

Forum:[Hack Forums](#)

Thread: [\[SOURCE\]A simple socket server\[/SOURCE\]](#)

arun hacker - January 18, 2017, 12:08 am

Hello Everyone, I've coded a simple program in assembly that listens on port 1337 for incoming connections and send "Hello World" string to the client and after that closes it's connection. I've coded this to show how socket programming is done in assembly. You can further go on learning about network programming in assembly after reading this code.

```
Code ; A simple server that listens on port 1337
; When a client connects on this port
; The server will send a simple String Hello World
; To the client
; To assembly type:
; nasm -f elf32 server.asm
; ld -o server server.o
; ./server
```

```
[SECTION .data]
msg db "Listening for incoming connections...", 0x0, 0xa
msg_len equ $-msg
```

```
final_mesg db "Message sent successfully", 0x0, 0xa
final_mesg_len equ $-final_mesg
```

```
buff db "Hello World", 0x0, 0xa
buff_len equ $-buff
```

```
[SECTION .text]
global _start
```

```
_start:
; int sockfd = socket(AF_INET(2), SOCK_STREAM(1), IPPROTO_IP(0));
; As you can see here the socket function takes 3 argument
; AF_INET(2) is an address family that is used to designate the type of addresses
; that your socket can communicate with( in this case, IPv4 addresses)
; SOCK_STREAM(1) means TCP based communication
; IPPROTO_IP(0)
; We will push the arguments of socket in reverse order into stack
; Because stack follows LIFO rule
```

```
push 0x66
pop eax ; syscall to socket
mov ebx, 1 ; SYS_SOCKET(0x1)
```

```
xor esi, esi ; cleaning up esi register
push esi ; IPPROTO_IP (0x0)
push ebx ; SOCK_STREAM (0x1)
push 0x2 ; AF_INET (0x2)
; ecx needs to hold the pointer to this structure
```

```

mov ecx, esp          ; save pointer to socket() args
int 0x80              ; exec SYS_SOCKET
; After calling SYS_SOCKET it returns a socket descriptor to EAX
; As the subsequent functions rely on this socket descriptor
; we need to save it in an unused register from where we can pull it later
; So we will move our socket descriptor return into EDI register
; In our top of stack there resides 0x2
; so first we will pop that value into edi register
; Because 0x2 is needed for another syscall function
pop edi              ; now edi contains 0x2
xchg edi, eax         ; save result(sockfd) into edi

```

```

;
; bind(sockfd, const struct sockaddr *addr, socklen_t addrlen);
;
xchg ebx, eax         ; sys_bind(0x2)
mov al, 0x66          ; syscall: sys_socketcall
; bind takes three arguments
; first we will create our sockaddr struct
push esi              ; sin_addr = 0 (INADDR_ANY)
push word 0x3905      ; sin_port = 1337 (network byte order)
push word bx          ; sin_family = AF_INET(0x2)
mov ecx, esp          ; save pointer to sockaddr_in struct

```

```

push 0x10             ; addrlen = 16
push ecx              ; struct sockaddr pointer
push edi              ; sockfd
mov ecx, esp          ; save pointer to bind() args
int 0x80              ; exec SYS_BIND

```

```

; Now we need to listen for incoming connection
; listen(sockfd, int backlog);
mov al, 0x66          ; syscall 102
mov bl, 0x4           ; sys_listen
; listen takes 2 arguments
push esi              ; backlog = 0
push edi              ; sockfd
mov ecx, esp          ; save pointer to listen() args
int 0x80              ; exec SYS_LISTEN

```

```

; we will write a simple message on our screen
; to confirm that it is listening for incoming connections
; before that we must backup our register's value to use it later
push eax
push ebx

```

```

mov eax, 4            ; sys_write
mov ebx, 1            ; STDOUT
mov ecx, msg          ; our message
mov edx, msg_len      ; length of our message
int 0x80

```

```

pop ebx

```

```
pop eax
```

```
; Now we need a way to accept incoming connections  
; int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

```
mov al, 0x66          ; syscall : sys_socketcall  
inc ebx               ; sys_accept(0x5)
```

```
push esi              ; addrlen = 0  
push esi              ; addr = 0  
push edi              ; sockfd  
mov ecx, esp          ; save pointer to accept() args  
int 0x80              ; exec sys_accept
```

```
; after sys_accept it will return the client's socket file descriptor  
; We will save it in esi register  
mov esi, eax
```

```
; Now we need to send our message to the client  
mov al, 0x66  
mov ebx, 0x9          ; sys_send
```

```
mov eax, 4            ; SYS_WRITE  
mov ebx, esi          ; write to client fd  
mov ecx, buff         ; our message  
mov edx, buff_len     ; length of our message  
int 0x80              ; sys_write
```

```
; Write our final message on our screen to confirm our message has been sent  
mov eax, 4  
mov ebx, 1  
mov ecx, final_mesg  
mov edx, final_mesg_len  
int 0x80
```

```
exit:  
mov eax, 1  
xor ebx, ebx  
int 0x80
```

Infamia - January 18, 2017, 11:06 am

You prefer NASM over GCC? ew.

Anyway it's fine but your style is just ugly, Assembly has different ways of displaying data i.e hexadecimal, binary etc so you can add it where it's more readable, 0xa is just useless on a declaration because it's not a long type of number nor is it more readable, less even.

but the code should be fine over-all, i haven't tested it because i don't have nasm- but I'll hope it works.

TouchMyMalware - January 18, 2017, 12:13 pm

(Jan 18, 2017 12:06 PM)Infamia Wrote: [\[CITE\]](#) You prefer NASM over GCC? ew.

Anyway it's fine but your style is just ugly, Assembly has different ways of displaying data i.e hexadecimal, binary etc so you can add it where it's more readable, 0xa is just useless on a declaration because it's not a long type of number nor is it more readable, less even.

but the code should be fine over-all, i haven't tested it because i don't have nasm- but i'll hope it works.

Do you prefer GAS syntax over Intel's then?

Infamia - January 18, 2017, 1:14 pm

(Jan 18, 2017 01:13 PM)TouchMyMalware Wrote: [\[CITE\]](#) Do you prefer GAS syntax over Intel's then?

Answered the reason over Discord, I think I biased my question a bit more towards ARM which doesn't have any substantial differences syntax-wise, since all-in-all it is its own syntax, but it more resembles Intel's x86 instruction.

But I'd change my answer thinking about it now since OP uses x86 (from the use of `eax` registers) and not an ARM-based processor.

arun hacker - January 19, 2017, 6:05 am

(Jan 18, 2017 12:06 PM)Infamia Wrote: [\[CITE\]](#) You prefer NASM over GCC? ew.

Anyway it's fine but your style is just ugly, Assembly has different ways of displaying data i.e hexadecimal, binary etc so you can add it where it's more readable, 0xa is just useless on a declaration because it's not a long type of number nor is it more readable, less even.

but the code should be fine over-all, i haven't tested it because i don't have nasm- but i'll hope it works.

Yes I prefer intel syntax over AT&T one.

I believe my code would be ugly for those who doesn't understand assembly well.

Quote:Assembly has different ways of displaying data i.e hexadecimal, binary etc

then what the heck is 0xa, it's the hexadecimal of decimal number 10.

Infamia - January 19, 2017, 11:24 am

(Jan 19, 2017 07:05 AM)arun hacker Wrote: [\[CITE\]](#) Yes I prefer intel syntax over AT&T one.

*read last post

(Jan 19, 2017 07:05 AM)arun hacker Wrote: [\[CITE\]](#) I believe my code would be ugly for those who doesn't understand assembly well.

I understand Assembly well enough to read somebody's code; not to the point where I can tell you what it does, but I can tell you what each command specifically does- ignoring definitions and other non-standard code. Your code is just not pretty to look

at. Albeit ASM itself is kind of weird; but your code is just jumbled up.

(Jan 19, 2017 07:05 AM)arun hacker Wrote: [\[CITE\]](#) then what the heck is 0xa, it's the hexadecimal of decimal number 10.

?

My point on that is saying,

Code `mov rax, 0xFFFFFFFF`
Is waaay better code than;

Code mov rax, 4294967295

In example so to contradict hexadecimal's usefulness and show where it's useless, here's a time you just used hexadecimal on a number that was completely futile:

```
Code mov ebx, 1          ; SYS_SOCKET(0x1)
```

```
[other code]
```

```
push 0x2                ; AF_INET (0x2)
```

It seems like you just completely pasted 75% of the code off some other site and wrote 25% of it by hand. But I won't complain as I said, the code most likely works- but I simply don't like the formatting because it uses preferentially incompatible data types in forth of place wherein other data types completely parallel the same output but have simply excess characters.

Masm4ever - January 20, 2017, 2:38 am

Let me guess: mov eax, 4 is your idea (uppercase SYS_WRITE); same for mov ebx, esi

Can you explain in simple words the role of the first two lines?

Code ; Now we need to send our message to the client

```
mov al, 0x66
```

```
mov ebx, 0x9            ; sys_send
```

```
mov eax, 4              ; SYS_WRITE
```

```
mov ebx, esi            ; write to client fd
```

```
mov ecx, buff           ; our messge
```

```
mov edx, buff_len       ; length of our message
```

```
int 0x80                ; sys_write
```

TouchMyMalware - January 20, 2017, 8:53 am

(Jan 20, 2017 03:38 AM)Masm4ever Wrote: [\[CITE\]](#) Let me guess: mov eax, 4 is your idea (uppercase SYS_WRITE); same for mov ebx, esi

Can you explain in simple words the role of the first two lines?

Don't bother. The entire code he posted was disgusting and useless anyway. Did you see any stack cleanup happening? I didn't.

CPLUSPLUSBOTNET - January 20, 2017, 4:52 pm

(Jan 20, 2017 03:38 AM)Masm4ever Wrote: [\[CITE\]](#) Let me guess: mov eax, 4 is your idea (uppercase SYS_WRITE); same for mov ebx, esi

Can you explain in simple words the role of the first two lines?

Code ; Now we need to send our message to the client

```
mov al, 0x66
```

```
mov ebx, 0x9            ; sys_send
```

```
mov eax, 4              ; SYS_WRITE
```

```
mov ebx, esi            ; write to client fd
```

```
mov ecx, buff           ; our messge
```

```
mov edx, buff_len       ; length of our message
```

```
int 0x80                ; sys_writecode obfsucation, duh.
```