✓کارنیل، بزرگترین شبکه موفقیت ایرانیان می باشد، که افرادی زیادی توانسته اند با آن به موفقیت برسند، فاطمه رتبه ۱۱ کنکور کارشناسی، محمد حسین رتبه۶۸ کنکور کارشناسی، سپیده رتبه ۳ کنکور ارشد، مریم و همسرش راه اندازی تولیدی مانتو، امیر راه اندازی فروشگاه اینترنتی،کیوان پیوستن به تیم تراکتور سازی تبریز، میلاد پیوستن به تیم صبا، مهسا تحصیل در ایتالیا، و..... این موارد گوشه از افرادی بودند که با کارنیل به موفقیت رسیده اند، شما هم می توانید موفقیت خود را با کارنیل شروع کنید.

برای پیوستن به تیم کارنیلی های موفق روی لینک زیر کلیک کنید.

www.karnil.com

همچنین برای ورود به کانال تلگرام کارنیل روی لینک زیر کلیک کنید.

https://telegram.me/karnil

چگونه یک بارکننده خود راه انداز بنویسیم
How To Program a Bootstrap Loader

دلایل بسیار زیادی برای نوشتن یک bootstrap وجود دارد.مهمترین دلیل برای برنامه نویسی یک bootstrap جدید برای یک برنامه نویس بالا بردن آگاهی خودش در رابطه با چگونگی کارکردن کامپیوتر در سطوح پایین تر میباشد(منظور از نظر سخت افزاری و نرم افزاری و نه از نظر سطح دانش)
از دیگر دلایل میتوان به RECOVERY کردن فایلهای گم شده و یا نوشتن یک سیستم عامل جدید اشاره کرد.
برای اینکار احتیاج دانستن دقیق چگونگی کارکرد BIOS میباشد و اینکه چگونه باعث لود شدن یک سیستم عامل میشود.
Bootstrap یک برنامه بسیار کوچکی میباشد که توسط BIOS(Basic Input Output System) هنگامی که کامپیوتر را روشن میکنیم بعد از مرحله POST(Power On Self Test) به داخل RAM بارگزاری میشود.اینکار به این دلیل انجام میشود که چون BIOS هیچگونه اطلاعی از سیستم عامل مورد نظر ما ندارد در واقع وقتی شما یک کامپیوتر را میخرید میتوانید روی آن هرگونه سیستم عاملی را که میخواهید نصب کرده و از آن استفاده کنید و به همین دلیل BIOS به گونه ای طراحی شده است که هیچگونه دخل و تصرفی در سیستم عامل کاربر نداشته باشد در واقع BIOS بدنبال یک آدرس مشخص بر روی هارد دیسک میگردد.که آن آدرس برنامه bootstrap میباشد و وظیفه BIOS فقط لود کردن آن برنامه برروی RAM بوده و بعد از اجرای bootstrap این برنامه وظیفه اجرای سیستم عامل رو بعهده دارد (اینکار به این دلیل انجام میشود که امکان سیستم عامل مورد نظر وظایف خاصی را دنبال کرده و یا برروی هاردیسکی غیر از هارددیسک اول نصب شده باشد و به همین دلیل پیدا کردن و اجرا کردن هسته سیستم عامل بعهده bootstrap میباشد) .
Bootstrap از اولین sector هارددیسک اول و از track صفر و headصفر و sector یک آن هارددیسک خوانده میشود و اینکه bootstrap از کدام دستگاه لود شود بستگی به تنظیمات شما در BIOS دارد که کدام دستگاه را بعنوان اولین مرجع برای bootstrap انتخاب کرده باشید.
هنگام لود شدن bootstrap تنها ۵۱۲ بایت از اولین سکتور خوانده شده و به داخل رم بارگزاری میشود که این ۵۱۲ بایت در آدرس فیزیکی 0000:7C00 در رم قرار میگیرد و بعد از آن BIOS ۲ بایت آخر فایل bootstrap را که در رم قرار دارد را امتحان میکند (واقع در افست 1FEh) برای پیدا کردن مقدار AA55h با پیدا کردن این مقدار BIOS متوجه میشود که هارددیسک موردنظر bootable بوده.
Bootstrap حتماً باید ۵۱۲ بایت طول داشته باشد بعلت اینکه فقط بر روی یک سکتور مینشیند و ۲ بایت آخر آن توسط BIOS چک میشود. بعد از این اینکه BIOS ۲ بایت مورد نظر را میبابد به آدرس 0000:7C00 رفته و کنترل را به bootstrap واگذار میکند.
این نکته بسیار مهمی برای برنامه نویسان میباشد که بدانند در حالت واقعی Real Mode پردازنده یا همان CPU فقط و فقط در حالت ۱۶ بیت کار میکند و باید تمامی ثباتهای segment آماده شده باشد و قطعات کدها نباید بیش از ۶۴ کیلوبایت باشد.
Bootstrap در حالت کلی حالت پردازنده را از ۱۶ بیت به ۳۲ بیت تغییر نمیدهد ولی این دلیل نمیشود که نتوان به وسیله آن اینکار را نکرد.
قسمت دوم bootstrap که کارهای دیگری بغیر از بار کردن سیستم عامل را برعهده دارد را نمیتوان در آن ۵۱۲ بایت ذخیره کرد.
یک bootstrap ساده میتواند مانند زیر باشد:

```
;*************************************
     [BITS 16]
     ORG     0
     INT     0x18

     TIMES   510-($-$$) DB 0
     DW      0xAA55
  ;*************************************
```

از این به بعد تمامی کدهایی که شما میبینید در فایلی به نام bootstrap.asm ذخیره میگردد.این مثال میتوان بوسیله NASM ترجمه شده و به کد ماشین تبدیل گردد این کد باید بدین صورت دخیره گردد:

Nasm –O bootstrap.asm bootstrap.bin

و حالا ما باید بتوانیم که فایل compile شده خود را بر روی track صفر و head صفر وsector اول بنویسیم که ساده ترین روش استفاده از debug میباشد (البته توجه کنید بوسیله debug شما فقط میتوانید برروی floppy disk و یا hard disk اینکار را انجام داده پیشنهاد میشود برای امتحان اینکار بر روی hard diskخود قبل از اینکار از آن یک پشتیبان تهیه کنید).

Debug bootstrap.bin
-W 100 0 0 1
-Q

توضیح کد:

بوسیله دستور w میتوانیم فایلی را که به داخل حافظه لود کرده ایم در جایی که میخواهیم بنویسم که در اینجا ما میخواهیم فایلمان را در track صفر ،head صفر ، sector یک بنویسیم که در debug نیازی به مشخص کردن track و head نیست و بجای آن شماره دیسک و بعد از آن سکتور مورد نظر و بعد از آن شماره آن را مشخص میکنیم در واقع در این دستور 1 0 0 به این معنا میباشد که فایل را در فلاپی (اولین صفر) در سکتور صفر به شماره یک آن کپی کن. شماره اول یعنی 100 به این معنی هست که فایل ما در آدرس ۱۰۰ رم ذخیره شده ما فقط آدرس شروع را به debug میدهیم.

برای اطلاعات بیشتر میتوانید به راهنمای debug مراجعه کنید.

در اکثر سیستم عاملهای unix based میتوانید از دستور زیر استفاده کنید:

dd if=bootstrap.bin of=/dev/fd0

برای اطلاعات بیشتر میتوانید در unix خود دستور زیر را اجرا کرده

Man dd

برنامه های Bootstrap میتوانند دارای قابلیتهای دیگری بغیر از بوت کردن سیستم را بعهده گیرند.اینگونه برنامه ها قابلیت برپایی و راه اندازی بعضی از سخت افزارها،تغییر حالت پردازنده به حالت محافظت شده و یا بکارگیری یک وظیفه خاص برروی پردازنده.

به هرحال برنامه های bootstrap میتوانند فایلهای بزرگتراز ۵۱۲ بایت و با قابلیتهای بسیار بیشتری را به داخل حافظه بارگزاری کنند.

دو تکنیک متفاوت برای پیاده سازی اینکار وجود دارد(فایلهای بزرگتر از ۵۱۲ بایت) .

۱- اولین روش واقف بودن به این نکته میباشد که بدانیم چقدر سکتور باید در حافظه لود شود و اینکه این مقدار سکتور چقدر طول میکشد که به داخل حافظه برود و بتوان کنترل را به آن فایلهایی که در حافظه لود شده است منتقل کرد و این متد بطور معمول سیستم فایل جاری برروی دیسک را از بین میبرد و باعث خرابی هزاران فایلی میشود که بر روی سیستم فایل جاری موجود میباشد به این دلیل که برنامه bootstrap که نوشته ایم احتیاج بیشتر از ۵۱۲ بایت داشته و به این دلیل که ما از سیستم فایل جاری استفاده نکرده در واقع خودمان دوباره چرخ را اختراع کرده به همین دلیل این کار باعث بهم ریختگی در سیستم فایل جاری میشود.

۲- راه پیشرفته تر و همچنین سخت تر و پیچیده تر و در آخر مورد اطمینان تر اینست که سیستم فایل جاری را شناسایی کرده و بتوانیم در ساختار آن (منظور تمامی دایرکتوریها آن سیستم فایل) حرکت کرده و به bootstrap اجازه دهیم که فایل مورد نظر را از روی همین سیستم فایل شناسایی کرده و برروی حافظه لود کند.

FAT12 سیستم فایل مورد استفاده در دیسکها نرم (floppy disk) میباشد دو data block میباشد که از FAT12 ارث برده اند یکی رشته OEM_ID و دیگری پارامترهای بلوک BIOS میباشد.
رشته OEM_ID هیچگونه کاری بغیر از اینکه دیسکت با کدام نرم افزار format شده است را انجام نمیدهد پارامترهای بلوک BIOS که در مستندات Microsoft بعنوان BPB از آن یاد شده است یک رکورد میباشد که حاوی فیلدهایی است که مشخصات فیزیکی دیسک را توضیح میدهد.
این مشخصات میتواند برای شناسایی خواص دیسک مانند حجم کلی دیسکت بکار گرفته شود.
BIOS سریعاً شروع میکند به اجرای کدی که در آدرس 0000:7C00 وجود دارد که حاوی این بلاک داده ها میباشد(منظور OEM_ID,BPB میباشد) بنابراین اولین کاری که bootstrap انجام میدهد اجرای یک دستور JMP میباشد که بعد از این اطلاعات میباشد.
دسترسی به فایل ثانویه ای که کارهای بیشتری انجام احتیاج به این دارد که بفهمیم FAT چگونه کار میکند.
FAT12 دیسک را بصورت آرایه های ترتیبی سازماندهی میکند و به هر سلول داده کلاستر(cluster) گفته میشود بنابراین با توجه به چگونگی format یک دیسک تعداد سکتورهای بر روی هر cluster میتواند متغیر باشد. ساختاری که به عنوان FAT(File Allocation Table) شناخته میشود trackهای هر cluster را برروی دیسک نگهداری میکند.FAT خودش حاوی یک مقدار صحیح میباشد که آن آدرس به کلاستر بعدی اشاره میکند با دنبال کردن این زنجیره اعداد ما به علامت EOF(End Of File) خواهیم رسید در واقع فایلها در سیستم فایل FAT اینگونه ذخیره میشوند.
FAT12 یک ساختار اضافه که به آن Root Directory گفته میشود را داراست که در واقع وظیفه اندیس گذاری فایلهای را در اولین cluster بر عهده دارد.
با کنار هم قرار دادن تمامی اینها میتوان به مهم دستیافت که برای دسترسی به یک فایل ابتدا نام آنرا در Root Directory جستجو کرده و بعد زنجیره اعداد داخل FAT را دنبال کرده تا بتوان فایل را خواند.

SOURCE CODE

کد زیر یک نمونه ای است که چگونگی خواندن یک فایل ثانویه را از روی Root Directory نشان میدهد و اینکه میتوان آنرا اجرا کرد.

```
;*********************************************************************
*****
[BITS 16]
ORG 0
jmp     START

OEM_ID                  db "QUASI-OS"
BytesPerSector          dw 0x0200
SectorsPerCluster       db 0x01
ReservedSectors         dw 0x0001
TotalFATs               db 0x02
MaxRootEntries          dw 0x00E0
TotalSectorsSmall       dw 0x0B40
MediaDescriptor         db 0xF0
SectorsPerFAT           dw 0x0009
SectorsPerTrack         dw 0x0012
NumHeads                dw 0x0002
HiddenSectors           dd 0x00000000
TotalSectorsLarge       dd 0x00000000
DriveNumber             db 0x00
Flags                   db 0x00
Signature               db 0x29
VolumeID                dd 0xFFFFFFFF
VolumeLabel             db "QUASI  BOOT"
SystemID                db "FAT12   "
```

```
START:
; code located at 0000:7C00, adjust segment registers
     cli
     mov     ax, 0x07C0
     mov     ds, ax
     mov     es, ax
     mov     fs, ax
     mov     gs, ax
; create stack
     mov     ax, 0x0000
     mov     ss, ax
     mov     sp, 0xFFFF
     sti
; post message
     mov     si, msgLoading
     call    DisplayMessage
LOAD_ROOT:
; compute size of root directory and store in "cx"
     xor     cx, cx
     xor     dx, dx
     mov     ax, 0x0020                          ; 32 byte directory
entry
     mul     WORD [MaxRootEntries]               ; total size of
directory
     div     WORD [BytesPerSector]               ; sectors used by
directory
     xchg    ax, cx
; compute location of root directory and store in "ax"
     mov     al, BYTE [TotalFATs]                ; number of FATs
     mul     WORD [SectorsPerFAT]                ; sectors used by
FATs
     add     ax, WORD [ReservedSectors]          ; adjust for
bootsector
     mov     WORD [datasector], ax               ; base of root
directory
     add     WORD [datasector], cx
; read root directory into memory (7C00:0200)
     mov     bx, 0x0200                          ; copy root dir
above bootcode
     call    ReadSectors
; browse root directory for binary image
     mov     cx, WORD [MaxRootEntries]           ; load loop counter
     mov     di, 0x0200                          ; locate first root
entry
.LOOP:
     push    cx
     mov     cx, 0x000B                          ; eleven character
name
     mov     si, ImageName                       ; image name to find
     push    di
rep  cmpsb                                       ; test for entry
match
     pop     di
     je      LOAD_FAT
     pop     cx
     add     di, 0x0020                          ; queue next
directory entry
     loop    .LOOP
     jmp     FAILURE
LOAD_FAT:
```

```
; save starting cluster of boot image
     mov     si, msgCRLF
     call    DisplayMessage
     mov     dx, WORD [di + 0x001A]
     mov     WORD [cluster], dx                  ; file's first
cluster
; compute size of FAT and store in "cx"
     xor     ax, ax
     mov     al, BYTE [TotalFATs]                ; number of FATs
     mul     WORD [SectorsPerFAT]                ; sectors used by
FATs
     mov     cx, ax
; compute location of FAT and store in "ax"
     mov     ax, WORD [ReservedSectors]          ; adjust for
bootsector
; read FAT into memory (7C00:0200)
     mov     bx, 0x0200                          ; copy FAT above
bootcode
     call    ReadSectors
; read image file into memory (0050:0000)
     mov     si, msgCRLF
     call    DisplayMessage
     mov     ax, 0x0050
     mov     es, ax                              ; destination for
image
     mov     bx, 0x0000                          ; destination for
image
     push    bx
LOAD_IMAGE:
     mov     ax, WORD [cluster]                  ; cluster to read
     pop     bx                                  ; buffer to read
into
     call    ClusterLBA                          ; convert cluster to
LBA
     xor     cx, cx
     mov     cl, BYTE [SectorsPerCluster]        ; sectors to read
     call    ReadSectors
     push    bx
; compute next cluster
     mov     ax, WORD [cluster]                  ; identify current
cluster
     mov     cx, ax                              ; copy current
cluster
     mov     dx, ax                              ; copy current
cluster
     shr     dx, 0x0001                          ;divide by two
     add     cx, dx                              ; sum for (3/2)
     mov     bx, 0x0200                          ; location of FAT in
memory
     add     bx, cx                              ; index into FAT
     mov     dx, WORD [bx]                       ; read two bytes
from FAT
     test    ax, 0x0001
     jnz     .ODD_CLUSTER
.EVEN_CLUSTER:
     and     dx, 0000111111111111b               ; take low twelve
bits
    jmp     .DONE
.ODD_CLUSTER:
     shr     dx, 0x0004                          ; take high twelve
bits
```

```
.DONE:
     mov     WORD [cluster], dx                 ; store new cluster
     cmp     dx, 0x0FF0                         ; test for end of
file
     jb      LOAD_IMAGE
DONE:
     mov     si, msgCRLF
     call    DisplayMessage
     push    WORD 0x0050
     push    WORD 0x0000
     retf
FAILURE:
     mov     si, msgFailure
     call    DisplayMessage
     mov     ah, 0x00
     int     0x16                               ; await keypress
     int     0x19                               ; warm boot computer

;*************************************************************************
*****
; PROCEDURE DisplayMessage
; display ASCIIZ string at "ds:si" via BIOS
;*************************************************************************
*****
DisplayMessage:
     lodsb                                      ; load next
character
     or      al, al                             ; test for NUL
character
     jz      .DONE
     mov     ah, 0x0E                           ; BIOS teletype
     mov     bh, 0x00                           ; display page 0
     mov     bl, 0x07                           ; text attribute
     int     0x10                               ; invoke BIOS
     jmp     DisplayMessage
.DONE:
     ret

;*************************************************************************
*****
; PROCEDURE ReadSectors
; reads "cx" sectors from disk starting at "ax" into memory location
"es:bx"
;*************************************************************************
*****
ReadSectors:
.MAIN
     mov     di, 0x0005                         ; five retries for
error
.SECTORLOOP
     push    ax
     push    bx
     push    cx
     call    LBACHS
     mov     ah, 0x02                           ; BIOS read sector
     mov     al, 0x01                           ; read one sector
     mov     ch, BYTE [absoluteTrack]           ; track
     mov     cl, BYTE [absoluteSector]          ; sector
     mov     dh, BYTE [absoluteHead]            ; head
     mov     dl, BYTE [DriveNumber]             ; drive
     int     0x13                               ; invoke BIOS
```

```
     jnc     .SUCCESS                          ; test for read
error
     xor     ax, ax                            ; BIOS reset disk
     int     0x13                              ; invoke BIOS
     dec     di                                ; decrement error
counter
     pop     cx
     pop     bx
     pop     ax
     jnz     .SECTORLOOP                       ; attempt to read
again
     int     0x18
.SUCCESS
     mov     si, msgProgress
     call    DisplayMessage
     pop     cx
     pop     bx
     pop     ax
     add     bx, WORD [BytesPerSector]         ; queue next buffer
     inc     ax                                ; queue next sector
     loop    .MAIN                             ; read next sector
     ret

;*************************************************************************
*****
; PROCEDURE ClusterLBA
; convert FAT cluster into LBA addressing scheme
; LBA = (cluster - 2) * sectors per cluster
;*************************************************************************
*****
ClusterLBA:
     sub     ax, 0x0002                        ; zero base cluster
number
     xor     cx, cx
     mov     cl, BYTE [SectorsPerCluster]      ; convert byte to
word
     mul     cx
     add     ax, WORD [datasector]             ; base data sector
     ret

;*************************************************************************
*****
; PROCEDURE LBACHS
; convert "ax 2; LBA addressing scheme to CHS addressing scheme
; absolute sector = (logical sector / sectors per track) + 1
; absolute head   = (logical sector / sectors per track) MOD number
of heads
; absolute track  = logical sector / (sectors per track * number of
heads)
;*************************************************************************
*****
LBACHS:
     xor     dx, dx                            ; prepare dx:ax for
operation
     div     WORD [SectorsPerTrack]            ; calculate
     inc     dl                                ; adjust for sector
0
     mov     BYTE [absoluteSector], dl
     xor     dx, dx                            ; prepare dx:ax for
operation
     div     WORD [NumHeads]                   ; calculate
```

```
     mov     BYTE [absoluteHead], dl
     mov     BYTE [absoluteTrack], al
     ret

absoluteSector db 0x00
absoluteHead   db 0x00
absoluteTrack  db 0x00

datasector  dw 0x0000
cluster     dw 0x0000
ImageName   db "LOADER  BIN"
msgLoading  db 0x0D, 0x0A, "Loading Boot Image ", 0x0D,
0x0A, 0x00
msgCRLF     db 0x0D, 0x0A, 0x00
msgProgress db ".", 0x00
msgFailure  db 0x0D, 0x0A, "ERROR : Press Any Key to Reboot", 0x00

     TIMES 510-($-$$) DB 0
     DW 0xAA55
;*************************************************************************
```

داخل کد:

کد بالا یک طراحی آبشاری دارد. در آخر لیست چهار تابع پایه وجود دارد که از تکرار نوشتن کد اضافه جلوگیری میکند.
DisplayMessage باعث بکارگیری یکسری از روالهای BIOS شده بابت چاپ وضعیت کار بر روی مانیتور برای اطلاع کاربر.
ReadSectors باعث بکارگیری روالهای BIOS برای خواندن اطلاعات خام از دیسکت بر روی حافظه میشود.
ClusterLBA روالی که Microsoft's Cluster addressing scheme را به Logical Block Addressing تبدیل میکند برای اینکه فایل را بتوان بر روی دیسک map کرد LBACHS ، Logical Block Address را تبدیل میکند به ساختار Sector،Head،Cylinder که بوسیله BIOS قابل فهم هست.

بدنه اصلی برنامه که بوسیله برچسب START مشخص شده است رجیسترهای پردازنده را آماده میکند.
این قسمت یکی از مهمترین قسمتهای کد میباشد که باعث برپایی محیط سیستم عامل میشود مقادیری که در درون ثباتها در CPU در اول کار وجود دارند امکان دارد باعث انجام یکسری اعمال ناخواسته شود هنگامی که در حال کار با توابع BIOS هستیم.ولی هنگامی که CPU در حالت شناخته شده ای میباشد در واقع هنگامی که مقادیر ناخواسته در ثباتها نمیباشد کد ما میتواند جایگاه Root Directory را از BPB پیدا کرده و آنرا محاسبه کند و آنرا به داخل RAM بار کند.
قسمت دیگر استفاده از BPB برای پیدا کردن FAT و بار کردن آن به داخل RAM بابت جستجو در آن میباشد استفاده از اولین cluster برای loader.bin ، bootstrap سیستم فایل FAT را جستجو کرده و یک ارجاع به تابع ReadSectors میدهد که بتواند فایل loader.bin را به داخل حافظه بار کند و در آخر عملیات کنترل به loader.bin داده میشود البته بوسیله عملگر RETF

پایان
خواهشمند است پیشنهادات و انتقادات خود را به آدرس: netspc@gmail.com ارسال فرمایید.
اسفند ۱۳۸۴
May 2006

TR@NSL@T!0N BY netspc
TH@NX TO _LOVE_CODER_ F0R @DVICE
TH@NX TO MASTER F0R SUPP0RT