Linux Services

Linux Operating System services

Disk and file system management

Screen display and printing

Keyboard entry

Other I/O management

Date and time

Program run and terminate

Command arguments

Linux System Calls in Assembly Programs

System call

Standard function to perform standard **SERVICE**

Function written in 8086 instructions and included with DOS

Services are complex and detailed ⇒ functions are long

```
; code segment
Access memory locations          ; x86 instructions
Access register locations        ; x86 instructions
ALU operations                   ; x86 instructions
Read data from disk file         ; Linux system call
Write data to disk file          ; Linux system call
Read data from keyboard          ; Linux system call
Display data on screen           ; Linux system call
Finish execution                 ; Linux system call
```

Calling Linux System Functions

General procedure

1. Write required parameters to registers
2. **EAX ← FUNCTION CODE NUMBER**

   Code number identifies specific system call
3. Interrupt **`INT 80h`**

   Calls Linux function handler

   Linux reads code number and runs service function

   Linux returns control to user program
4. Check exit conditions

   Error codes

   Returned values

Basic Linux System Calls

Code	Function
1	Exit - terminate current process
3	Read from file or device
4	Write to file or device
5	Open a file or device
6	Close a file or device
8	Create a file

File

Named data structure in file system directory

Device

Hardware device represented as file — keyboard, display, network card, ...

Exit

exit (terminate process)	
Function	**EAX ← 1**
Entry parameters	**EBX ← exit code**
Exit parameters	None

Example

```
MOV EAX, 1
MOV EBX, 0
INT 80h
```

File Modes

Mode

File access permissions

Read / Write / Execute

Defined separately for User / Group / Other

Example

Output of **`ls -l`** is **`rwxrw-r--`**

`rwx`	**`rw-`**	**`r--`**
User	Group	Other
Read / Write / Execute	Read / Write	Read Only

Changing File Modes

Change mode

Grant group execute rights	`chmod g+x filename`
Deny other read rights	`chmod o-r filename`
New permissions	`rwxrwx---`

Octal form

4 = read

2 = write

1 = executable

Set permissions	`chmod 764 filename`
User	`4 + 2 + 1 = 7`
Group	`4 + 2 = 6`
Other	`4`
New permissions	`rwxrw-r--`

Create File

`creat`	
Function	`EAX ← 8`
Entry parameters	`EBX ← pointer to ASCIIZ pathname` `ECX ← mode (octal file permissions)`
Exit parameters	`EAX ←` { `integer file descriptor (success)`; `negative integer (error)` }

Success ⇒ new file open for writing

File descriptor

Refers to open file for reading/writing

Index to file in system-wide table of open files

Create Example

```
section .data
   filename db '/home/user/name.txt', 00
   filehandle dd 0
section .text
global _start
_start:
     MOV       EAX, 8                ; creat
     MOV       EBX, filename         ; pointer to file path
     MOV       ECX, 777q             ; full permissions
     INT       80h                   ; call Linux
     CMP       EAX, 0
     JL        stop                  ; stop on error
     MOV       [filehandle], EAX
     ;; continue
stop:  MOV      EAX, 1               ; exit
     INT      80h
```

Open File

open	
Function	`EAX ← 5`

Entry parameters	`EBX ← pointer to ASCIIZ pathname`
	`ECX ← flags (opening mode)`
	`EDX ← mode (file permissions)`

Exit parameters	`EAX ←` { `integer file descriptor (success)`; `negative integer (error)` }

Flags
 	
Binary **OR** of octal values
	`0 = read only  1 = write only  2 = r/w`
	`0100 = create if not exist`
	`(EDX ignored if 0100 not set)`
	`0200 = do not create if file exists`
	`01000 = truncate to 0 length if exists`
	`(if opened for write)`
	`02000 = append write to end of file`

creat same as **open** with `flags = 1 + 0100 + 01000`	

Open Example

```
section .data
   filename db '/home/user/name.txt', 00
   filehandle dd 0
section .text
global _start
_start:
     MOV       EAX, 5                ; open
     MOV       EBX, filename         ; pointer to file path
     MOV       ECX, 02001q           ; open for appending
     INT       80h                   ; call Linux
     CMP       EAX, 0
     JL        stop                  ; stop on error
     MOV       [filehandle], EAX
     ;; continue
stop:  MOV      EAX,1
     INT      80h
```

File Error Numbers

1	Operation not permitted
2	No such file or directory
4	Interrupted system call
5	I/O error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file number
12	Out of memory
13	Permission denied
14	Bad address
16	Device or resource busy

17	File exists
20	Not a directory
21	Is a directory
22	Invalid argument
23	File table overflow
24	Too many open files
26	Text file busy
27	File too large
28	No space left on device
29	Illegal seek
30	Read-only file system

Write to File

`write`	
Function	`EAX ← 4`

Entry parameters	`EBX ← file descriptor`
	`ECX ← pointer to output buffer`
	`EDX ← number of bytes to write`

Exit parameters	`EAX ← number of bytes actually written` `if EAX < EDX device may be full`

Write Example

```
section .data
    filename db '/home/user/name.txt', 00
    filehandle dd 0
    str_1: db 0,0,0,0,0,0,0,0,0,0,0,0,0
section .text
global _start
_start:
     /* open file for append and save filehandle */
     MOV       EAX, 4                ; write
     MOV       EBX, [filehandle]     ; pointer to file
     MOV       ECX, str_1            ; pointer to output buffer
     MOV       EDX, 13               ; write 13 bytes
     INT       80h                   ; call Linux
     CMP       EAX, EDX
     JL        stop                  ; stop on full device
     ;; continue
```

Read from File

`read`	
Function	**EAX ← 3**
Entry parameters	**EBX ← file descriptor** **ECX ← pointer to input buffer** **EDX ← number of bytes to read**
Exit parameters	**EAX ← number of bytes actually read** **if EAX < EDX at end of file (EOF)**

Read Example

```
section .data
    filename db '/home/user/name.txt', 00
    filehandle dd 0
    str_1: db "hello, world",0x0a
section .text
global _start
_start:
     /* open file for reading and save filehandle */
     MOV       EAX, 3                ; read
     MOV       EBX, [filehandle]     ; pointer to file
     MOV       ECX, str_1            ; pointer to output buffer
     MOV       EDX, 13               ; write 13 bytes
     INT       80h                   ; call Linux
     CMP       EAX, EDX
     JL        stop                  ; stop on EOF
     ;; continue
```

Close File

`close`	
Function	**EAX ← 6**
Entry parameters	**EBX ← file descriptor**
Exit parameters	**EAX ← { integer file descriptor (success); negative integer (error)**

Close Example

```
section .data
   filename db '/home/user/name.txt', 00
   filehandle dd 0
section .text
global _start
_start:
    MOV      EAX, 5                ; open
    MOV      EBX, filename         ; pointer to file path
    MOV      ECX, 02001q           ; open for appending
    INT      80h                   ; call Linux
    CMP      EAX, 0
    JL       stop                  ; stop on error
   ;; file read / write operations
    MOV      [filehandle], EAX
    MOV      EAX, 6                ; close
    MOV      EBX, [filehandle]     ; pointer to file
    INT      80h                   ; call Linux
stop:  MOV     EAX,1
    INT     80h
```

stdin / stdout

Standard input (`stdin`)

Default input device

Usually keyboard

Standard output (`stdout`)

Default output device

Usually screen

File descriptor 0

Points to **`stdin`** as open file

Read from **`stdin`** permits reading from keyboard

File descriptor 1

Points to stdout as open file

Write to **`stdout`** permits printing to screen

Hello World — Version 1

```
section .data
     str_1:    db "hello, world",0x0a,0
section .text
global _start
_start:
       mov ESI, str_1          ; ESI points to string
L1:    cmp byte [ESI], 0       ; test for end of string
       JZ stop                 ; end loop on 0
       mov EAX, 4              ; write
       mov EBX, 1              ; write to stdout
       mov ECX, ESI            ; pointer to buffer
       mov EDX, 1              ; display 1 byte
       INT 80h
       INC ESI                 ; ESI++
       JMP L1
stop:  MOV EAX, 1
       MOV EBX, 0
       INT 80h
```

Copy from Keyboard to Screen

```
section .data
     str_1:  times 256 db 0           ; 256 byte buffer
section .text
global _start
_start:
        mov EAX, 3      ; read
        mov EBX, 0      ; read from stdin
        mov ECX, str_1  ; pointer to buffer
        mov EDX, 256    ; read 256 bytes or until <ENTER>
        INT 80h
        mov EDX, EAX    ; bytes to display = number of bytes read
        mov EAX, 4      ; write
        mov EBX, 1      ; write to stdout
        mov ECX, str_1  ; pointer to buffer
        INT 80h
stop:   MOV EAX, 1
        MOV EBX, 0
        INT 80h
```

NASM Macros

Aliasing Constants

`label EQU expression`

Assigns alias **`label`** for numerical constant **`expression`**

`expression` may include pointer arithmetic

`label` cannot be changed after assignment

`$ operator`

Last assigned address + 1 = next unused address

Example

```
        cr EQU 0x0d             ; carriage return
        lf EQU 0x0a             ; line feed
    section .data               ; size(data section) = 64 bytes
        s1 db 'this is a string',cr,lf   ; 18 bytes
        len EQU $-s1                     ; len = 18
        times 64-$+s1 db 0               ; 64 - 18 = 46
```

Equivalent to

```
    section .data
        s1 db 'this is a string',0x0d,0xa
        times 46 db 0
```

EQU Example

```
section .data
        message1 db 'hello, world'
        message2 db '123456789'
        here equ $
        ; $ = &('9')+1
        ; here = numerical constant <– value &('9')+1
        msglen2 equ here-message2
        ; message2 = address of '1' = &('1')
        ; msglen2 = &('9')+1 – &('1')
        ;          = length of string message2
        message3 db 'goodbye, world'
section .text
        mov eax, message1       ; pointer to message1
        mov ebx, here           ; pointer to message3
        mov ecx, msglen2        ; length of message1
        mov eax, 1
        INT 80h
```

Single Line Macros

%define preprocessor directive

Expanded at assembly time

```
%define a(x) 1+b(x)
%define b(x) 2*x
%define linux INT 80h

section .data
var     resb    64
section .text
        MOV EAX, var
        MOV EDX, EAX
        mov eax,a(8)    ; a(8) = 1+b(8) = 1+2*8 = 17 = 11 h
        MOV EAX, 1
        linux
```

Multi–Line Macros

Defined with %macro for n parameters

```
%macro prologue 2    ; macro prologue accepts 2 parameters
     push ebp
     mov ebp, esp
     sub esp, %1     ; %1 = parameter 1
     add esi, %2     ; %2 = parameter 2
%endmacro
```

Example

```
section .text
    prologue 4,2
    MOV EAX, 1
    INT 80h
```



```
section .text
    push ebp
    mov ebp, esp
    sub esp, 4
    add esi, 2
    MOV EAX, 1
    INT 80h
```

Hello World — Version 2

```
section .data
     str_1:    db "hello, world",0x0a,0
     len EQU $-str_1
section .text
global _start
_start:
       mov EAX, 4              ; write
       mov EBX, 1              ; write to stdout
       mov ECX, str_1          ; pointer to buffer
       mov EDX, len            ; display len bytes
       INT 80h
stop:  MOV EAX, 1
       MOV EBX, 0
       INT 80h
```

lseek

lseek	
Function	**EAX ← 19 (decimal)**
Entry parameters	**EBX ← file descriptor** **ECX ← pointer offset (bytes to move)** **EDX ← whence** **0 – offset to start of file (SOF)** **1 – offset to current pointer** **2 – offset to end of file (EOF)** **pointer > EOF** No immediate change in file size Writing to **pointer** > **EOF** File size = new **pointer** = new **EOF** Old **EOF** to new data filled with **0**
Exit parameters	**EAX ← new offset (bytes) from start of file**

Overwrite File Data

```
section .data
          s1: db 'ABCDE'
          filename db 'name.txt', 0
section .bss
          b1: resb 6
          fp: resd 1
section .text
global _start
_start:
open:     MOV EAX, 5           ; open
          MOV EBX, filename
          MOV ECX, 2           ; r/w
          INT 80h
          CMP EAX, 0
          JL end               ; on error
          MOV [fp], EAX        ; descriptor
read:     MOV EAX, 3           ; read
          MOV EBX, [fp]        ; descriptor
          MOV ECX, b1          ; buffer
          MOV EDX, 5           ; 5 bytes
          INT 80h
          MOV EBX, b1
          MOV byte [EBX+5],10 ; linefeed
reset:    MOV EAX, 19          ; lseek
          MOV EBX, [fp]        ; descriptor
          MOV ECX, 0           ; offset
          MOV EDX, 0           ; SOF
          INT 80h
```

```
write:    MOV EAX, 4           ; write
          MOV EBX, [fp]        ; descriptor
          MOV ECX, s1          ; string
          MOV EDX, 5           ; 5 bytes
          INT 80h
close:    MOV EAX, 6           ; close
          MOV EBX, [fp]        ; descriptor
          INT 80h
s_write:  MOV EAX, 4           ; write
          MOV EBX, 1           ; stdout
          MOV ECX, b1          ; string
          MOV EDX, 6           ; 6 bytes
          INT 80h

end:      MOV EAX, 1
          MOV EBX, 0
          INT 80h
```

HEX to Decimal

Algorithm

X = integer in hexadecimal representation

Decimal representation of **X**

$$X = N_{n-1} \times 10^{n-1} + N_{n-2} \times 10^{n-2} + \ldots + N_1 \times 10^1 + N_0 \times 10^0$$

To find coefficients N_{n-1} , N_{n-2} , … , N_1 , N_0

$$X \% 10 = N_0$$

$$INT[X/10] = N_{n-1} \times 10^{n-2} + N_{n-2} \times 10^{n-3} + \ldots + N_1 \times 10^0$$

Continue until **INT[X/10] = 0**

Example

102h / 0ah = 19h

19h / 0ah = 2h

2h / 0ah = 0

102h % 0Ah = 8

19 % 0Ah = 5

2h % 0Ah = 2

$102h = 258_{10}$

Program Functions

Get 16-bit number from keyboard

Linux calls to read characters and store in buffer

Enter 1234

Convert ASCII characters to hexadecimal integer values

0 – 9 codes are **30** to **39**

A – F codes are **41** to **46** (upper case)

a – f codes are **61** to **66** (lower case)

Buffer = **31 32 33 34** → **01 02 03 04**

Convert hex characters to a hexadecimal value

$$\text{Value} = 01 \times (10h)^3 + 02 \times (10h)^2 + 03 \times (10h)^1 + 04 \times (10h)^0$$

$$= 1 \times (2)^{12} + 2 \times (2)^8 + 3 \times (2)^4 + 4$$

$$= 1234h$$

Convert hex value to decimal digits = 04 06 06 00

Convert decimal digits to ASCII character buffer 34 36 36 30

Print character buffer

Program Setup

```
; equates, defines, and macros
     NL EQU 10              ; 10 = 0x0a = new line character
     %define   linux int 80h  ; call linux
     %macro stop 0
          mov eax, 1
          mov ebx, 0
          int 80h
     %endmacro
     %macro disp 2          ; display macro with 2 parameters
          mov eax, 4
          mov ebx, 1
          mov ecx, %1       ; parameter 1 = pointer to buffer
          mov edx, %2       ; parameter 2 = number of bytes
          int 80h
     %endmacro
     %macro read 2          ; keyboard macro with 2 parameters
          mov eax, 3
          mov ebx, 0
          mov ecx, %1       ; parameter 1 = pointer to buffer
          mov edx, %2       ; parameter 2 = number of bytes
          int 80h
     %endmacro
```

Get Number From Keyboard — ASCII to Hexadecimal

```
section .data
          numbuff   times 4 db 0   ; for keyboard input
          number    times 2 db 0   ; for input value
          buffer    times 5 db 0   ; for result
          d_buffer  times 1 db 0   ; for output

section .text
global _start
_start:   read numbuff, 4          ; 4 bytes from keyboard
asc2hex:  mov ecx,4                ; 4 characters
          mov esi, numbuff         ; ESI ← pointer
          mov edi, esi             ; EDI ← pointer
L1:       lodsb                    ; AL ← character byte
          cmp al,0x40              ; handles A to F
          jl dec_num               ; if 0 ≤ character ≤ 9 jump
          and al,0xDF              ; insure upper case
          sub al,0x37              ; 41 → 0A, 42 → 0B, … , 46 → 0F
          jmp replace
dec_num:  sub al,0x30              ; 30 to 39 → 0 to 9
replace:  stosb                    ; replace character byte
          loop L1                  ; continue for 4 passes on loop
```

Convert Hex Characters to Hexadecimal Value

```
        xor edx, edx          ; zero accumulator
        mov cx, 0x0C          ; 16^3 = (2^4)^3 = 2^12    12 = 0Ch
        mov esi, numbuff      ; ESI ← pointer to digits
L2:     lodsb                 ; AL ← [ESI]  ESI++
        mov ah,0              ; AX = digit value
        shl ax, cl            ; AX ← AX * 2^CX  (CX = 0C, 08, 04)
        add dx, ax            ; add result to accumulator
        sub cx, 4             ; CX ← CX - 4
        cmp cx, 0             ; continue if CX > 0
        jg L2
        lodsb                 ; get low order digit
        mov ah, 0
        add dx, ax            ; add result to accumulator
        mov [number], dx      ; store result
```

Convert Hex Value to Decimal Digits (in ASCII)

```
        xor edi, edi                ; zero EDI
        mov ecx, 10                 ; CX ← 0Ah
        mov ax, [number]            ; hex value to AX
go:     xor edx, edx                ; zero EDX
        div cx                      ; AX ← AX / 10
                                    ; DX ← AX % 10
        add dl,0x30                 ; num to ascii (add 30h)
        mov [buffer + edi], dl      ; remainder to buffer
        inc di                      ; advance buffer index
        cmp ax, 0                   ; stop if AX == 0
        jne go                      ; continue

        ; At end of loop, di = number of digits + 1
```

Display Character Buffer

```
more:
        dec di                          ; di ← number of digits
        mov si, [buffer + edi]          ; si ← high order digit
        mov [d_buffer], si
        disp d_buffer, 1                ; call Linux to display 1 byte
        cmp di,0                        ; continue to low end of buffer
        jg more                         ; stop after display for di = 0
        mov byte [d_buffer], 10         ; new line
        disp d_buffer, 1                ; call Linux to display 1 byte
        stop                            ; call Linux to exit
```

Calculate x^2 — 1

```
section .bss
    ; buffers
section .text
    ; index
    ; call move to buffer
    ; arithmetic
    ; call move to buffer
    ; convert to decimal
    ; call print
    ; end
    ; move to buffer routine
    ;       move to buffer
    ;       call print
    ;       return
```

Calculate x^2 — 2

```
      stdout EQU 1
      %macro disp 2              ; display macro with 2 parameters
         mov eax, 4
         mov ebx, 1
         mov ecx, %1     ; parameter 1 = pointer to buffer
         mov edx, %2     ; parameter 2 = number of bytes
         int 80h
      %endmacro
section .data
section .bss
      a_buff: resb 4
      f_buff: resb 2
section .text
global _start
_start:
      mov esi, 1         ; index variable
L0:   mov eax, esi       ; eax <-- esi
      call ax2b          ; copy eax to a_buff
      mul eax            ; edx.eax <-- eax*eax
      call ax2b          ; copy ax to a_buff
      ;
```

Calculate x^2 — 3

```
print_decimal:
      mov edi,3                  ; count down index
      mov ecx,10                 ; ECX <-- 0Ah
go:   xor edx,edx                ; zero EDX
      div ecx                    ; EAX <-- EAX / 10
                                 ; EDX <-- EAX % 10
      add dl,0x30                ; num to ascii (add 30h)
      mov [a_buff + edi], dl     ; put remainder at end of buffer
      dec edi                    ; decrease buffer index
      cmp eax,0                  ; stop if AX == 0
      jne go                     ; continue
      mov byte [f_buff], 10      ; new line
      disp a_buff, 5             ; call screen display function
      inc esi                    ; index++
      cmp esi, 16
      jle L0                     ; end at index 17
end:  MOV EAX, 1                 ; exit
      MOV EBX, 0
      INT 80h
```

Calculate x^2 — 4

```
ax2b:   mov edi, a_buff         ; pointer to ascii buffer
        mov ecx, 4              ; 4 iteration loop
L1:     rol ax, 4               ; put high character in lowest 4 bits
        push eax                ; save value of eax
        and ax, 0x000f          ; zero all but lowers 4 bits
        cmp al, 0x0a            ; check if al > 9
        jl L2
        add al, 0x07            ; convert A - F to ascii
L2:     add al, 0x30            ; convert to ascii
        stosb                   ; write AL to ascii buffer
        pop eax                 ; restore value of eax
        loop L1
        mov word [f_buff], 0x0909       ; 2 tab characters
        ;
        push eax                ; save value of eax
        disp a_buff, 6          ; call function to display to screen
        pop eax                 ; restore value of eax
        ;
        RET
```

Debugging with **debug** — 1

```
section .data
     str: db "Hello, world",10
     here EQU $
     str_len EQU here-str
section .text
global _start
_start:
     mov esi, 0                    ; index into string
     mov eax, 4                    ; Linux print function
     mov ebx, 1                    ; stdout
     mov edx, 1                    ; print 1 byte
```

Prepare print function

```
L1:  lea ecx, [esi+str]            ; ECX <-- str + ESI
     int 80h
     inc esi
     cmp esi, str_len
     jle L1
```

Loop: advance string pointer on each iteration

```
mov eax,1
mov ebx,0
int 80h
```

Assemble, link, run **hello.asm**

```
~/asm$ hello
H~/asm$
```

But it **SHOULD** work!

Debugging with **debug** — 2

```
[ Trace ] - ver 0.3 Beta
Copyleft (C) 2001-2005 Amjads. Made in Taiwan

EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFCD7EE0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480C0 Flags=00000246 PF ZF IF

080480C0 BE00000000              MOV     ESI,00000000
-u
080480C0 BE00000000              MOV     ESI,00000000
080480C5 BB01000000              MOV     EBX,00000001
080480CA BA01000000              MOV     EDX,00000001
080480CF B804000000              MOV     EAX,00000004
080480D4 8D8EF4900408            LEA     ECX,[ESI+080490F4]
080480DA CD80                    INT     80
080480DC 46                      INC     ESI
080480DD 81FE0D000000            CMP     ESI,0000000D
-u
080480E3 7EEA                    JLE     080480CF
080480E5 B801000000              MOV     EAX,00000001
080480EA BB00000000              MOV     EBX,00000000
080480EF CD80                    INT     80
```

Debugging with **debug** — 3

```
080480C0 BE00000000              MOV     ESI,00000000
-t

EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFCD7EE0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480C5 Flags=00000246 PF ZF IF

080480C5 BB01000000              MOV     EBX,00000001
-t

EAX=00000000 EBX=00000001 ECX=00000000 EDX=00000000 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFCD7EE0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480CA Flags=00000246 PF ZF IF

080480CA BA01000000              MOV     EDX,00000001
-t

EAX=00000000 EBX=00000001 ECX=00000000 EDX=00000001 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFCD7EE0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480CF Flags=00000246 PF ZF IF

080480CF B804000000              MOV     EAX,00000004
-t

EAX=00000004 EBX=00000001 ECX=00000000 EDX=00000001 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFCD7EE0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480D4 Flags=00000246 PF ZF IF

080480D4 8D8EF4900408            LEA     ECX,[ESI+080490F4]
```

Debugging with **debug** — 4

```
080480D4 8D8EF4900408          LEA     ECX,[ESI+080490F4]
-t

EAX=00000004 EBX=00000001 ECX=080490F4 EDX=00000001 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFE756E0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480DA Flags=00000246 PF ZF IF

080480DA CD80                  INT     80
-t
H
EAX=00000001 EBX=00000001 ECX=080490F4 EDX=00000001 ESI=00000000 EDI=00000000
SS =0000007B ESP=BFE756E0 EBP=00000000 DS =0000007B ES =0000007B FS =00000000
GS =00000000 CS =00000073 EIP=080480DC Flags=00000246 PF ZF IF

080480DC 46                    INC     ESI
```

Why did **EAX** change???

Oh. **`INT 80h`** returns **`EAX <— number of bytes actually written`**

I guess it **SHOULDN'T** work.

But why doesn't the assembler tell me I made a mistake???

Answer: the assembler thinks I know what I'm doing.

Debugging with **debug** — 5

```
section .data
     str: db "Hello, world",10
     here EQU $
     str_len EQU here-str
section .text
global _start
_start:
     mov esi, 0                    ; index into string
     mov ebx, 1                    ; stdout
     mov edx, 1                    ; print 1 byte
L1:  mov eax, 4                    ; Linux print function
     lea ecx, [esi+str]            ; ECX <-- str + ESI
     int 80h
     inc esi
     cmp esi, str_len
     jle L1
```

Prepare print parameters

Loop: reset **EAX** and advance string pointer on each iteration

```
mov eax,1
mov ebx,0
int 80h
```

Assemble, link, run **hello.asm**

```
~/asm$ hello
Hello, world
~/asm$
```

Now it works!