call.
    ;   struct sockaddr_in {
    ;       short            sin_family;
        ;       unsigned short   sin_port;
        ;       struct in_addr   sin_addr;
        ;       char             sin_zero[8];
    ;   };
    serverSockaddr      resb    (2+2+4+8)
    serverSockaddrLen   equ 16

    ; Read buffer for reading from stdin and the socket
    readBuffer      resb    1024
    readBufferLen       resd    1
    readBufferMaxLen    equ 1024
```

Reserving memory space and initializing variables in data

```
section .text
    global _start

_start:
    ; Pop argc
    pop eax ;pop eax from the stack
```

Start the program and validate the input

```
; Check if we have the correct number of arguments (2), for the
    ; program name and IP address.
    cmp eax, 3
    je parse_program_arguments
```

If there are correct number of arguments…

If there's an incorrect number of arguments…

Yes, Parse the arguments given by user (prog. Name and IP address)

```
; Otherwise, print the usage and quit.
push msgUsage   ;message to correct the user on how to run the program
push msgUsageLen    ;length of proper usage message
call cWriteString   ;call special function for writting string
add esp, 8      ;remove top 8 bytes from the stack pointer
call cExit  ;call function to end the program
```

NO, End the program with error message if number of arguments is incorrect based on user input

```
parse_program_arguments:
    ; Set the direction flag to increment, so edi/esi are
INCREMENTED
    ; with their respective load/store instructions.
    cld   ;clear direction flag

    ; Pop the program name string
    pop eax     ;pop eax from the stack
```

Reading the user input and converting string to decimals

```
; cStrIP_to_Octets
;   Parses an ASCII IP address string, e.g. "127.0.0.1", and stores the
;   numerical representation of the 4 octets in the ipOctets variable.
;       arguments: pointer to the IP address string
;       returns: 0 on success, -1 on failure
;
cStrIP_to_Octets:
    push ebp     ;push ebp ontp the stacl
    mov ebp, esp     ; move the stack pointer into the base pointer

    ; Allocate space for a temporary 3 digit substring variable of the IP
    ; address, used to parse the IP address.
    sub esp, 4  ;subtract 4 bytes from stack pointer

    ; Point esi to the beginning of the string
    mov esi, [ebp+8]     ;move the address of base pointer + 8 into esi

    ; Reset our counter, we'll use this to iterate through the
    ; 3 digits of each octet.
    mov ecx, 0  ;move 0 into ecx

    ; Reset our octet counter, this is to keep track of the 4
    ; octets we need to fill.
    mov edx, 0  ;move zero into edx

    ; Point edi to the beginning of the temporary
    ; IP octet substring
    mov edi, ebp     ;move base pointer into edi
    sub edi, 4  ;subtract 4 from edi

    string_ip_parse_loop:
        ; Read the next character from the IP string
        lodsb   ;Load byte at address DS:ESI into AL
        ; Increment our counter
        inc ecx ;increment ecx counter
```

Start of the cStrip_to_Octets function which is called from here, next page details error checking based on this function and how program handles it

```
string_ip_parse_loop:
    ; Read the next character from the IP string
    lodsb   ;Load byte at address DS:ESI into AL
    ; Increment our counter
    inc ecx ;increment ecx counter

    ; If we encounter a dot, process this octet
    cmp al, '.' ; compare al to '.'
    je octet_complete   ;jump if equal to octet_complete
    ; If we encounter a null character, process this
    ; octet.
    cmp al, 0   ;compare al to 0
    je null_byte_encountered    ;jump if equal to null_byte_encountered
    ; If we're already on our third digit,
    ; process this octet.
    cmp ecx, 4  ;compare ecx to 4
    jge octet_complete  ;if ecx is greater than 4 jump to octet_complete

    ; Otherwise, copy the character to our
    ; temporary octet string.
    stosb   ;Store AL at address ES:EDI

    jmp string_ip_parse_loop ;jump to string_ip_parse_loop

null_byte_encountered:
    ; Check to see if we are on the last octet yet
    ; (current octet would be equal to 3)
    cmp edx, 3  ;compare edx to 3
    ; If so, everything is working normally
    je octet_complete   ;jump if equal to octet_complete
    ; Otherwise, this is a malformed IP address,
    ; and we will return -1 for failure
    mov eax, -1 ;move -1 into eax
    jmp malformed_ip_address_exit   ;jump to malformed_ip_address_exit

octet_complete:
    ; Null terminate our temporary octet variable.
    mov al, 0   ;move 0 into al
    stosb   ;Store AL at address ES:EDI

    ; Save our position in the IP address string
    push esi    ;push esi onto the stack
    ; Save our octet counter
    push edx    ;push edx onto the stack
```

Process the digit and check for completion of octet or incorrect IP address

```
    ; Save our position in the IP address string
    push esi    ;push esi onto the stack
    ; Save our octet counter
    push edx    ;push edx onto the stack

    ; Send off our temporary octet string to our cStrtoul
    ; function to turn it into a number.
    mov eax, ebp    ;move base pointer into eax
    sub eax, 4  ;subtract 4 from eax
    push eax    ;push eax onto the stack
    call cStrtoul   ;call cStrtoul routine
    add esp, 4  ;remove 4 bytes from stack pointer

    ; Check if we had any errors converting the string,
    ; if so, go straight to exit (eax will hold error through)
    cmp eax, 0  ;compare eax to zero
    jl malformed_ip_address_exit    ;if less than jump to malformed_ip_addre

    ; Restore our octet counter
    pop edx ;pop edx from stack

    ; Copy the octet data to the current IP octet
    ; in our IP octet array.
    mov edi, ipOctets   ;move ipOctets variable into edi
    add edi, edx    ;add edx to edu
    ; cStrtoul saved the number in eax, so we should
    ; be fine writing al to [edi].
    stosb   ;Store AL at address ES:EDI

    ; Increment our octet counter.
    inc edx ;add 1 to edx

    ; Restore our position in the IP address string
    pop esi ;pop esi from the stack
    ; Reset the position on the temporary octet string
    mov edi, ebp    ;move base pointer into edi
    sub edi, 4  ;subtract 4 from edu
    ; Continue to processing the next octet
    mov ecx, 0  ;move 0 into ecx

    cmp edx, 4  ;compare edx to 4
    jl string_ip_parse_loop ;if less than jump to string_ip_parse_loop

; Return 0 for success
mov eax, 0  ;move 0 into eax

malformed_ip_address_exit:
mov esp, ebp    ;move base pointer into stack pointer
pop ebp ;pop base pointer
ret ;return
```

Parsing the next octet from the IP address string and then next is to check for completion or errors

Going back to main () for IP address validation

```
add esp, 4  ;remove top 4 bytes from the stack pointer
; Check for errors
cmp eax, 0  ;compare eax to zero
jl invalid_program_arguments
;if less than, jump to "invalid_program_arguments"
```

Here checking to see if there are errors in IP address, if so proceed with…

YES, Error check if eax is greater than zero (jump to opening network socket)

NO, Error check to see if eax is NOT greater than zero (print error and

```
; Next on the stack is the port
; Convert the port string to a 16-bit word.
call cStrtoul
;call function to convert port string to work
add esp, 4
;remove top 4 bytes from the extended stack pointer
mov [ipPort], eax
;move the contents of eax into the address of the ipPort variable
; Check for errors
cmp eax, 0
;compare eax to zero
jge network_open_socket
;if eax is greater than zero jump to network_open_socket (no error)
```

```
; Otherwise, print error for invalid
arguments and quit.
    invalid_program_arguments:
    ;fall into
invalid_program_arguments if eax is not
greater than zero
    push msgInvalidArguments
    ;push the invalid arguments message
on the stack
    push msgInvalidArgumentsLen
    ;push the message length
    call cWriteString
    ;call the function to write the
string
    add esp, 8
    ;remove top 8 bytes from stack
pointer
    call cExit
```

```
network_open_socket:
    ;;; Open a socket and store it in sockfd ;;;

    ; Syscall socketcall(1, ...); for socket();
    mov eax, 102 ;move 102 to eax
    mov ebx, 1   ;move 1 to ebx
    mov ecx, socketArgs     ;mov socketArgs variable into ecx
    int 0x80     ;call the kernel

    ; Copy our socket file descriptor to our variable sockfd
    mov [sockfd], eax  ;mov thte socket file descriptor from eax into the
sockfd variable
```

Open network socket and store in sockfd if...

```
cmp eax, 0 ;compare eax
to zero
    jge network_connect

    ;if eax is greater
than zero
    ;jump to
network_connect lable
```

YES, eax is greater than zero (call to connect to the network)

NO, eax is less than or equal to zero (print error and quit)

Checks if socket () is returned as a valid socket file descriptor

Setup to connect to network...

```
network_connect:
    ;;; Setup the argument to connect() and call connect() ;;;

    ; Fill in the sockaddr_in structure with the
    ; network family, port, and IP address information,
    ; along with the zeros in the zero field.
    mov edi, serverSockaddr ;move server socket address into edi

    ; Store the network family, AF_INET = 2
    mov al, 2   ;move 2 into a lower
    stosb   ;Store AL at address ES eDI
    mov al, 0   ;mov zero into a lower
    stosb   ;store AL at address ES DI

    ; Store the port, in network byte order (big endian).

    ; High byte first
    mov ax, [ipPort]    ;move ipPort variable into ax
    ; Truncate the lower byte
    shr ax, 8   ;shift right ax by 8 bits
    stosb   ;store AL at address ES DI

    ; Low byte second
    mov ax, [ipPort]    ;move ipPort variable value into az
    stosb   ;store byte from al into di
```

```
; Otherwise, print error creating socket and quit.
push msgErrorSocket ;push message error socket on to the stack
push msgErrorSocketLen  ;push message length
call cWriteString   ;call function to write the string
add esp, 8  ;remove top 8 bytes from the stack
call cExit  ;call function to end the program
```

Error message and QUIT program

```
; Store the 4 octets of the IP address, reading from the
; ipOctets 4-byte array and copying to the respective
; locations in the serverSockaddr structure.
mov esi, ipOctets   ;move the ipOctets variable value into esi
; movsb * 4 = movsd
movsd   ;Move Scalar Double-Precision Floating-Point Value

; Zero out the remaining 8 bytes of the structure
mov al, 0   ;move zero into a lower
mov ecx, 8  ;move 8 into ecx
rep stosb   ;repeat string operation unto remaining bytes are zeroed ou

; Setup the array that will hold the arguments for connect
; we are passing through the socketcall() system call.
mov edi, connectArgs    ;move connectArgs variable value into edu
; sockfd
mov eax, [sockfd]   ;move the address of sockfd into eax
stosd   ;store eax register at edi
; Pointer to serverSockaddr structure
mov eax, serverSockaddr ;move serverSockAddr variable into eax
stosd   ;store eax register in edi
; serverSockaddrlen
mov eax, serverSockaddrLen  ;move serverSockaddrLen variable into eax
stosd   ;store eax register in edi

; Syscall socketcall(3, ...); for connect();
mov eax, 102    ;move 102 into eax
mov ebx, 3  ;move 3 into ebx
mov ecx, connectArgs    ;move connectArgs variable contents into ecx
int 0x80    ;call kernel
```

```
; Check if connect() returned a success
    cmp eax, 0  ;compare eax to zero
    jge network_setup_file_descriptors
    ;if eax is greater than zero jump to network_setup_file_descriptors
```

Check to see if connect() returns a success...

YES, eax is greater than zero (network setup file descriptor...)

NO, eax is less than or equal to zero (print error and QUIT)

```
; Check if connect() returned a success
cmp eax, 0  ;compare eax to zero
jge network_setup_file_descriptors
;if eax is greater than zero jump to network_setup_file_descriptors
```

Start of network_setup_file_descriptors...

Socket connection error and QUIT

```
; Otherwise, print error creating socket and quit.
push msgErrorConnect    ;push connection error message
push msgErrorConnectLen ;push message length
call cWriteString   ;call function to write string to stdout
add esp, 8  ;remove top 8 bytes from stack pointer
jmp network_premature_exit  ;jump to network_premature_exit
```

```
network_premature_exit:
network_close_socket:
    ; Syscall close(sockfd);
    mov eax, 6          ;move 6 into eax
    mov ebx, [sockfd]   ;move sockfd into ebx
    int 0x80            ;call the kernetl

    call cExit          ;call function to end the program
;
; cExit
;   Exits program with the exit() syscall.
;       arguments: none
;       returns: nothing
;
cExit:
    ; Syscall exit(0);
    mov eax, 1          ;mov 1 into eax
    mov ebx, 0          ;mov 0 into ebx
    int 0x80            ;call kernel
    ret                 ;return
```

Premature exit+close socket (QUIT)

```
network_setup_file_descriptors:
    ;;; Clear the read fd array, add stdin and the socket fd to the
    ;;; array. ;;;

    ; Point edi to the beginning of the read file descriptor array
    mov edi, masterReadFdArray  ;move start of masterReadFdArray into edi

    ; Zero out all 128 bytes of the read file descriptor array
    mov al, 0   ;move zero into al
    mov ecx, readFdArrayLen ;move reaad FdArrayLen into ecx
    rep stosb   ;repeat string operation until bytes are zeroed out

    ; Add stdin, file descriptor 0, to the read file descriptor array
    mov edi, masterReadFdArray  ;mov masterReadFdArray into edi
    mov al, 1   ;move 1 into al
    stosb   ;store al address at edi

    ; Reset edi to the beginning of the read file descriptor array
    mov edi, masterReadFdArray  ;move beginning of masterReadFdArray into edi
    ; Copy the value of the socket file descriptor to eax
    mov eax, [sockfd]   ;copy sockfd variable into eax

    ; Divide eax by 8, so we can find the offset from the beginning of
    ; the file descriptor array, so we can set the necessary bit for
    ; the socket file descriptor in the read file descriptor array.
    shr eax, 3  ;shift right eax by 3 bits
    ; Increment the pointer by the offset
    add edi, eax    ;add eax to edi

    ; Make another copy of the socket file descriptor in ecx
    mov ecx, [sockfd]   ;copy sockfd into ecx
    ; Isolate the bit offset
    and cl, 0x7 ;perform bitwise and with 0x7 on cl
    ; Left shift a 1 to make a bit mask at that bit offset
    mov al, 1   ;move 1 into al
    shl al, cl  ; shift left al by value in cl
    ; Bitwise OR the bit high at correct bit position in the array
    or [edi], al    ;bitwise or of edi address by value in a
```

Starting bulk of network_setup_file_descriptors

```
network_read_write_loop:
    ; Copy over the master read file descriptor array to the
    ; checking read file descriptor array, which we will pass
    ; to select and check which file descriptors are set/unset.
    mov edi, checkReadFdArray   ;move checkReadFdArray start to edi
    mov esi, masterReadFdArray  ;move masterReadFdArray start to esi
    mov ecx, readFdArrayLen ;move readFdArrayLen to ecx
    rep movsb   ;repeat move bytes at address DS:(E)SI to address ES:(E)DI

    ; Syscall select(sockfd+1, readFdArray, 0, 0, 0);
    ; nfds, the first argument of select, is the highest
    ; file descriptor + 1, in our case it would be sockfd+1,
    ; since stdin is always file descriptor 0.
    mov eax, 142    ;move 142 into eax
    mov ebx, [sockfd]   ;move address of sockfd variable into ebx
    inc ebx ;increment ebx one byte
    mov ecx, checkReadFdArray
    ;move checkReadFdArray into exx
    mov edx, 0  ;move zero into edx
    mov esi, 0  ;move zero into esi
    mov edi, 0  ;move 0 into edi
    int 0x80    ;call kernel
```

Start of network_read_write_loop: ...continued

```
; Check the return value of select for errors
cmp eax, 0  ;compare eax to zero

jg check_read_file_descriptors  ; if eax greater than zero jump to check_read_file_descriptors
```

Checking the return value of sys_select 142 for errors…

YES, eax is greater than zero move to (check_read_file_descriptors)

NO, eax is less than or equal to zero (Print error calling and QUIT)

Check_read_file_descriptor and check_stdin_file_descriptor…proceed.

```
check_read_file_descriptors:
check_stdin_file_descriptor:
    ;;; Check if the stdin file descriptor is set ;;;

    ; Read the first byte (where the first bit, stdin, will be
    ; located) of the updated file descriptor array
    mov esi, checkReadFdArray   ;move checkReadFdArray into esi
    lodsb   ;Load byte at address DS:ESI into AL
    ; Mask the first bit in the array
    and al, 0x01    ;and value at al with 0x01
    ; Check if it is set
    cmp al, 0x01    ;compare al to 0x01
    jne check_socket_file_descriptor    ;jump if not equal to check_socket_file_descriptor
    ; Otherwise, it is set, and we need to read the data into a
    ; buffer, and then write it to the socket
    call cReadStdin ;call function to read from standard input
    call cWriteSocket   ;call function to write socket
```

Network premature exit routine+exit and end the program

```
; Otherwise, print error calling select and quit
push msgErrorSelect ;push error calling select message onto stack
push msgErrorSelectLen  ;push message length
call cWriteString   ;call function to write to standard out
add esp, 8  ;remove 8 bytes from stack pointer
jmp network_premature_exit  ; jump to network_premature_exit routine

network_premature_exit:
network_close_socket:
    ; Syscall close(sockfd);
    mov eax, 6  ;move 6 into eax
    mov ebx, [sockfd]   ;move sockfd into ebx
    int 0x80    ;call the kernetl

    call cExit  ;call function to end the program
;
; cExit
;   Exits program with the exit() syscall.
;       arguments: none
;       returns: nothing
;
cExit:
    ; Syscall exit(0);
    mov eax, 1  ;mov 1 into eax
    mov ebx, 0  ;mov 0 into ebx
    int 0x80    ;call kernel
    ret     ;return
```

```
; cReadStdin
;   Reads from stdin into readBuffer.
;   Sets readBuffLen with number of bytes read.
;       arguments: none
;       returns: number of bytes read on success, -1 on error, in eax
;
cReadStdin:
    ; Syscall read(0, readBuffer, readBufferMaxLen);
    mov eax, 3  ;move 3 into eax
    mov ebx, 0  ;move 0 into ebx
    mov ecx, readBuffer ;move readBuffer into ecx
    mov edx, readBufferMaxLen   ;move readBufferMaxLen into edx
    int 0x80    ;call the kernel

    mov [readBufferLen], eax    ;move eax into readBufferLen variable address
    ret ;return
```

Jump to cReadStdin function; syscall read (0, readBuffer, readBufferMaxLen)

Jump back to main() function where call cWriteSocket is called

```
call cWriteSocket   ;call function to write socket
```

```
; cWriteSocket
;   Writes readBufferLen bytes of readBuff to the socket sockfd.
;       arguments: none
;       returns: number of bytes written on success, -1 on error, in eax
;
cWriteSocket:
    ; Syscall write(sockfd, readBuff, readBuffLen);
    mov eax, 4  ;move 4 into eax
    mov ebx, [sockfd]   ;move sockfd address into ebx
    mov ecx, readBuffer ;move buffer into ecx
    mov edx, [readBufferLen]    ;move address of buffer length into edx
    int 0x80    ;call kernel
    ret ;return
```

After returning to main() check if the socket file descriptor is now set and file descriptor...

```
check_socket_file_descriptor:
    ;;; Check if the socket file descriptor is set ;;;

    ; Reset esi to the beginning of the read file descriptor array
    mov esi, checkReadFdArray ;move start of checkReadFdArray into esi
    ; Copy the value of the socket file descriptor to eax
    mov edx, 0  ;move 0 into edx
    mov eax, [sockfd]   ;move sckfd variable into eax

    ; Divide eax by 8, so we can find the offset from the beginning
    ; of the file descriptor array, so we can set the necessary bit
    ; for the socket file descriptor in the read file descriptor
    ; array.
    shr eax, 3  ;shift right eax by 3 bits
    ; Increment the pointer by the offset
    add esi, eax    ;increment esi by value in eax

    ; Make another copy of the socket file descriptor in ecx
    mov ecx, [sockfd]   ;move sockfd into ecx
    ; Isolate the bit offset
    and cl, 0x7 ;and bitwise operation on cl by 0x7
    ; Left shift a 1 to make a bit mask at that bit offset
    mov bl, 1   ;move 1 into bl
    shl bl, cl  ;shift left bl by value in cl

    ; Read the byte and mask the correct bit for the socket fd
    lodsb   ;Load byte at address DS:ESI into AL
```

```
cmp al, bl  ;compare al to bl
jne check_socket_file_descriptor_done
;if not equal jump to check_socket_file_descriptor_done
```

Ensure correct bit is first set

Non equal values in lower bytes of al and bl registers (Loop back to select () system call to check for more data)

Equal values in lower byte of al and bl (go onto CReadSocket)

```
; Loop back to the select() system call to check for more data
check_socket_file_descriptor_done:
jmp network_read_write_loop ;jump to network_read_write_loop
```

```
; Otherwise, it is set, and we need to read the data into a
; buffer, and then write it to stdout
call cReadSocket    ;call function to read socket
```

```
call cWriteStdout    ;call
function to write to standard out
```

Call function to write the buffer to stdout

Call function to read socket reads from socket sockfd into readBuffer

```
; cReadSocket
;   Reads from the socket sockfd into readBuffer.
;   Sets readBuffLen with number of bytes read.
;       arguments: none
;       returns: number of bytes read on success, -1 on error, in eax
;
cReadSocket:
    ; Syscall read(sockfd, readBuffer, readBufferMaxLen);
    mov eax, 3  ;move 3 into eax
    mov ebx, [sockfd]   ;move contents of sockfd into ebx
    mov ecx, readBuffer ;move readbuffer into ecx
    mov edx, readBufferMaxLen   ;move buffer length into edx
    int 0x80    ;call kernel

    mov [readBufferLen], eax    ;move eav into address of readBufferLen
    ret ;return
```

```
; cWriteStdout:
;   Writes readBufferLen bytes of readBuff to stdout.
;       arguments: none
;       returns: number of bytes written on success, -1 on error, in eax
;
cWriteStdout:
    ; Syscall write(1, readBuffer, readBufferLen);
    mov eax, 4  ;move 4 into eax
    mov ebx, 1  ;mov 1 into abx
    mov ecx, readBuffer ;mov readBuffer into ecx
    mov edx, [readBufferLen]    ;move readBufferLen address into edx
    int 0x80    ;call kernel
    ret ;return
```

Returns to system call function check_socket_file_descriptor_done:

```
; Loop back to the select() system call to check for more data
check_socket_file_descriptor_done:
```