بَرْمَجَة وَ تَصْمِيمْ نظَام تَشْغِيلْ
"نظام إقرأ"

أحمد بن عصام عبد الرحيم أحمد

٢٣ مايو ٢٠١٠

المحتويات

الأمثلة التوضيحية

قائمة الأشكال

القسم I.

الأساسيات Basics

١. مقدمة عن أنظمة التشغيل وتاريخ ظهورها

٢. معمارية حواسيب x86

القسم II.

إقلاع الحاسب Booting

٣. إقلاع الحاسب ومحمل النظام Bootloader

أحد أهم الأساسيات في برمجة نظام تشغيل هي كتابة محملٌ له ، هذا المحمل يعمل على نسخ نواة النظام من أحد الأقراص الى الذاكرة الرئيسية ثم ينقل التنفيذ الى النواة ، وهكذا تنتهي دورة عمل المحمل ويبدأ نظام التشغيل متمثلا في النواة بالبدء بتنفيذ الاوامر والمهمات وتلبية إحتياجات المستخدم. في هذا الفصل سندرس كيفية برمجة المحمل وماهيته وسيتم الاقلاع من قرص مرن بنظام FAT12 ، فالغرض هذه المرحلة هو دراسة أساسيات المحمل وتحميل وتنفيذ نواة مبسطة ، وفي الفصل الثالث سنعود مجددا الى الحديث عن أنظمة الملفات.

١.٣. إقلاع الحاسب

إقلاع الحاسب (Boot-Strapping) هي أول خطوة يقوم بها الجهاز عند وصله بالكهرباء لتحميل نظام التشغيل، وتبدأ هذه العملية مباشرة عند الضغط على مفتاح التشغيل في الحاسب ، حيث ترسل إشارة كهربائية[١] الى اللوحة الام (MotherBoard) والتي تقوم بتوجيهها الى وحدة موزد الطاقة (Power Supply Unit). بعد ذلك يأتي دور وحدة PSU لكي تقوم بمهمة تزويد الحاسب وملحقاته بالكمية المطلوبة من الطاقة، وإرسال اشارة Power Good إلى اللوحة الام وبالتحديد الى نظام ال BIOS . تدل هذه الاشارة على أنه تم تزويد الطاقة الكافية ، وفورا سيبدأ برنامج الفحص الذاتي (Power on Self Test) الذي يعرف اختصاراً ب POST بفحص أجهزة ومحلقات الحاسب (مثل الذاكرة ولوحة المفاتيح والماوس والناقل التسلسلي ...الخ) والتأكد من أنها سليمة. بعدها يقوم ال POST بنقل التحكم الى نظام ال BIOS حيث سيقوم ال POST بتحميل ال BIOS الى نهاية الذاكرة 0xFFFF0 و سيقوم أيضا بوضع تعليمة قفز (jump) في أول عنوان في الذاكرة الى نهاية الذاكرة ، كذلك من مهام ال POST هي تصفير المسجلين CS:IP وهذا يعني أن أول تعليمية سينفذها المعالج هي تعليمة القفز الى نهاية الذاكرة وبالتحديد الى ال BIOS . يستلم ال BIOS التحكم ويبدأ في انشاء جدول المقاطعات (Interrupt Vector Table) وتوفير العديد من المقاطعات ،ويقوم بالمزيد من عمليات الفحص والاختبار للحاسب ، وبعد ذلك يبدأ في مهمة البحث عن نظام تشغيل في الاجهزة الموجودة بناءا على ترتيبها في اعدادات ال BIOS في برنامج Setup ،وفي حالة لم يجد ال BIOS جهازا قابلا للاقلاع في كل القائمة فانه يصدر رسالة خطأ بعدم توفر نظام تشغيل ويوقف الحاسب عن العمل (Halt) ، وفي حالة توفر جهازاً قابلاً للإقلاع سيقوم ال BIOS

[١] هذه الإشارة تحوي على بت (bit) تدل قيمته اذا كانت 1 على أنه تم تشغيل الحاسب.

بتحميل القطاع الأول منه (يحوي هذا القطاع على برنامج المحمل) الى الذاكرة الرئيسية وبالتحديد الى العنوان الفيزيائي 0x07c00 وسيُنقَل التنفيذ الى المحمل.

خلال هذه المهمة (اقلاع النظام) يوفر لنا نظام ال BIOS العديد من المقاطعات على جدول المقاطعات والذي يتم انشائه بدءاً من العنوان 0x0 ، هذه المقاطعات هي خدمات يوفرها لنا نظام البايوس لاداء وظيفة معينة مثل مقاطعة لطباعة حرف على الشاشة. واحدة من أهم المقاطعات التي يستخدمها نظام البايوس للبحث عن جهاز الاقلاع هي المقاطعة int 0x19 حيث تكمن وظيفتها في البحث عن هذا الجهاز ومن ثم تحميل القطاع الأول منه الى العنوان الفيزيائي 0x07c00 ونقل التنفيذ اليه . طريقة البحث والتحميل ليست بالامر المعقد حيث على هذه المقاطعة البحث في أول قطاع (من أي جهاز موجود على قائمة الاجهزة القابلة للاقلاع) عن التوقيع 0xAA55 وهي عبارة عن بايتين يجب أن تكون على آخر القطاع الاول تدل على أن هذا الجهاز قابل للاقلاع. ومن الجدير بالذكر أن المقاطعات التي يوفرها لنا نظام البايوس يمكن استخدامها فقط اذا كان المعالج يعمل في النمط الحقيقي Real Mode أما إذا تم تغيير نمط المعالج الى النمط المحمي Protected Mode – كما سنرى ذلك لاحقا– فانه لن يمكن الاستفادة من هذه المقاطعات بل سيتسبب استخدامها في حدوث استثناءات (Exception) توقف عمل الحاسب.

٢.٣. محمل النظام Bootloader

محمل النظام هو برنامج وظيفته الرئيسية هي تحميل نواة نظام التشغيل ونقل التنفيذ اليها.هذا المحمل يجب ان تتوفر فيه الشروط الاتية :

١. حجم البرنامج يجب أن يكون 512 بايت بالضبط.

٢. أن يتواجد على القطاع الأول في القرص : القطاع رقم 1 ، الرأس 0 ، المسار 0 ، وأن يحمل التوقيع المعروف.

٣. أن يحوي شفرة تحميل النواة ونقل التنفيذ اليها.

٤. أن يكون البرنامج object code خالي من أي أضافات (header,symbol table,...etc) وهو ما يعرف أيضا بـــ Flat Binary .

الشرط الأول يُقيد وظيفة المحمل وقدرته على توفير خصائص متقدمة[٢]، حيث أن هذا الحجم لا يكفي لكي يبحث المحمل عن نواة النظام وتمهيد الطريق لها للبدء بتنفيذها ، وبسبب أن النواة ستكون 32-bit فانه يجب تجهيز العديد من الأشياء بدءاً من جداول الواصفات (العامة والخاصة) وتفعيل البوابة A20 وانتهاءاً بتغيير نمط المعالج الى النمط المحمي والقفز الى النواة للمباشرة في تنفيذها . كل ذلك يحتاج الى

[٢]مثل خاصية ال Safe Mode

حجم أكبر من الحجم المشروط لذلك عادة ما يلجأ مبرمجوا المحملات الى تجزئيها على مرحلتين وهو ما يسمى ب Multi-Stage Boot Loader . الشرط الثاني للمحمل وهو أن يتواجد على أول قطاع في القرص وهو يحمل العنوان الفيزيائي التالي:

- القطاع رقم 1
- المسار رقم 0
- الرأس رقم 0

وتحقيقُ هذا الشرط ليس بالأمر المعقد خصوصا مع توفر العديد من الادوات التي تساعد على نسخ مقطع من قرص ما الى مقطع في قرص آخر ، أما الشق الثاني من الشرط فهو متعلق بتمييز القطاع الاول كقطاع قابل للاقلاع من غيره ، حيث حتى يكون القطاع قابلا للاقلاع فانه يجب أن يحمل التوقيع 0xAA55 في البايت رقم 510 و 511 . وبدون هذا التوقيع فان البايوس (وتحديدا مقاطعة رقم 0x19) لن تتعرف على هذا القطاع كقطاع قابلٌ للإقلاع. أما الشرط الثالث فهو شرط اختياري وليس اجباري ، فمن الممكن أن تكون وظيفة المحمل هي عرض رسالة ترحيب فقط ! ولكن في أغلب الحالات الواقعية يجب أن تُحمِّل النواة وتُنفِّذ عن طريق هذا المحمل. وقد أسلفنا وذكرنا أن تحميل نواة 32-bit يختلف عن تحميل نواة 16-bit ، حيث في الاولى يجب تجهيز الطريق أمام النواة وتفعيل بعض الخصائص لذلك وَجب تقسيم مهمة محمل النظام الى مرحليتن – كماسنرى ذلك – ، أما في حالة كانت النواة 16-bit فانه يمكن تحميلها بمرحلة واحدة فقط . والشرط الاخير يتعلق بصيغة الملف التنفيذي للمحمل، حيث أغلب المترجمات تخرج صيغ تنفيذية تحوي على الكثير من المعلومات المضافة من قبله (كصيغ ELF,PE,COFF,...etc) وهذا ما يجعل عملية تنفيذ المحمل وتشغيله من قبل البايوس مستحيلة ، فالبايوس عندما يقرأ محمل النظام الى الذاكرة فانه ينقل التنفيذ الى أول بايت فيه والذي يجب ان يكون قابل للتنفيذ وليس معلومات أو هيدر عن الملف – كما في حالة الصيغ السابق ذكرها– . لذلك يجب أن تكون صيغة المحمل هي عبارة عن الصيغة الثنائية المقابلة للأوامر الموجودة فيه بدون أي اضافات أي Object Code او Flat Binary.

ويجدر بنا الحديث عن لغة برمجة محمِّل النظام، فغالبا تستخدم لغة التجميع (Assembly 16-bit) لأسباب كثيرة ، منها أن الحاسب عندما يبدأ العمل فان المعالج يكون في النمط الحقيقي تحقيقا لأغراض التوفقية (Backward Compatibility) مع الأجهزة السابقة ، أيضا استخدام لغة التجميع 16-bit يجعل من الممكن استدعاء مقاطعات وخدمات البايوس – قبل الانتقال الى بيئة 32-bit – ، أخيراً لا حاجة لملفات وقت التشغيل run-time library ، حيث أن لغة التجميع ماهي الا مختصرات للغة الآلة Machine Language. كل هذا لا يجعل عملية كتابة محمِّل النظام بلغة السي مستحيلا ! فهناك كمٌّ كبير من المحملات تستخدم لغة السي والتجميع في آن واحد (مثل GRUB,NTLDR,LILO...etc) ، لكن قبل برمجة مثل هذه المحملات يجب برمجة بعض ملفات ال run-time لتوفير بيئة لكي تعمل برامج السي عليها ، أيضا يجب كتابة loader لكي يقرأ الصيغة الناتجة من برنامج السي ويبدأ التنفيذ من دالة ال main .

٣.٣. مخطط الذاكرة

أثناء مرحلة الإقلاع وعندما يُنقل التنفيذ الى محمل النظام فان الذاكرة الرئيسية ل

٤.٣. برمجة محمل النظام

المثال ١.٣ يوضح أصغر محمل للنظام يمكن كتابته وتنفيذه ، باستخدام المجمع NASM[٣] وهو مجمع متعدد المنصات ويوفر ميزة انتاج ملفات ثنائية object code .

Listing ٣١.: Smallest Bootloader

```
;Simple Bootloader do nothing.

bits 16       ; 16-bit real mode.

start:        ; label are pointer.
    cli     ; clear interrupt.
    hlt     ; halt the system.

    times 510-($-$$)  db    0   ; append zeros.

    ;   $ is the address of first instruction (should be 0
       x07c00).
    ; $$ is the address of current line.
    ;   $-$$ means how many byte between start and current.

    ; if cli and hlt take 4 byte then time directive will
       fill
    ; 510-4 = 506 zero's.

    ; finally the boot signature 0xaa55
    db    0x55  ; first byte of a boot signature.
    db    0xaa  ; second byte of a  boot signature.
```

[٣] راجع الملحق ا لمعرفة كيفية استخدام المجمع لترجمة المحمل وكيفية نسخه الى floppy disk or CD ليتم القلاع منه سواءاً كان على جهاز فعلي أو على جهاز تخيلي (Virtual Machine) .

وعندما يبدأ الجهاز بالعمل فان البايوس يقوم بنسخ هذا المحمل الى العنوان 0x0000:0x7c00 ويبدأ بتنفيذه ، وفي هذا المثال فان المحمل هذا الذي يعمل في النمط الحقيقي (real mode) لا يقوم بشيء ذو فائدة حيث يبدأ بتنفيذ الامر cli الذي يوقف عمل المقاطعات ، يليها الامر hlt الذي يوقف عمل المعالج وبالتالي يتوقف النظام عن العمل ، وبدون هذا الأمر فان المعالج سيستمر في تنفيذ أوامر لا معنى لها (garbage) والتي ستؤدي الى سقوط (Crash) النظام . وبسبب أن حجم المحمل يجب أن يكون 512 بايت وأن آخر بايتين فيه يجب أن تكونا التوقيع الخاص بالمحمل فانه يجب أن تكون أول 510 بايت ذات قيمة واخر بايتين هما 0xaa55 ، لذلك تم استخدام الموجه times لكي يتم ملئ المتبقي من أول 510 بايت بالقيمة صفر (ويمكن استخدام أي قيمة اخرى) وبعد ذلك تم كتابة التوقيع الخاص بالمحمل وذلك حتى يتم التعرف عليه من قِبل البايوس.

١.٤.٣. عرض رسالة ترحيبية

طالما ما زلنا نعمل في النمط الحقيقي فان ذلك يمكننا من استخدام مقاطعات البايوس ، وفي المثال ٢.٣ تم عرض رسالة باستخدام مقاطعة البايوس int 0x10 الدالة 0xe .

Listing ٣٢.: Hello World

```
;Hello Bootloader

bits 16       ; 16-bit real mode.
org  0x0      ; this number will added to all addresses (
   relocating).

start:
    jmp main  ; jump over data and function to entry point.



; ***************
; data
; ***************

hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam"
   ,0xa,0xd,0

; *****************************************
; puts16: prints string using BIOS interrupt
;   input:
```

```
;       es: pointer to data segment.
;       si: point to the string
; ****************************************

puts16:

    lodsb     ; read character from ds:si to al ,and
       increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe    ; print character routine number.
    int 0x10    ; call BIOS.

    jmp puts16    ; continue prints until 0 is found.

  end_puts16:

    ret


; ************************************
;   entry point of bootloader.
; ************************************

main:

    ;_____________________
    ; intit registers
    ;_____________________

    ; because bootloader are loaded at 0x07c00 we can
       refrence this location with many different
       combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x:07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
       for offset.

    mov ax,0x07c0
```

```
mov ds,ax
mov es,ax

mov si,hello_msg
call puts16

cli     ; clear interrupt.
hlt     ; halt the system.

times 510-($-$$)  db    0   ; append zeros.

; finally the boot signature 0xaa55
db    0x55
db    0xaa
```

النتيجة :

الشيء الملاحظ في المثال السابق هو أن مقطع الكود code segment ومقطع البيانات data segment متواجدان في نفس المكان على الذاكرة (داخل ال 512 بايت) لذلك يجب تعديل قيم مسجلات المقاطع للاشارة الى المكان الصحيح. و بداية نذكر أن البايوس عندما ينقل التنفيذ الى برنامج محمل النظام الذي قمنا بكتابته فانه في حقيقة الأمر يقوم بعملية far jump والتي ينتج منها تصحيح قيم ال cs:ip لذلك لا داعي للقلق حول هذين المسجلين، لكن يجب تعديل قيم مسجلات المقاطع الاخرى مثل ds,es,ss,fs,gs. وكما نعلم أن العنوان الفيزيائي لمحمل النظام هو 0x07c00 يمكن الوصول اليه بأكثر من 4000 طريقة مختلفة ، لكن سوف نقتصر على استخدام 0x07c0:0x0 أو 0x0:0x7c00 نظراً لان هذه هي القيم الفعلية التي تستخدمها البايوس.

وفي حالة استخدام العنونة الاولى فان مسجلات المقاطع يجب أن تحوي القيمة 0x07c0 (كما في المثال اعلاه) أما بقية العنوانين (سواءا للمتغيرات وال label) فانها يجب أن تبدأ من القيمة 0x0، وكما هو معروف ان المجمعات عندما تبدأ في عملية ترجمة الملف الى ملف ثنائي فانها تبدأ بترقيم العناوين بدءاً من العنوان 0x0 لذلك كانت وظيفة الموجه org هي عمل اعادة تعيين (relocating) للعناوين بالقيمة التي تم كتابتها ، وفي المثال أعلاه كانت القيمة هي 0x0 ، أما في حالة استخدام الطريقة الثانية للوصول الى مكان محمل النظام فان مسجلات المقاطع يجب أن تحوي القيمة 0x0 بينما المسجلات الاخرى يجب أن تبدأ قيمها من العنوان 0x7c00 ، وهذا لا يمكن بالوضع الطبيعي لان المجمعات ستبدأ من العنوان 0x0 لذلك يجب استخدام الموجه org وتحديد قيمة ال relocate بالقيمة 0x7c00 .

٣.٤.٢. معلومات قطاع الاقلاع

إضافة الى محمل النظام فان قطاع الإقلاع boot sector يجب أن يحوي كذلك على معلومات تساعد في وصف نظام الملفات المستخدم ووصف القرص الذي سيتم الاقلاع منه ، هذه المعلومات تحوي معرف OEM وتحوي بيانات BIOS Parameter Block (تختصر ب BPB) ويجب أن تبدأ كل هذه البيانات من البايت رقم 3[٤]. وسوف يتم استخدام هذه البيانات بكثرة أثناء تطوير محمل النظام كذلك أحد فوائد هذه البيانات هو تعرف أنظمة التشغيل على نظام الملفات المستخدم في القرص.

Listing ٣٣.: Bios Parameter Block

```
OEM_ID              db      "eqraOS  "    ; Name of your
   OS, Must be 8 byte! no more no less.

bytes_per_sector    dw      0x200    ; 512 byte per
   sector.
sectors_per_cluster db      0x1          ; 1 sector per
   cluster.
reserved_sectors    dw      0x1          ; boot sector is
    reserved.
total_fats          db      0x2          ; two fats.
root_directory      dw      0xe0         ; root dir has
   224 entries.
total_sectors       dw      0xb40        ; 2880 sectors
   in the volume.
media_descriptor    db      0xf0         ; 1.44 floppy
   disk.
sectors_per_fat     dw      0x9          ; 9 sector per
   fat.
sectors_per_track   dw      0x12         ; 18 sector per
   track.
number_of_heads     dw      0x2          ; 2 heads per
   platter.
hidden_sectors      dd      0x0          ; no hidden
   sector.
total_sectors_large dd      0x0

; Extended BPB.
```

[٤]لهذا السبب فان أول تعليمة في المحمل ستكون تعليمة القفز الى الشفرة التنفيذية، وبدون القفز فان المعالج سيبدأ بتنفيذ هذه البيانات باعتبار انها تعليمات وهذا ما يؤدي في الاخر الى سقوط النظام.

```
drive_number          db        0x0
flags                 db        0x0
signature             db        0x29          ; must be 0x28
   or 0x29.
volume_id             dd        0x0           ; serial number
   written when foramt the disk.
volume_label          db        "MOS FLOPPY " ; 11 byte.
system_id             db        "fat12   "    ;  8 byte.
```

المثال ٤.٣ يوضح شفرة المحمل بعد اضافة بيانات OEM and BPB.

Listing ٣٤.: BPB example

```
;Hello Bootloader

bits 16       ; 16-bit real mode.
org  0x0      ; this number will added to all addresses (
   relocating).

start:
    jmp main  ; jump over data and function to entry point.


;*****************************************
; OEM Id and BIOS Parameter Block (BPB)
;*****************************************

; must begin at byte 3(4th byte), if not we should add nop
   instruction.

OEM_ID                db        "eqraOS  "    ; Name of your
   OS, Must be 8 byte! no more no less.

bytes_per_sector      dw        0x200    ; 512 byte per
   sector.
sectors_per_cluster   db        0x1          ; 1 sector per
   cluster.
reserved_sectors      dw        0x1          ; boot sector is
    reserved.
total_fats            db        0x2          ; two fats.
```

```
root_directory      dw      0xe0        ; root dir has
    224 entries.
total_sectors       dw      0xb40       ; 2880 sectors
    in the volume.
media_descriptor    db      0xf0        ; 1.44 floppy
    disk.
sectors_per_fat     dw      0x9         ; 9 sector per
    fat.
sectors_per_track   dw      0x12        ; 18 sector per
    track.
number_of_heads     dw      0x2         ; 2 heads per
    platter.
hidden_sectors      dd      0x0         ; no hidden
    sector.
total_sectors_large dd      0x0

; Extended BPB.

drive_number        db      0x0
flags               db      0x0
signature           db      0x29        ; must be 0x28
    or 0x29.
volume_id           dd      0x0         ; serial number
    written when foramt the disk.
volume_label        db      "MOS FLOPPY " ; 11 byte.
system_id           db      "fat12   "    ;  8 byte.


; ***************
; data
; ***************

hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam"
    ,0xa,0xd,0

; ****************************************
; puts16: prints string using BIOS interrupt
;   input:
;       es: pointer to data segment.
;       si: point to the string
; ****************************************
```

```
puts16:

    lodsb     ; read character from ds:si to al ,and
       increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe    ; print character routine number.
    int 0x10    ; call BIOS.

    jmp puts16    ; continue prints until 0 is found.

  end_puts16:

    ret


; ************************************
;   entry point of bootloader.
; ************************************

main:

    ;_____________________
    ; intit registers
    ;_____________________

    ; because bootloader are loaded at 0x07c00 we can
       refrence this location with many different
       combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
       for offset.

    mov ax,0x07c0
    mov ds,ax
    mov es,ax

    mov si,hello_msg
```

```
call puts16

cli     ; clear interrupt.
hlt     ; halt the system.

times 510-($-$$)  db    0    ; append zeros.

; finally the boot signature 0xaa55
db    0x55
db    0xaa
```

و المخرج ا.١ يوضح الشفرة السابقة في حالة عرضها بأي محرر سادس عشر Hex Editor حيث كما نلاحظ أن بيانات المحمل متداخلة مع الشفرة التنفيذية (تعليمات المعالج) لذلك يجب أن يتم القفز فوق هذه البيانات حتى لا تُنَفذ كتعليمات خاطئة ، كذلك يجب التأكد من آخر بايتين وأنها تحمل التوقيع الصحيح.

Listing ٣٥.: Hex value of bootloader

```
Offset(h) 00 01 02 03 04 05 06 07

00000000  E9 72 00 65 71 72 61 4F  ér.eqraO
00000008  53 20 20 00 02 01 01 00  S .....
00000010  02 E0 00 40 0B F0 09 00  .à.@ð...
00000018  12 00 02 00 00 00 00 00  ........
00000020  00 00 00 00 00 00 29 00  ......).
00000028  00 00 00 4D 4F 53 20 46  ...MOS F
00000030  4C 4F 50 50 59 20 66 61  LOPPY fa
00000038  74 31 32 20 20 20 57 65  t12   We
00000040  6C 63 6F 6D 65 20 74 6F  lcome to
00000048  20 65 71 72 61 4F 53 2C   eqraOS,
00000050  20 43 6F 64 65 64 20 62   Coded b
00000058  79 20 41 68 6D 61 64 20  y Ahmad
00000060  45 73 73 61 6D 0A 0D 00  Essam...
00000068  AC 3C 00 74 07 B4 0E CD  ¬<.t.´.Í
00000070  10 E9 F4 FF C3 B8 C0 07  .Ăéôÿ¸À.
00000078  8E D8 8E C0 BE 3E 00 E8  .Ø.À¾>.è
00000080  E6 FF FA F4 00 00 00 00  æÿúô....
00000088  00 00 00 00 00 00 00 00  ........
                  ...
                  ...
```

```
000001F0   00 00 00 00 00 00 00 00   ........
000001F8   00 00 00 00 00 00 55 AA   ......Uª
```

ويمكن الاستفادة من هذه المحررات والتعديل المباشر في قيم الهيكس للملف الثنائي[٥]، فمثلا يمكن حذف التوقيع واستبداله بأي رقم ومحاولة الإقلاع من القرص ! بالتأكيد لا يمكن الاقلاع بسبب أن البايوس لن يتعرف على القرص بأنه قابل للإقلاع ، كذلك كمثال يمكن عمل حلقة لا نهائية وطباعة الجملة الترحيبية في كل تكرار ، ويجب أولا اعادة تجميع الملف الثنائي باستخدام أي من برامج ال Disassembler وإدخال تعليمة قفز بعد استدعاء دالة طباعة السلسلة الى ما قبلها.

Listing ٣٦.: Some Code

```
;Hello Bootloader

bits 16       ; 16—bit real mode.
org  0x0      ; this number will added to all addresses (
   relocating).

start:
    jmp main  ; jump over data and function to entry point.


;*****************************************
; OEM Id and BIOS Parameter Block (BPB)
;*****************************************

; must begin at byte 3(4th byte), if not we should add nop
   instruction.

OEM_ID              db        "eqraOS  "    ; Name of your
   OS, Must be 8 byte! no more no less.

bytes_per_sector    dw        0x200    ; 512 byte per
   sector.
sectors_per_cluster db        0x1          ; 1 sector per
   cluster.
reserved_sectors    dw        0x1          ; boot sector is
    reserved.
total_fats          db        0x2          ; two fats.
```

[٥] في حالة لم نتمكن من الوصول الى ملف المصدر source code.

```
root_directory      dw      0xe0        ; root dir has
    224 entries.
total_sectors       dw      0xb40       ; 2880 sectors
    in the volume.
media_descriptor    db      0xf0        ; 1.44 floppy
    disk.
sectors_per_fat     dw      0x9         ; 9 sector per
    fat.
sectors_per_track   dw      0x12        ; 18 sector per
    track.
number_of_heads     dw      0x2         ; 2 heads per
    platter.
hidden_sectors      dd      0x0         ; no hidden
    sector.
total_sectors_large dd      0x0

; Extended BPB.

drive_number        db      0x0
flags               db      0x0
signature           db      0x29        ; must be 0x28
    or 0x29.
volume_id           dd      0x0         ; serial number
    written when foramt the disk.
volume_label        db      "MOS FLOPPY " ; 11 byte.
system_id           db      "fat12   "    ;  8 byte.


; ***************
; data
; ***************

hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam"
    ,0xa,0xd,0

; ********************************************
; puts16: prints string using BIOS interrupt
;   input:
;       es: pointer to data segment.
;       si: point to the string
; ********************************************
```

```
puts16:

    lodsb     ; read character from ds:si to al ,and
       increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe    ; print character routine number.
    int 0x10    ; call BIOS.

    jmp puts16    ; continue prints until 0 is found.

  end_puts16:

    ret


; ************************************
;   entry point of bootloader.
; ************************************

main:

    ;_____________________
    ; intit registers
    ;_____________________

    ; because bootloader are loaded at 0x07c00 we can
       refrence this location with many different
       combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
       for offset.

    mov ax,0x07c0
    mov ds,ax
    mov es,ax

    mov si,hello_msg
```

```
call puts16

cli     ; clear interrupt.
hlt     ; halt the system.

times 510-($-$$)  db    0   ; append zeros.

; finally the boot signature 0xaa55
db    0x55
db    0xaa
```

٣.٤.٣. تحميل قطاع من القرص باستخدام المقاطعة int 0x13

بعد أن تم تشغيل محمل النظام لعرض رسالة ترحيبة ، فان مهمة المحمل الفعلية هي تحميل وتنفيذ المرحلة الثانية له حيث كما ذكرنا سابقا أن برمجة محمل النظام ستكون على مرحلتين وذلك بسبب القيود على حجم المرحلة الاولى ، وتكمن وظيفة المرحلة الاولى في البحث عن المرحلة الثانية من محمل النظام ونقل التنفيذ اليها ، وبعدها يأتي دور المرحلة الثانية في البحث عن نواة النظام ونقل التحكم اليها. وسنتناول الان كيفية تحميل مقطع من القرص المرن الى الذاكرة الرئيسية ونقل التحكم اليها باستخدام مقاطعة البايوس int 0x13 وفي الفصل السادس سيتم دراسة الموضوع بالتفصيل عن طريق البرمجة المباشرة لمتحكم القرص المرن controller.

إعادة القرص المرن

عند تكرار القراءة من القرص المرن فانه يجب في كل مرة أن نعيد مكان القراءة والكتابة الى أول مقطع sector في القرص وذلك لكي نضمن عدم حدوث مشاكل، وتستخدم الدالة 0x0 من المقاطعة int 0x13 لهذا الغرض.
المدخلات :

- المسجل ah : 0x0.
- المسجل dl : رقم محرك القرص المرن وهو 0x0.

النتيجة:

- المسجل ah : الحالة.
- CF : 0x1 اذا حدث خطأ ، 0x0 اذا تمت العملية بنجاح.

مثال:

Listing ٣٧.: Some Code

```
reset_floppy:

    mov ah,0x0     ; reset floppy routine number.
    mov dl,0x0     ; drive number

    int 0x13    ; call BIOS

    jc reset_floppy   ; try again if error occur.
```

قراءة المقاطع sectors

أثناء العمل في النمط الحقيقي فاننا سنستخدم مقاطعة البايوس `int 0x13` الدالة `0x2` لقراءة المقاطع (sectors) من القرص المرن الى الذاكرة الرئيسية RAM .
المدخلات :

- المسجل `ah`: الدالة `0x2`
- المسجل `al`: عدد المقاطع التي يجب قرائتها.
- المسجل `ch`: رقم الاسطوانة (Cylinder) ، بايت واحد.
- المسجل `cl`: رقم المقطع ، من البت 0 – 5 ، أما اخر بتين يستخدمان مع القرص الصلب hard disk.
- المسجل `dh`: رقم الرأس.
- المسجل `dl` : رقم محرك القرص المرن وهو `0x0`.
- العنوان `es:bx` : مؤشر الى المساحة التي سيتم قراءة المقاطع اليها.

النتيجة:

- المسجل `ah` : الحالة.
- المسجل `al`: عدد المقاطع التي تم قرائتها.
- `CF` : `0x1` اذا حدث خطأ ، `0x0` اذا تمت العملية بنجاح.

مثال:

Listing ٣٨.: Some Code

```
read_sectors:

  reset_floppy:

    mov ah,0x0    ; reset floppy routine number.
    mov dl,0x0    ; drive number

    int 0x13    ; call BIOS

    jc reset_floppy   ; try again if error occur.

    ; init buffer.
    mov ax,0x1000
    mov es,ax
    xor bx,bx

  read:

    mov ah,0x2    ; routine number.
    mov al,1    ; how many sectors?
    mov ch,1    ; cylinder or track number.
    mov cl,2    ; sector number "fisrt sector is 1 not 0",
       now we read the second sector.
    mov dh,0    ; head number "starting with 0".
    mov dl,0    ; drive number ,floppy drive always zero.

    int 0x13    ; call BIOS.
    jc read     ; if error, try again.

    jmp 0x1000:0x0  ; jump to execute the second sector.
```

٣.٥. مقدمة الى نظام FAT12

نظام الملفات هو برنامج يساعد في حفظ الملفات على القرص بحيث ينشئ لنا مفهوم الملف وخصائصه والعديد من البيانات المتعلقة به من تاريخ الانشاء والوقت ، كذلك يحتفظ بقائمة بجميع الملفات وأماكن تواجدها في القرص ، أيضاً أحد أهم فوائد أنظمة الملفات هي متابعة الأماكن الغير المستخدمة في القرص

والأماكن التي تضررت بسبب أو لآخر bad sectors ، كذلك أنظمة الملفات الجيدة تقوم بعمل تجميع الملفات المبعثرة على القرص Defragmentation حتى تستفيد من المساحات الصغيرة التي ظهرت بسبب حذف ملف موجود أو تخزين ملف ذو حجم أقل من المساحة الخالية. وبدون أنظمة الملفات فان التعامل مع القرص سيكون مستحيلا ! حيث لن نعرف ماهي المساحات الغير مستخدمة من الاخرى ولن نستطيع ان نقوم بقراءة ملف طلبه المستخدم لعرضه على الشاشة !
وبشكل عام فان نظام الملفات يتكون من:

- برنامج للقراءة والكتابة من القرص وسنطلق عليه اسم المحرك (Driver).
- وجود هيكلة بيانات Data Structure معينة على القرص،يتعامل معها درايفر نظام الملفات.

وحيث أن برمجة برنامج القراءة والكتابة تعتمد كلياُ على هيكلة نظام الملفات على القرص ، فاننا سنبدأ بالحديث عنها أولا وسوف نأخذ نظام FAT12 على قرص مرن كمثال ، نظراً لبساطة هذا النظام وخلوه من التعقيدات وفي الفصل الخامس –بإذن الله– سيتم التطرق الى أنظمة ملفات أخرى بالتفصيل.

١.٥.٣. قيود نظام FAT12

يعتبر نظام FAT12 من أقدم أنظمة الملفات ظهوراً وقد انتشر استخدامه في الاقراص المرنة منذ أواخر السبعينات ، ويعيب نظام FAT12 :

- عدم دعمه للمجلدات الهرمية ويدعم فقط مجلد واحد يسمى الجذر Root Directory.
- طول العنقود (Cluster) هو 12 بت ، بمعنى أن عدد الكلسترات هي 2^{12}.
- أسماء الملفات لا تزيد عن 12 بت.
- يستوعب كحد أقصى 4077 ملف فقط.
- حجم القرص يحفظ في 16 بت ، ولذا فانه لا يدعم الاقراص التي حجمها يزيد عن 32 MB.
- يستخدم العلامة 0x01 لتمييز التقسيمات على القرص (Partitions).

وكما ذكرنا أننا سنستخدم هذا النظام في هذه المرحلة نظراً لبساطته ، وعلى الرغم من أنه قد تلاشى استخدامه في هذا الزمن الا انه يعتبر أساس جيد للأنظمة المتقدمة لذا وجب دراسته.

٣.٥.٢. هيكلة نظام FAT12 على القرص

عند تهئية القرص المرن[٦] (Format) بنظام FAT12 فان تركيبة القرص تكون على الشكل التالي:

وأول مقطع هو مقطع الاقلاع (Boot Sector) ويحوي شفرة محمل النظام (المرحلة الاولى) بالاضافة الى بيانات ومعلومات BPB and OEM id ، هذا المقطع عنوانه الفيزيائي على القرص هو : المقطع 1 المسار 0 الرأس 0 وهذا العنوان هو الذي يجب تمرير الى مقاطعة البايوس int 0x13 التي تقوم بالقراءة من القرص كذلك في حالة ما أردنا التعامل المباشر مع متحكم القرص المرن. ونظراً لصعوبة هذه العنونة والتي تعرف ب Absolute Sector فان أنظمة الملفات تتعامل مع نظام عنونة مختلف للوصول الى محتويات القرص ، فبدلاً من ذكر كل من المقطع والمسار والرأس للوصول الى مقطع ما فان هذه العنونة تستخدم فقط رقم للمقطع . نظام العنونة الذي تستخدمه أنظمة الملفات يسمى بالعنونة المنطقية (Logical Sector Addressing) ويختصر ب LBA هو نظام بسيط يعتمد على ترقيم المقاطع بشكل متسلسل بدئاً من مقطع الاقلاع (Boot Sector) والذي يأخذ العنوان 0 ، والمقطع الثاني 1 وهكذا هلم جرا حتى نصل الى آخر مقطع في القرص. وبما أنه يجب استخدام العنونة الحقيقة بدلا من المنطقية لحظة القراءة من القرص (تذكر مقاطعة البايوس int 0x13 والمسجلات التي يجب ادخال قيمها) فانه يجب ايجاد طريقة للتحويل من العنونة الحقيقة الى المنطقية –سنناقش الموضوع لاحقا–. ننتقل الى المقطع التالي لمقطع الإقلاع وهو مقطع (أو عدة مقاطع) يمكن أن يحجزها المبرمج لاداء أي وظيفة يريدها وتسمى المقاطع المحجوزة الاضافية Extra Reserved Sectors ، والمقصود بمحجوزة أي انه لا يوجد لها وجود في دليل FAT ، ومقطع الإقلاع هو مقطع محجوز دائما لذلك كانت قيمة المتغير reserved sectors في معلومات BPB هي واحد ، وفي حالة ما أردت حجز مقاطع أخرى كل ما عليك هو زيادة هذه القيمة بعدد المقاطع المرغوبة ، وللوصول الى محتويات هذا المقطع الاضافي(ان كان له وجود) فان العنوان الحقيقي له هو المقطع 2 المسار 0 الرأس 0 ، أما العنوان المنطقي له هو المقطع 1. وبشكل عام فانه في الغالب لا يتم استخدام مقاطع اضافية سوى مقطع الاقلاع . المقطع الثالث هو جدول FAT ، وهو جدول يحوي سجلات بطول 12 بت عن كل كلستر (Cluster) في القرص ، بيانات هذا السجل توضح ما اذا كان الكلستر قيد الاستخدام أم لا ، وهل هو آخر كلستر للملف أم لا وإذا كان ليس باخر فانه يوضح لنا الكلستر التالي للملف ، ويوضح الشكل التالي تركيبة هذا الجدول

اذاً هذا وظيفة هذا الجدول هي معرفة الكاسترات الخالية من غيرها كذلك الوظيفة الاخرى هي معرفة جميع الكلسترات لملف ما ويتم ذلك بالنظر الى قيمة السجل (قيمة ال 12 بت) ، والقيم هي :

- القيمة 0x00: تدل على أن الكلستر خالي.
- القيمة 0x01 : تدل على أن الكلستر محجوز.

[٦] سواءاً كانت التهئية من قبل درايفر نظام الملفات الذي سنقوم ببرمجته أو كانت من قبل نظام الشتغيل المستخدم أثناء عملية التطوير ، فمثلا في ويندوز يمكن إعادة تهئية القرص المرن بنظام FAT12 .

- القيم من 0x02 الى 0xfef : تدل على عنوان الكلستر التالي (بمعنى آخر أن الكلستر محجوز وتوجد كلسترات متبقية للملف).
- القيم من 0xff0 الى 0xff6: قيم محجوزة.
- القيمة 0xff6 : تدل على Bad Cluster.
- القيم من 0xff8 الى 0xfff: تدل على أن هذا الكلستر هو الاخير للملف.

ويمكن النظر الى جدول FAT بأنه مصفوفة من القيم أعلاه ، وعندما نريد تحميل ملف فاننا سنأتي بعنوان أول كلستر له من جدول Root Directory (سنأتي عليها لاحقا) وبعدها نستخدم عنوان الكلستر ك index الى جدول FAT ونقرأ القيمة المقابله للكلستر ، فاذا كانت القيمة بين 0x02 الى 0xfef فانها تدل على الكلستر التالي للملف ، ومن ثم سنستخدم هذه القيمة أيضا ك index ونقرأ القيمة الجديدة ، ونستمر على هذا الحال الى أن نقرأ قيمة تدل على نهاية الملف. هذا الجدول FAT يبدأ من المقطع المنطقي 1[٧] وطوله 9 مقاطع أي أن نهاية هذا الجدول تكون في المقطع تكون في آخر المقطع 10، ولمعرفة العنوان الحقيقي للمقطع فانه يمكن استخدام بعض المعادلات للتحويل ، والقسم التالي سيوضح ذلك بالاضافة الى شرح مبسط عن هيكلة القرص المرن وكيفية حفظه للبيانات . وبعد جدول FAT توجد نسخة أخرى من هذا الجدول وتستخدم كنسخة احتياطية backup وهي بنفس حجم وخصائص النسخة الاولى ، وبعدها يأتي دليل الجذر Root Directory وهو مصفوفة من 224 سجل كل سجل بطول 32 بايت ، وظيفية هذا الدليل هي حفظ أسماء الملفات الموجودة على القرص المرن بالاضافة الى العديد من المعلومات التي تخص وقت الانشاء والتعديل وحجم الملف وعنوان أول كلستر للملف ، عنوان الكلستر هو أهم معلومة لكي نستطيع تحميل الملف كاملا ، حيث كما ذكرنا أن هذا العنوان سيعمل ك index في جدول FAT وبعدها سنحدد ما اذا كانت توجد كلسترات أخرى يجب تحميلها أم أن الملف يتكون من كلستر واحد. والجدول التالي يوضح محتويات السجل الواحد في دليل ال root directory بداءاً من البايت الاول الى الاخير:

- البايتات 0-7: اسم الملف(وفي حالة كان الحجم أقل من 8 بايت يجب استخدام حرف المسافة لتعبئة المتبقي).
- البايتات 8-10: امتداد الملف(يجب استخدام المسافة أيضا لتعبئة المتبقي).
- البايت 11: خصائص الملف وهي :
  - البت 0: القراءة فقط.
  - البت 1: مخفي.
  - البت 2: ملف نظام.
  - البت3: اسم القرص Volume Label.

[٧] بافتراض الوضع الغالب وهو عدم وجود مقاطع إضافية باستثناء مقطع الإقلاع

- البت 4: الملف هو مجلد فرعي.
- البت 5: أرشيف.
- البت 6: جهاز.
- البت 7: غير مستخدم.

- البايت 12: غير مستخدم.
- البايت 13: وقت الانشاء بوحدة MS.
- البايتات 14–15: وقت الانشاء بالترتيب التالي:
  - البتات 0–4: الثواني (0–29).
  - البتات 5–10: الدقائق (0–59).
  - البتات 11–15: الساعات (0–23).
- البايتات 16–17: سنة الانشاء بالترتيب التالي:
  - البتات 0–4: السنة (0=1980; 127=2107).
  - البتات 5–8: الشهر (1=يناير; 12=ديسمبر).
  - البتات 9–15: الساعة (0–23).
- البايتات 18–19: تاريخ آخر استخدام (تتبع نفس الترتيب السابق).
- البايتات 20–21: EA index.
- البايتات 22–23: وقت آخر تعديل (تتبع نفس ترتيب البايتات 14–15).
- البايتات 24–25: تاريخ آخر تعديل (تتبع نفس ترتيب البايتات 16–17).
- البايتات 26–27: عنوان أول كلستر للملف.
- البايتات 28–29: حجم الملف.

ويجب ملاحظة أن حجم السجلات هو ثابت Fixed Lenght Record فمثلا اسم الملف يجب ان يكون بطول 8 بايت وفي حالة زاد على ذلك فان هذا سوف يحدث ضرراً على هذا الدليل ، أيضا في حالة كان الاسم بحجم أقل من المطلوب فانه يجب تكلمة العدد الناقص من الحروف بحرف المسافة Space.

٣.٥.٣. هيكلة القرص المرن

يتكون القرص المرن من قرص Platter (أو عدة أقراص) مقسمة الى مسارات (Tracks) وكل من هذه المسارات يتكون من العديد من القطاعات ويوجد عادة رأسين للقراءة والكتابة على كل قرص. وفي الاقراص المرنة ذات الحجم 1.44 MB يوجد 80 مساراً (من المسار 0 الى المسار 79) وكل مسار يتكون من 18 قطاع ، وبالتالي فان عدد القطاعات الكلية هي $2 * 18 * 80$ وتساوي 2880 قطاعاً.
ولتخزين بيانات على القرص فانه يجب تحديد العنوان الحقيقي والذي يتكون من عنوان القطاع والمسار والرأس ، وأول قطاع في القرص (قطاع الاقلاع) يأخذ العنوان: القطاع 1 المسار 0 الرأس 0 ، والقطاع الثاني يأخذ العنوان: القطاع 2 المسار 0 الرأس 0 ، وهكذا يستمر نظام التخزين في القرص المرن الى أن يصل الى العنوان 18 المسار 0 الرأس 0 وهو عنوان آخر قطاع على المسار الاول والرأس الاول ، وسيتم حفظ البيانات التالية في الرأس الثاني على العنوان: القطاع 1 المسار 0 الرأس 1 ويستمر الى أن يصل الى آخر قطاع في هذا المسار على الرأس الثاني، وبعدها سيتم حفظ البيانات التالية في الرأس الاول المسار الثاني ... ،وهكذا. والصورة التالية توضح شكل القرص المرن بعد عمل تهئية (Format) له.

٣.٥.٤. القراءة و الكتابة من نظام FAT12

حتى نتمكن من التعامل مع القرص المرن (قراءة وكتابة القطاعات) فانه يلزمنا برمجة درايفر لنظام FAT12 والذي سيعمل كوسيط بين المستخدم وبين القرص المرن، بمعنى أن أي طلب لقراءة ملف ما يجب أن تذهب أولا الى نظام FAT12 حيث سيقرر ما اذا كان الملف موجوداً أم لا (عن طريق البحث في دليل Root directory) وفي حالة كان موجوداً سيعود لنا بجميع خصائص الملف ورقم أول كلستر له لكي نتمكن من تحميل الملف كاملاً ، ونفس المبدأ في حالة طلب المستخدم كتابة ملف على القرص فان درايفر نظام FAT12 سيبحث في جدول FAT عن مساحة خالية مناسبة للملف وذلك باتباع أحد الخورازميات المعروفة وبعدها سيتم حفظ الملف وكتابة البيانات المتعلقة به في دليل Root directory .
وسنأخذ مثال على الموضوع وذلك ببرمجة المرحلة الثانية من محمل النظام Second Stage Bootloader وستقتصر وظيفته حالياً في طباعة رسالة ترحيبة دلالة على أنه تم تحميل وتنفيذ المرحلة الثانية بنجاح ، وفي الأقسام التالية سنبدأ في تطوير المرحلة الثانية وتجهيز مرحلة الانتقال الى بيئة 32 بت.
مهمة المرحلة الاولى ستتغير عن ما سبق ، حيث الان يجب على المرحلة الاولى أن تقوم بالبحث عن المرحلة الثانية من محمل النظام ونقل التنفيذ اليها ، ويتم هذا وفق الخطوات التالية:

١. تحميل جدول Root Directory من القرص الى الذاكرة ومن ثم البحث عن ملف المرحلة الثانية وأخذ رقم أول كلستر له.
٢. تحميل جدول FAT من القرص الى الذاكرة ومن ثم تحميل جميع الكلسترات للملف.
٣. نقل التنفيذ الى أول بايت في المرحلة الثانية من محمل النظام.

إنشاء المرحلة الثانية من محمل النظام

بداية سنقوم بإنشاء المرحلة الثانية من محمل النظام ونسخها الى القرص المرن ، ونظراً لان تطوير نظامنا الخاص يجب ان يتم تحت نظام آخر فان هذا النظام الآخر غالبا ما يحوي درايفر لنظام ملفات FAT12 حيث يتكفل بعملية كتابة البيانات الى جدول Root Directory بالاضافة الى البحث عن كلسترات خالية في جدول FAT دون أي تدخل من قبل مطور النظام الجديد، لذلك في هذه المرحلة من التطوير سنتجاهل جزئية الكتابة في نظام FAT12 ونترك المهمة لنظام التشغيل الذي نعتمد عليه في عملية تطوير النظام الجديد ، وبهذا سيكون الدرايفر الذي سننشئه في هذا الفصل ما هو الا جزء من الدرايفر الكامل الذي سيتم تكلمته في الفصل الخامس بمشيئة الله.والشفرة التالية توضح مثال للمرحلة الثانية من المحمل لعرض رسالة بسيطة.

Listing ٣٩.: Some Code

```
; Second Stage Bootloader.
; loaded by stage1.bin at address 0x050:0x0 (0x00500).


bits 16   ; 16—bit real mode.
org  0x0    ; offset to zero.

start:  jmp stage2


; data and variable
hello_msg db    "Welcome to eqraOS Stage2",0xa,0xd,0

; include files:
%include "stdio.inc"    ; standard i/o routines.




; ************************************
; entry point of stage2 bootloader.
; ************************************

stage2:

    push cs
    pop ds    ; ds = cs.
```

```
mov si,hello_msg
call puts16

cli     ; clear interrupt.
hlt     ; halt the system.
```

وسيتم تسمية الملف بالاسم stage2.asm أما الملف الناتج من عملية التجميع سيكون بالاسم stage2.sys ويمكن تسميته بأي اسم اخر بشرط أن لا يزيد الاسم عن 8 حروف والامتداد عن 3 حروف ، وفي حالة كان طول الاسم أقل فان درايفر FAT12 سيقوم باضافة مسافات Spaces حتى لا يتضرر جدول Root Directory. ويمكننا أن نفرق بين اسماء الملفات الداخلية (وهي التي يتم اضافة مسافات عليها ويستخدمها نظام FAT12) والأسماء الخارجية (وهي التي ينشئها المستخدم).

تحميل ال Root Directory الى الذاكرة

جدول Root Directory يحوي أسماء كل الملفات و أماكن تواجدها على القرص لذا يجب تحميله أولا والبحث عن ملف المرحلة الثانية (ذو الاسم الخارجي stage2.sys) وعند البحث يجب البحث بالاسم الداخلي الذي يستخدمه نظام الملفات لذلك يجب أن نبحث عن الملف "stage2 sys" ، ونأتي برقم الكلستر الأول للملف.
وقبل تحميل هذا الجدول فانه يجب علينا أولاً معرفة عنوان أول قطاع فيه وحساب عدد القطاعات التي يشغلها هذا الجدول ، كذلك يجب تحديد المساحة الخالية (Buffer) لكي يتم نقل هذا الجدول اليها. والشفرة التالية توضح كيفية عمل ذلك.

Listing ٣١٠.: Some Code

```
;-------------------------------
; Compute Root Directory Size
;-------------------------------

  xor cx,cx
  mov ax,32           ; every root entry size are 32 byte.
  mul word[root_directory]  ; dx:ax = 32*224 bytes
  div word[bytes_per_sector]
  xchg ax,cx          ; cx = number of sectors to load.

;-----------------------------------------
; Get start sector of root directory
;-----------------------------------------

  mov al,byte[total_fats]     ; there are 2 fats.
```

```
  mul word[sectors_per_fat]   ; 9*2 sectors
  add ax,word[reserved_sectors] ; ax = start sector of
     root directory.

  mov word[data_region],ax
  add word[data_region],cx  ; data_region = start sector
     of data.


;-------------------------------------------------------
; Load Root Dir at 0x07c0:0x0200 above bootloader.
;-------------------------------------------------------

  mov bx,0x0200     ; es:bs = 0x07c0:0x0200.
  call read_sectors
```

بعد تحميل هذا الجدول يجب البحث فيه عن اسم ملف المرحلة الثانية من محمل النظام ومن ثم حفظ رقم أول كلستر له في حالة كان الملف موجوداً ، أما اذا كان الملف غير موجود فنصدر رسالة خطأ ونوقف النظام عن العمل. والشفرة التالية توضح ذلك.

Listing ٣١١.: Some Code

```
;--------------------
; Find stage2.sys
;--------------------

  mov di,0x0200       ; di point to first entry in root
     dir.
  mov cx,word[root_directory] ; loop 224 time.

find_stage2:

  mov si,kernel_loader_name
  push cx
  push di
  mov cx,11         ; file name are 11 char long.

  rep cmpsb
  pop di
  je find_successfully

  mov di,32       ; point to next entry.
```

```
    pop cx

    loop find_stage2

    ; no found ?
    jmp find_fail

find_successfully:
;-------------------------
; Get first Cluster.
;-------------------------

    mov ax,word[di+26]     ; 27 byte in the di entry are
        cluster number.
    mov word[cluster_number],ax
```

تحميل جدول FAT الى الذاكرة

جدول FAT يوضح حالة كل الكلسترات الموجودة على القرص سواءا كانت خالية أم معطوبة أم انها مستخدمة ، ويجب تحميل هذا الجدول الى الذاكرة لكي نستطيع عن طريق رقم الكلستر الذي تحصلنا عليه من جدول Root Directory أن نحمل جميع كلسترات الملف. وبنفس الطريقة التي قمنا بها لتحميل جدول Root Directory سيتم بها تحميل جدول FAT حيث يجب تحدد عنوان أول قطاع للجدول و عدد القطاعات التي يشغلها الجدول ، وكذلك المساحة الخالية في الذاكرة لكي يتم حفظ الحدول بها . والشفرة التالية توضح ذلك.

Listing ٣١٢.: Some Code

```
;-------------------------
; Compute FAT size
;-------------------------

    xor cx,cx
    xor ax,ax
    xor dx,dx

    mov al,byte[total_fats]      ; there are 2 fats.
    mul word[sectors_per_fat]    ; 9*2 sectors
    xchg ax,cx
```

```
;-----------------------------
; Get start sector of FAT
;-----------------------------

  add ax,word[reserved_sectors]

;-------------------------------------------------------------
; Load FAT at 0x07c0:0x0200
; Overwrite Root dir with FAT, no need to Root Dir now.
;-------------------------------------------------------------

  mov bx,0x0200
  call read_sectors
```

تحميل كلسترات الملف

وحدة القراءة والكتابة للقرص المرن هي بالقطاع Sector لكن نظام الملفات FAT12 يتعامل مع مجموعة من القطاعات ككتلة واحدة Cluster، وكلما كبر حجم الكلستر زادت المساحات الخالية بداخله Internel Fragmentation لذلك يجب اختيار حجم ملائم ، وفي تنفيذ نظام FAT12 على قرص مرن أخترنا أن كل كلستر يقابل قطاع واحد فقط من القرص المرن. المشكلة التي ستواجهنا هي كيفية قراءة كلستر من القرص ، فالقرص المرن لا يقرأ اي قطاع الا بتحديد العنوان المطلق له Absolute Address ولذلك يجب تحويل رقم الكلستر الى عنوان مطلق وتحويل عنوان LBA أيضا الى عنوان مطلق.
التحويل من رقم Cluster الى عنوان LBA يتم كالاتي:

Listing ٣١٣.: Some Code

```
; ************************************************
; cluster_to_lba: convert cluster number to LBA
;   input:
;       ax: Cluster number.
; output:
;       ax: lba number.
; ************************************************
cluster_to_lba:

    ; lba = (cluster - 2)* sectors_per_cluster
    ; the first cluster is always 2.

    sub ax,2
```

```
    xor cx,cx
    mov cl, byte[sectors_per_cluster]
    mul cx

    add ax,word[data_region]     ; cluster start from data
       area.
    ret
```

حيث يتم طرح العدد 2 من رقم الكلستر وهذا بسبب أن أول رقم كلستر في نظام FAT12 هو 2 – كما سنرى ذلك لاحقا–.
وللتحويل من عنوان LBA الى عنوان Absolute Address :

Listing ٣١٤.: Some Code

```
; ******************************************
; lba_to_chs: Convert LBA to CHS.
;   input:
;        ax: LBA.
; output:
;       absolute_sector
;       absolute_track
;       absolute_head
; ******************************************
lba_to_chs:

    ; absolute_sector =  (lba % sectors_per_track) + 1
    ; absolute_track  =  (lba / sectors_per_track) /
       number_of_heads
    ; absolute_head   =  (lba / sectors_per_track) %
       number_of_heads

    xor dx,dx
    div word[sectors_per_track]
    inc dl
    mov byte[absolute_sector],dl

    xor dx,dx
    div word[number_of_heads]
    mov byte[absolute_track],al
    mov byte[absolute_head],dl
```

```
    ret
```

ولتحميل كلستر من القرص يجب أولا الحصول على رقمه من جدول Root Directory وبعد ذلك نقوم بتحويل هذا الرقم الى عنوان LBA وبعدها نقوم بتحويل عنوان LBA الى عنوان مطلق Abolsute Address ومن ثم استخدام مقاطعة البايوس int 0x13 لقراءة القطاعات من القرص، والشفرة التالية توضح ذلك.

Listing ٣١٥.: Some Code

```
;————————————————————————
; Load all clusters(stage2.sys)
; At address 0x050:0x0
;————————————————————————

  xor bx,bx
  mov ax,0x0050
  mov es,ax

load_cluster:

  mov ax,word[cluster_number]    ; ax = cluster number
  call cluster_to_lba        ; convert cluster number to
     LBA addressing.

  xor cx,cx
  mov cl,byte[sectors_per_cluster]  ; cx = 1 sector

  call read_sectors_bios         ; load cluster.
```

ودالة قراءة القطاعات من القرص تستخدم مقاطعة البايوس int 0x13 وهي تعمل فقط في النمط الحقيقي ويجب استبدالها لاحقا عند التحويل الى النمط المحمي بدالة اخرى 32-bit.

Listing ٣١٦.: Some Code

```
; ******************************************
; read_sectors_bios: load sector from floppy disk
;   input:
;       es:bx : Buffer to load sector.
;       ax:   first sector number ,LBA.
;       cx:   number of sectors.
; ******************************************
read_sectors_bios:
```

```
begin:
  mov di,5    ; try 5 times to load any sector.

load_sector:

  push ax
  push bx
  push cx

  call lba_to_chs

  mov ah,0x2            ; load sector routine number.
  mov al,0x1            ; 1 sector to read.
  mov ch,byte[absolute_track]   ; absolute track number.
  mov cl,byte[absolute_sector]  ; absolute sector number.
  mov dh,byte[absolute_head]    ; absolute head number.
  mov dl,byte[drive_number]   ; floppy drive number.

  int 0x13            ; call BIOS.

  jnc continue  ; if no error jmp.

  ; reset the floppy and try read again.

  mov ah,0x0          ; reset routine number.
  mov dl,0x0          ; floppy drive number.
  int 0x13          ; call BIOS.

  pop cx
  pop bx
  pop ax

  dec di
  jne load_sector

  ; error.
  int 0x18

continue:

  mov si,progress_msg
  call puts16
```

```
pop cx
pop bx
pop ax

add ax,1              ; next sector
add bx,word[bytes_per_sector] ; point to next empty
   block in buffer.


loop begin     ; cx time

ret
```

ولتحميل بقية كلسترات الملف يجب أخذ رقم أول كلستر للملف والذهاب به الى جدول FAT وقراءة القيمة المقابلة له والتي ستدل على ما اذا كان هذا آخر كلستر أم أن هنالك كلسترات اخرى يجب تحميلها.ويلزم الأخذ بالاعتبار بنية جدول FAT وانه يتكون من سجلات بطول 12 بت وتعادل بايت ونصف ، أي أنه اذا كان رقم الكلستر هو 0 فاننا يجب أن نقرأ السجل الاول من جدول FAT وبسبب انه لا يمكن قراءة 12 بت فسوف تتم قراءة 16 بت (السجل الاول بالاضافة الى نصف السجل الثاني) وعمل mask لاخر 4 بت (لازالة ما تم قرائته من السجل الثاني). وفي حالة كان رقم الكلستر هو 1 فيجب قراءة السجل الثاني من جدول FAT والذي يبدأ من البت 12-23 وبسبب أنه لا يمكن قراءة 12 بت سنقوم بقراءة 16 بت أي من البت 8-23 وازالة أول 4 بت.

وباختصار، لقراءة القيمة المقابلة لرقم كلستر ما فيجب أولا تطبيق القانون :
$$cluster = cluster + (cluster/2)$$

وقراءة 16 بت ، وفي حالة ما اذا كان رقم الكلستر هو رقم زوجي فيجب عمل Mask لاخر 4 بت ، أما اذا كان رقم الكلستر فردي فيجب ازالة أول 4 بت . والشفرة التالية توضح كيفية تحميل جميع كلسترات المرحلة الثانية من محمل النظام الى الذاكرة ونقل التنفيذ اليها .

Listing ٣١٧.: Some Code

```
read_cluster_fat_entry:

  mov ax,word[cluster_number]

  ; Every FAT entry are 12-bit long( byte and half one).
  ; so we must map the cluster number to this entry.
  ; to read cluster 0 we need to read fat[0].
  ; cluster 1 -> fat[1].
```

```
    ; cluster 2 -> fat[3],...etc.

    mov cx,ax    ; cx = cluster number.
    shr cx,1     ; divide cx by 2.
    add cx,ax    ; cx = ax + (ax/2).
    mov di,cx
    add di,0x0200
    mov dx,word[di] ; read 16-bit form FAT.


    ; Now, because FAT entry are 12-bit long, we should
       remove 4 bits.
    ; if the cluster number are even, we must mask the last
       four bits.
    ; if it odd, we must do four right shift.

    test ax,1
    jne odd_cluster

  even_cluster:

    and dx,0x0fff
    jmp next_cluster

  odd_cluster:

    shr dx,4


  next_cluster:
    mov word[cluster_number],dx   ; next cluster to load.

    cmp dx,0x0ff0          ; check end of file, last cluster?
    jb load_cluster          ; no, load the next cluster.


    ; yes jmp to end
    jmp end_of_first_stage

  find_fail:

    mov si,fail_msg
```

```
  call puts16

  mov ah,0x0
  int 0x16    ; wait keypress.
  int 0x19    ; warm boot.


end_of_first_stage:

  ; jump to stage2 and begin execute.
  push 0x050      ; segment number.
  push 0x0      ; offset number.

  retf      ; cs:ip = 0x050:0x0

  times 510-($-$$)  db    0   ; append zeros.

  ; finally the boot signature 0xaa55
  db    0x55
  db    0xaa
```

٤. برمجة محمل النظام – المرحلة الثانية

بسبب القيود على حجم محمل النظام فان هذا قد أدى الى تقسيم المهمة الى مرحلتين حيث اقتصرت مهمة المرحلة الاولى على تحميل المرحلة الثانية من المحمل ، أما المرحلة الثانية stage 2 فلا قيود عليها وغالبا ما يتم تنفيذ المهمات التالية في هذه المرحلة:

- الانتقال الى النمط المحميPMode.
- تفعيل البوابة A20 لدعم ذاكرة حتى 4 جيجا بايت.
- توفير دوال للتعامل مع المقاطعات Interrupt Handler.
- تحميل النواة ونقل التنفيذ والتحكم اليها.
- توفير خصائص أثناء الإقلاع مثل Safe Mode.
- دعم الإقلاع المتعدد Multi Boot وذلك عبر ملفات التهيئة.

١.٤. الانتقال الى النمط المحمي

المشكلة الرئيسية في النمط الحقيقي Real Mode هي عدم توفر حماية للذاكرة حيث يمكن لأي برنامج يعمل أن يصل لأي جزء من الذاكرة ، كذلك أقصى حجم يمكن الوصول له هو 1 ميجا من الذاكرة ، ولا يوجد دعم لتقنية Paging ولا للذاكرة الظاهرية Virtual Memory حتى تعدد البرامج لا يوجد دعم له.
كل هذه المشاكل تم حلها باضافة النمط المحمي الى المعالج ويمكن الانتقال بسهولة الى هذا النمط عن طريق تفعيل البت الاول في المسجل cr0 ، ولكن بسبب أن المعالج في هذا النمط يستخدم طريقة عنونة للذاكرة تختلف عن الطريقة المستخدمة في النمط الحقيقي فانه يجب تجهيز بعض الجداول تسمى جداول الواصفات Descriptor Table وبدون تجهيز هذه الجداول فان المعالج سيصدر استثناء General Protection Fault واختصاراً GPF والذي بدوره يؤدي الى حدوث triple fault وتوقف النظام عن العمل.
أحد هذه الجداول ويسمى جدول الواصفات العام (Global Descriptor Table) واختصاراً GDT وظيفته الاساسية هي تعريف كيفية استخدام الذاكرة ، حيث يحدد ما هو القسم الذي سينفذ كشفرة ؟ وما هو القسم الذي يجب أن يحوي بيانات ؟ ويحدد أيضا بداية ونهاية كل قسم بالاضافة الى صلاحية الوصول الى ذلك القسم.

١.١.٤. جدول الواصفات العام Global Descriptor Table

عند الانتقال الى النمط المحمي PMode فان أي عملية وصول الى الذاكرة تتم عن طريق هذا الجدول GDT ، هذا الجدول يعمل على حماية الذاكرة وذلك بفحص العنوان المراد الوصول اليه والتأكد من عدم مخالفته لبيانات هذا الجدول.هذه البيانات تحدد القسم الذي يمكن أن ينفذ كشفرة (Code) والقسم الذي لا ينفذ (Data) كذلك تحدد هذه البيانات العديد من الخصائص كما سنراها الان.
وعادة يتكون جدول GDT من ثلاث واصفات Descriptors (حجم كلٌ منها هو 64 بت) وهم:

- Null Descriptor: تكون فارغة في العادة.
- Code Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي ينفذ كشفرة Code.
- Data Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي لا ينفذ ويحوي بيانات Data.

بيانات أي واصفة Descriptor تأخذ الجدول التالي:

- البتات 0–15: تحوي أول بايتين (من بت 0 –15) من حجم المقطع.
- البتات 16–39: تحوي أول ثلاث بايتات من عنوان بداية المقطع Base Address.
- البت 40: بت الوصول Access Bit (يستخدم مع الذاكرة الظاهرية Virtual Memory.
- البتات 43–41: نوع الواصفة Descriptor Type:
  - البت 41: القراءة والكتابة:
    - Data Descriptor: القيمة 0 للقراءة فقط والقيمة 1 للقراءة والكتابة.
    - Code Descriptor: القيمة 0 للتنفيذ فقط execute والقيمة 1 للقراءة والتنفيذ.
  - البت 42: Expansion direction (Data segments), conforming (Code Segments).
  - البت 43: قابلية التنفيذ:
    - 0: اذا كان المقطع عبارة عن بيانات.
    - 1: اذا كان المقطع عبارة عن شفرة.
- البت 44: Descriptor Bit:
  - 0: System descriptor.
  - 1: Code or Data Descriptor.
- البتات 45–46: مستوى الحماية Privilege Level
  - 0: Highest (Ring 0).

- 3: Lowest (Ring 3).
- البت 47: Segment is in memory (Used with Virtual Memory).
- البتات 48-51: تحوي البت 16 -19 من حجم المقطع.
- البت 52: محجوزة.
- البت 53: محجوزة.
- البت 54: نوع المقطع Segment type:
  - 0: اذا كان المقطع 16 بت.
  - 1: اذا كان المقطع 32 بت.
- البت 55: Granularity:
  - 0: None.
  - 1: Limit gets multiplied by 4K.
- البتات 56-63: تحوي البت 23 -32 من عنوان بداية المقطع Base Address.

وفي هذه المرحلة سنقوم ببناء هذا الجدول ويتكون من واصفة للكود وللبيانات Code and Data Descriptor، بحيث يمكن القراءة و الكتابة من أول بايت في الذاكرة الى آخر الذاكرة 0xffffffff.

Listing ٤١.: Some Code

```
;**************************
; Global Descriptor Table
;**************************

begin_of_gdt:

; Null Descriptor: start at 0x0.

  dd  0x0     ; fill 8 byte with zero.
  dd  0x0

; Code Descriptor: start at 0x8.

  dw  0xffff    ; limit low.
  dw  0x0     ; base low.
  db  0x0     ; base middle.
  db  10011010b ; access byte.
```

```
    db  11001111b ; granularity byte.
    db  0x0     ; base high.

  ; Data Descriptor: start at 0x10.

    dw  0xffff    ; limit low.
    dw  0x0     ; base low.
    db  0x0     ; base middle.
    db  10010010b ; access byte.
    db  11001111b ; granularity byte.
    db  0x0     ; base high.

  end_of_gdt:
```

هذا الجدول يبدأ بالواصفة الخالية Null Descriptor وحجمها 8 بايت ومتحوياتها تكون صفراً في العادة ، أما الواصفة التالية لها فهي واصفة مقطع الشفرة Code Descriptor وتوضح المقطع من الذاكرة الذي سيتسخدم كشفرة وما هي بدايته وحجمه وصلاحيات استخدامه حيث يمكن أن نسمح فقط للبرامج التي تعمل على مستوى النواة Kernel Mode بالدخول الى هذا المقطع.وفيما يلي شرح لمحتويات هذه الواصفة ويمكنك المطابقة مع الجدول الذي يوضح الشكل العام لكل واصفة.
تبدأ واصفة الكود Code Descriptor من العنوان 0x8 وهذا العنوان مهم جدا حيث سيكون هذا العنوان هو قيمة المسجل CS ، والبتات من 0-15 تحدد حجم المقطع Segment Limit والقيمة هي 0xffff تدل على أن أكبر حجم يمكن التعامل معه هو 0xffff.
البتات من 16-39 تمثل البتات 0-23 من عنوان بداية المقطع Base Address والقيمة التي تم اختيارها هي 0x0 وبالتالي نعرف أن عنوان بداية مقطع الكود هو 0x0 وعنوان النهاية 0xffff .
البايت رقم 6 ويسمى Access Byte يحدد العديد من الخصائص وفيما يلي توضيح لمعنى كل بت موجودة فيه:

- البت 0: Access Bit ويستخدم مع الذاكرة الظاهرية لذلك اخترنا القيمة 0.
- البت 1: بت القراءة والكتابة ، وتم اختيار القيمة 1 لذا يمكن قراءة وتنفيذ أي بايت موجودة في مقطع الكود من 0x0-0xffff.
- البت 2: expansion direction لا يهم حاليا لذا القيمة هي 0.
- البت 3: تم اختيار القيمة 1 دلالة على أن هذا مقطع شفرة Code Segment.
- البت 4: تم اختيار القيمة 1 دلالة على أن هذا مقطع للشفرة او للبيانات وليس للنظام.
- البتات 5-6: مستوى الحماية وتم اختيار القيمة 0 دلالة على أن هذا المقطع يستخدم فقط في الحلقة صفر Ring0 أو ما يسمى Kernel Mode.

- البت 7: تستخدم مع الذاكرة الظاهرية لذا تم اهمالها.

البايت رقم 7 ويسمى granularity يحدد أيضا بعض الخصائص، وفيما يلي توضيح لمعنى كل بت موجودة فيه:

- البتات 0-3: تمثل البتات من 16-19 من نهاية حجم المقطع Segment Limit والقيمة هي 0xf ، وبهذا يكون أقصى عنوان للمقطع هو 0xfffff أي 1 ميجا من الذاكرة ، ولاحقاً عندما يتم تفعيل بوابة A20 سنتمكن من الوصول حتى 4 جيجا من الذاكرة.
- البتات 4-5: محجوزة للنظام لذا تم اهمالها.
- البت 6: تم اختيار القيمة 1 دلالة على هذا المقطع هو 32 بت.
- البت 7: باختيار القيمة 1 سيتم إحاطة المقطع ب 4 KB.

البايت الاخير في واصفة مقطع الكود (البايت رقم 8) يمثل البتات من 24-32 من عنوان بداية مقطع الكود والقيمة هي 0x0 وبالتالي عنوان بداية مقطع الكود الكلي هو 0x0 أي من أول بايت في الذاكرة.
إذاً واصفة مقطع الكود Code Descriptor حددت عنوان بداية مقطع الكود ونهايته وكذلك صلاحية التنفيذ وحددت بأن المقطع هو مقطع كود Code Segment.
الواصفة التالية هي واصفة مقطع البيانات Data Descriptor وتبدأ من العنوان رقم 0x10 وهي مشابهة تماما لواصفة الكود باستثناء البت رقم 43 حيث يحدد ما اذا كان المقطع كود أم بيانات.
وبعد إنشاء هذا الجدول (GDT) في الذاكرة ، يجب أن يَحمل المسجل gdtr على حجم هذا الجدول ناقصا واحد وعلى عنوان بداية الجدول، ويتم ذلك عن طرَيق إنشاء مؤشرا الى جدول GDT ومن ثم استخدام الامر lgdt (وهو أمر يعمل فقط في الحلقة صفر Ring0) ، والشفرة التالية توضح ذلك.

Listing ٤٢.: Some Code

```
bits  16    ; real mode.

;*****************************
; load_gdt: Load GDT into GDTR.
;*****************************

load_gdt:

    cli          ; clear interrupt.
    pusha        ; save registers
    lgdt [gdt_ptr]    ; load gdt into gdtr
    sti          ; enable interrupt
```

```
    popa        ; restore registers.

    ret


;*************************************
; gdt_ptr: data structure used by gdtr
;*************************************

gdt_ptr:

    dw end_of_gdt – begin_of_gdt – 1    ; size –1
    dd begin_of_gdt             ; base of gdt
```

٤.١.٢. العنونة في النمط المحمي PMode Memory Addressing

في النمط الحقيقي يستخدم المعالج عنونة Segment:Offset وذلك بأن تكون أي من مسجلات المقاطع (Segments Registers) تحوي عنوان بداية المقطع ، ومسجلات العناوين تحوي العنوان داخل مقطع ما ، ويتم ضرب عنوان المقطع بالعدد 0x10 وجمع ال offset اليه للحصول على العنوان النهائي والذي سيمر بداخل مسار العنوان Address Bus.
أما النمط المحمي PMode فانه يستخدم عنونة Descriptor:Offset وذلك بأن تكون مسجلات المقاطع تحوي عنوان أحد الواصفات التي قمنا ببنائها (مثلا مسجل CS يحوي العنوان 0x8 ومسجل البيانات DS يحوي العنوان 0x10) ، وال offset سيتم جمعها الى عنوان بداية المقطع Base Address والذي قمنا بتحديده في جدول الواصفات كذلك سيتم التأكد من أن هذا العنوان لا يتجاوز حجم المقطع Segment Limit أيضا سيتم التأكد من مستوى الصلاحية وأنه يمكن الوصول للعنوان المطلوب. ونظراً لان في النمط المحمي يمكن استخدام مسجلات 32-bit فانه يمكن عنونة 4 جيجا من الذاكرة[١].

٤.١.٣. الانتقال الى النمط المحمي

بعد إنشاء جدول GDT وتحميل مسجل GDTR يمكن الانتقال الى النمط المحمي عن طريق تفعيل البت الاول في مسجل التحكم cr0، وكما هو معروف أن هذا النمط لا يستخدم مقاطعات البايوس لذا يجب تعطيل عمل المقاطعات قبل الانتقال حتى لا تحدث أي مشاكل.

[١] بفرض أن بوابة A20 تم تفعيلها.

وبعد الانتقال الى النمط المحمي فان يجب تعيين الواصفة التي يجب استخدامها لمسجلات المقاطع ، وبالنسبة لمسجل CS فانه يمكن تعديل قيمته وذلك عن طريق تنفيذ far jump ،والكود التالي يوضح طريقة الانتقال الى النمط المحمي وتعديل قيم مسجلات المقاطع.

Listing ٤٣.: Some Code

```
    ;————————————————————
    ; Load gdt into gdtr.
    ;————————————————————

      call load_gdt

    ;————————————————————
    ; Go to PMode.
    ;————————————————————
      ; just set bit 0 from cr0 (Control Register 0).

      cli         ; important.
      mov eax,cr0
      or eax,0x1
      mov cr0,eax     ; entering pmode.


    ;————————————————————
    ; Fix CS value
    ;————————————————————
      ; select the code descriptor
      jmp 0x8:stage3


  ;**************************
  ; entry point of stage3
  ;**************************

  bits 32     ; code now 32—bit

  stage3:

    ;————————————————————;
    ; Set Registers.
    ;————————————————————;
```

```
  mov ax,0x10     ; address of data descriptor.
  mov ds,ax
  mov ss,ax
  mov es,ax
  mov esp,0x90000   ; stack begin from 0x90000.


;—————————————;
; Hlat the system.
;—————————————;
  cli     ; clear interrupt.
  hlt     ; halt the system.
```

٢.٤. تفعيل البوابة A20

بوابة A20 Gate هي عبارة عن OR Gate موجودة على ناقل النظام System Bus [٢] والهدف منها هو التحكم في عدد خطوط العناوين Address Line، حيث كانت الاجهزة قديما (ذات المعالجات التي تسبق معالج 80286) تحوي على 20 بت (خط) للعناوين (20 address line) ، وعندما صدرت اجهزة IBM PC والتي احتوت على معالج 80286 تم زيادة خط العناوين الى 32 خط وهكذا أصبح من الممكن عنونة 4 جيجا من الذاكرة ، وحتى يتم الحفاظ على التوافقية مع الاجهزة السابقة فانه يمكن التحكم في بوابة A20 من فتح الخطوط A20–A31 واغلاقها.
هذه البوابة مرتبطة مع متحكم 8042 وهو متحكم لوحة المفاتيح (Keyboard Controller) ، وعند تفعيل البت رقم 1 في منفذ خروج البيانات (output data port) التابع لمتحكم لوحة المفاتيح فان هذا يفتح بوابة A20 وبهذا نستطيع الوصول الى 4 جيجا من الذاكرة ، ابتداءاً من العنوان 0x0–0xffffffff
وعند اقلاع الحاسب فان البايوس يقوم بتفعيل هذه البوابة لأغراض حساب حجم الذاكرة واختبارها ومن ثم يقوم بغلقها مجدداً للحفاظ على التوافقية مع الاجهزة القديمة.
وتوجد العديد من الطرق لتفعيل هذه البوابة ، العديد منها يعمل على أجهزة معينة لذلك سيتم ذكر العديد من الطرق واستخدام أكثر الطرق محمولية على صعيد الاجهزة المختلفة.

١.٢.٤. متحكم لوحة المفاتيح 8042 والبوابة A20

عند الانتقال الى النمط المحمي (PMode) فانه لن يمكن استخدام مقاطعات البايوس ويجب التعامل المباشر مع متحكم أي عتاد والقراءة والكتابة من مسجلات المتحكم الداخلية . وبسبب ارتباط بوابة A20 مع

[٢] توجد البوابة تحديداً على خط العناوين رقم 20

متحكم لوحة المفاتيح فانه لا بد من التعامل مع هذا المتحكم لتفعيل البوابة ، وهذا يتم عن طريق استخدام أوامر المعالج in والامر out.
وبخصوص متحكم لوحة المفاتيح (متحكم 8042) فغالبا ما تأتي على شكل شريحة Integrated Circuit أو تكون مضمنة داخل اللوحة الأم (Motherboard) وتكون في ال South Bridge.ويرتبط هذا المتحكم مع متحكم آخر بداخل لوحة المفاتيح ، وعند الضغط على زر ما فانه يتم توليد Make Code ويُرسل الى المتحكم الموجود بداخل لوحة المفاتيح والذي بدروه يقوم بارساله الى متحكم 8042 عن طريق منفذ الحاسب (Hardware Port) .وهنا يأتي دور متحكم 8042 حيث يقوم بتحويل Make code الى Scan Code ويحفظها في مسجلاته الداخلية Buffer هذا المسجل يحمل الرقم 0x60 في أجهزة IBM and Compatible PC، وهذا يعني أنه في حالة قراءة هذا المسجل (عن طريق الأمر in) فانه يمكن قراءة القيمة المدخلة.
وفي الفصل السادس سيتم مناقشة متحكم لوحة المفاتيح بالتفصيل ، وسنكتفي هنا فقط بتوضيح الأجزاء المتعلقة بتفعيل بوابة A20.

٢.٢.٤. طرق تفعيل البوابة A20

بواسطة System Control Port 0x92

في بعض الاجهزة يمكن استخدام أحد منافذ الادخال والاخراج وهو I/O part 0x92 لتفعيل بوابة A20 ، وعلى الرغم من سهولة هذه الطريقة الا أنها تعتبر أقل محمولية وبعض الاجهزة لا تدعمها ، وفيما يلي توضيح للبتات على هذا المنفذ:

- البت 0: تفعيل هذا البت يؤدي الى عمل reset للنظام والعودة الى النمط الحقيقي.
- البت 1: القيمة 0 لتعطيل بوابة A20 ، والقيمة 1 لتفعيلها.
- البت 2: لا تستخدم.
- البت 3: power on password bytes
- البتات 4-5: لا تستخدم.
- البتات 6-7: HDD activity LED : القيمة 0: off، القيمة 1: on.

والمثال التالي يوضح طريقة تفعيل البوابة .

Listing ٤٤.: Some Code

```
;***********************************************
; enable_a20_port_0x92:
;   Enable A20 with System Control port 0x92
```

```
;************************************************

enable_a20_port_0x92:

    push ax      ; save register.

    mov al,2     ; set bit 2 to enable A20
    out 0x92,al

    pop ax       ; restore register.
    ret
```

ويجب ملاحظة أن هذه الطريقة لا تعمل في كل الاجهزة وربما يكون هناك ارقام مختلفة للمنافذ ، ويعتمد في الآخر على مصنعي اللوحات الام ويجب قراءة كتيباتها لمعرفة العناوين.

بواسطة البايوس

يمكن استخدام مقاطعة البايوس int 0x15 الدالة 0x2401 لتفعيل بوابة A20 ، والدالة 0x2400 لتعطيلها.مع التذكير بأن يجب أن يكون المعالج في النمط الحقيقي حتى نتمكن من استدعاء هذه المقاطعة، والكود التالي يوضح طريقة التفعيل باستخدام البايوس.

Listing ٤٥.: Some Code

```
;*************************************************
; enable_a20_bios:
;   Enable A20 with BIOS int 0x15 routine 0x2401
;*************************************************

enable_a20_bios:

    pusha      ; save all registers

    mov ax,0x2401 ; Enable A20 routine.
    int 0x15

    popa       ; restore registers
    ret
```

بواسطة متحكم لوحة المفاتيح

يوجد منفذين لمتحكم لوحة المفاتيح: المنفذ 0x60 وهو يمثل ال buffer (في حالة القراءة منه يسمى Output Buffer وفي حالة الكتابة يسمى Input Buffer، والمنفذ 0x64 وهو لإرسال الاوامر الى المتحكم ولقراءة حالة المتحكم (Status). حيث يتم ارسال الأوامر الى المتحكم عن طريق المنفذ 0x64 واذا كان هناك وسائط لهذا الأمر فترسل الى ال buffer (المنفذ 0x60) وكذلك تقرأ النتائج من المنفذ 0x60.
وحيث ان تنفيذ أوامر البرنامج (عن طريق المعالج) أسرع بكثير من تنفيذ الأوامر المرسلة الى متحكم لوحة المفاتيح (وبشكل عام الى أي متحكم لعتاد ما) فانه يجب ان نوفر طرقاً لانتظار المتحكم قبل العودة الى البرنامج لاستكمال التنفيذ .
ويمكن عن طريق قراءة حالة المتحكم (عن طريق قراءة المنفذ 0x64) أن نعرف ما اذا تم تنفيذ الاوامر المرسلة ام لا ، وكذلك هل هناك نتيجة لكي يتم قرائتها في البرنامج ام لا.
وما يهمنا من البتات عند قراءة حالة المتحكم حاليا هو أول بتين فقط ، ووظيفتهما هي:

- البت 0: حالة ال Output Buffer:
  - القيمة 0: ال Output Buffer خالي (لا توجد نتيجة ، لا تقرأ الان).
  - القيمة 1: ال Output Buffer ممتلئ (توجد نتيجة ، قم بالقراءة الان).
- البت 1: حالة ال Input Buffer:
  - القيمة 0: ال Input Buffer خالي (لا توجد أوامر غير منفذة ، يمكن الكتابة الان).
  - القيمة 1: ال Input Buffer ممتلئ (توجد أوامر غير منفذة ، لا تكتب الان).

والشفرة التالية توضح كيفية انتظار المتحكم حتى ينفذ الاوامر المرسله اليه (wait input) وكيفية انتظار المتحكم الى ان يأتي بنتيجة ما (wait output).

Listing ٤٦.: Some Code

```
;*****************************************
; wait_output: wait output buffer to be full.
;*****************************************

wait_output:

    in al,0x64     ; read status
    test al,0x1    ; is output buffer is empty?
    je wait_output  ; yes, hang.

    ret        ; no,there is a result.
```

```
;****************************************
; wait_input: wait input buffer to be empty.
        command executed already.
;****************************************

wait_input:

    in al,0x64     ; read status
    test al,0x2    ; is input buffer is full?
    jne wait_input  ; yes, hang.

    ret        ; no,command executed.
```

ولإرسال اوامر الي المتحكم فان يجب استخدام المنفذ 0x64 وتوجد الكثير من الأوامر ، ونظرا لان هذا الجزء غير مخصص لبرمجة متحكم لوحة المفاتيح فاننا سنناقش فقط الاوامر التي تهمنا حاليا ، وفي الفصل السادس سنعود الى الموضوع بالتفصيل ان شاء الله.
وقائمة الاوامر حاليا:

- الأمر 0xad: تعطيل لوحة المفاتيح.
- الأمر 0xae: تفعيل لوحة المفاتيح.
- الأمر 0xd0: القراءة من Output Port.
- الأمر 0xd1: الكتابة الى Output Port.
- الأمر 0xdd: تفعيل بوابة A20.
- الأمر 0xdf: تعطيل بوابة A20.

وعن طريق الأمر 0xdd فانه يمكن تفعيل البوابة A20 بسهولة كما في الشفرة التالية ، لكن أيضا هذه الطريقة لا تعمل على كل الاجهزة حيث هناك بعض المتحكمات لا تدعم هذا الأمر.

Listing ٤٧.: Some Code

```
;*************************************************
; enable_a20_keyboard_controller:
;   Enable A20 with command 0xdd
;*************************************************
```

```
enable_a20_keyboard_controller:

    ;cli
    push ax      ; save register.

    mov al,0xdd   ; Enable A20 Keyboard Controller Command.
    out 0x64,al

    pop ax       ; restore register.
    ret
```

وتوجد طريقة أخرى أكثر محمولية وهي عن طريق منفذ الخروج Output Port في متحكم لوحة المفاتيح ويمكن قراءة هذا المنفذ والكتابة اليه عن طريق ارسال الاوامر 0xd0 و 0xd1 على التوالي.
وعند قراءة هذا المنفذ (بارسال الامر d0 الى متحكم لوحة المفاتيح) فان القيم تعني:

- البت 0: System Reset:
  - القيمة 0: Reset Computer.
  - القيمة 1: Normal Operation.
- البت 1: بوابة A20:
  - القيمة 0: تعطيل.
  - القيمة 1: تفعيل.
- البتات 2-3: غير معرف.
- البت 4: Input Buffer Full.
- البت 5: Output Buffer Empty.
- البت 6: Keyboard Clock:
  - القيمة 0: High-Z.
  - القيمة 1: Pull Clock Low.
- البت 7: Keyboard Data:
  - القيمة 0: High-Z.
  - القيمة 1: Pull Data Low.

وعند تفعيل البت رقم 1 فان هذا يفعل بوابة A20 ويجب استخدام الامر or حتى يتم الحفاظ على بقية البتات .وبعد ذلك يجب كتابة القيم الى نفس المنفذ باستخدام الامر 0xd1 .
والشفرة التالية توضح كيفية تفعيل بوابة A20 عن طريق منفذ الخروج Output Port لمتحكم لوحة المفاتيح.

Listing ٤٨.: Some Code

```
;*************************************************
; enable_a20_keyboard_controller_output_port:
;   Enable A20 with write to keyboard output port.
;*************************************************

enable_a20_keyboard_controller_output_port:

    cli
    pusha     ; save all registers

    call wait_input   ; wait last operation to be finished.

    ;--------------------
    ; Disable Keyboard
    ;--------------------
    mov al,0xad   ; disable keyboard command.
    out 0x64,al
    call wait_input

    ;-----------------------------------
    ; send read output port command
    ;-----------------------------------
    mov al,0xd0   ; read output port command
    out 0x64,al
    call wait_output    ; wait output to come.
    ; we don't need to wait_input bescause when output came
       we know that operation are executed.

    ;-------------------------
    ; read input buffer
    ;-------------------------
    in al,0x60
    push eax    ; save data.
    call wait_input

    ;-----------------------------------
    ; send write output port command.
    ;-----------------------------------
    mov al,0xd1   ; write output port command.
```

```
out 0x64,al
call wait_input

;—————————
; enable a20.
;—————————
pop eax
or al,2      ; set bit 2.
out 0x60,al
call wait_input

;—————————————
; Enable Keyboard.
;—————————————
mov al,0xae    ; Enable Keyboard command.
out 0x64,al
call wait_input


popa       ; restore registers
sti

ret
```

حيث في البداية تم تعطيل لوحة المفاتيح (عن طريق ارسال الامر 0xad) واستدعاء الدالة wait input للتأكد من أن الامر قد تم تنفيذه ومن ثم تم ارسال أمر قراءة منفذ الخروج لمتحم لوحة المفاتيح (الامر 0xda) وانتظار المتحكم حتى ينتهي من تنفيذ الامر ، وقد تم استخدام الدالة wait output لانتظار قيمة منفذ الخروج ، وبعدها تم قراءة هذه القيمة وحفظها في المكدس (Stack) ،وبعد ذلك تم ارسال أمر الكتابة الى منفذ الخروج لمتحكم لوحة المفاتيح (الامر 0xd1) وانتظار المتحكم حى ينتهي من تنفيذ الامر ومن قمنا بارسال قيمة المنفذ الخروج الجديدة بعد أن تم تفعيل البت رقم 1 وهو البت الذي يفعل بوابة A20 ، وفي الاخير تم تفعيل لوحة المفاتيح مجددا.

٣.٤. أساسيات ال VGA

في عام 1987 قامت IBM بتطوير مقياس لمتحكمات شاشة الحاسب وهو Video Graphics Array واختصاراً VGA وجائت تسميته ب Array نظرا لانه تم تطويره كشريحة واحدة signle chip حيث استبدلت العديد من الشرائح والتي كانت تستخدم في مقاييس اخرى مثل MDA و CGA و EGA

، ويتكون ال VGA من Video Buffer, Video DAC , CRT Controller , Sequencer unit , Graphics Controller و Attribute Controller[٣].
ال Video Buffer هو مقطع من الذاكرة segment of memory يعمل كذاكرة للشاشة Memory Mapped ، وعند بداية التشغيل فان البايوس يخصص مساحة من الذاكرة بدءا من العنوان 0xa0000 كذاكرة للشاشة وفي حالة تم الكتابة الى هذه الذاكرة فان هذا سوف يغير في الشاشة ، هذا الربط يسمى Memory Mapping، أما ال Graphics Controller فهو الذي يقوم بتحديث محتويات الشاشة بناءاً على البيانات الموجودة في ال Video buffer.
وتدعم ال VGA نمطين للعرض الاول هو النمط النصي Text Mode والاخر هو النمط الرسومي APA Graphics Mode ويحدد النمط طريقة التعامل مع ال Video buffer وكيفة عرض البيانات.
النمط الرسومي All Point Addressable Graphics Mode يعتمد على البكسلات ، حيث يمكن التعامل مع كل بسكل موجود على حدة . والبكسل هو أصغر وحدة في الشاشة وتعادل نقطة على الشاشة. أما النمط النصي Text Mode فيعتمد على الحروف Characters ، ولتطبيق هذا النمط فان متحكم الشاشة Video Controller يستخدم ذاكرتين two buffers الاولى وهي خريطة الحروف Character Map وهي تعرف البكسلات لكل حرف ويمكن تغيير هذه الخريطة لدعم أنظمة محارف أخرى، أما الذاكرة الثانية فهيScreen Buffer وبمجرد الكتابة عليها فان التأثير سيظهر مباشرة على الشاشة.
ومقياس VGA هو مبني على المقاييس السابقة ، ابتداءا من مقياس Monochrome Display Adapter ويسمى اختصارا MDA والذي طورته IBM في عام 1981 ، و MDA لا تدعم النمط الرسومي والنمط النصي بها (يسمى Mode 7) يدعم 80 عمود و 24 صف (80*25). وفي نفس العام قامت IBM بتطوير مقياس Color Graphics Adapter (واختصارا CGA) الذي كان أول متحكم يدعم الالوان حيث يمكن عرض 16 لون مختلف.وبعد ذلك تم تطوير Enhanced Graphics Adapter.
ويجدر بنا التذكير بان متحكمات VGA متوافقة مع المقاييس السابقة Backward Compatible فعندما يبدأ الحاسب في العمل فان النمط سيكون النمط النصي Mode 7 (الذي ظهر في MDA) ، وهذا يعني اننا سنتعامل مع 80 عمود و 25 صف.

١.٣.٤. عنونة الذاكرة في متحكمات VGA

عندما يبدأ الحاسب بالعمل فان البايوس يخصص العناوين من 0xa0000 الى 0xbffff لذاكرة الفيديو Video memroy (موجودة على متحكم VGA) ، هذه العناوين مقسمة كالاتي:

- من 0xb0000 الى 0xb7777: للنمط النصي أحادي اللون Monochrome Text Mode.
- من 0xb8000 الى 0xbffff: Color Text Mode.

وعند الكتابة في هذه العناوين فان هذا سوف يؤثر في الشاشة واظهار القيم التي تم كتابتها ، والمثال التالي يوضح كيفية كتابة حرف A بلون أبيض وخلفية سوداء.

[٣]شرح هذه المكونات سيكون في الفصل الخامس باذن الله ، وسيتم التركيز على بعض الاشياء بحسب الحاجة حاليا.

Listing ٤٩.: Some Code

```
%define VIDEO_MEMORY  0xb8000   ; Base Address of Mapped
   Video Memory.
%define CHAR_ATTRIBUTE  0x7     ; White chracter on black
   background.

mov edi,VIDEO_MEMORY

mov [edi],'A'           ; print A
mov [edi+1],CHAR_ATTRIBUTE      ; in white foreground black
   background.
```

٢.٣.٤. طباعة حرف على الشاشة

لطباعة حرف على الشاشة يجب ارسال الحرف الى عنوان ال Video Memory وحتى نتمكن من طباعة العديد من الحروف فانه يجب انشاء متغيران (x,y) لحفظ المكان الحالي للصف والعمود ومن ثم تحويل هذا المكان الى عنوان في ال Video Memoey. وفي البداية ستكون قيم (x,y) هي (0 , 0) أي ان الحرف سيكون في الجزء الاعلي من اليسار في الشاشة ويجب ارسال هذا الحرف الى عنوان بداية ال Video Memory وهو 0xb8000 (Color text Mode). ولطباعة حرف آخر فان قيم (x,y) له هي (0 , 1) ويجب ارسال الحرف الى العنوان 0xb8001 ، وسنستخدم العلاقة التالية للتحويل بين قيم (x,y) الى عناوين لذاكرة العرض Video Memory:

$$videomemory = 0xb0000$$
$$videomemory+ = x + y * 80$$

وبسبب أن هناك 80 حرف في كل عمود فانه يجب ضرب قيمة y ب 80 . والمثال التالي يوضح كيفية طباعة حرف عند (4,4) .

$$address = x + y * 80$$

$$address = 4 + 4 * 80 = 324$$

; now add the base address of video memory.

$$address = 324 + 0xb8000 = 0xb8144$$

وبارسال الحرف الى العنوان 0xb8144 فان الحرف سوف يظهر على الشاشة في الصف الخامس والعمود الخامس (الترقيم يبدأ من صفر وأول صف وعمود رقمها صفر).

وكما ذكرنا ان النمط النصي Mode 7 هو الذي يبدأ الحاسب به ، في هذا النمط يتعامل متحكم العرض مع بايتين من الذاكرة لكل حرف يراد طباعته ، بمعنى اذا ما أردنا طباعة الحرف A فانه يجب ارسال الحرف الى العنوان 0xb8000 وخصائص الحرف الى العنوان التالي له 0xb8001 وهذا يعني انه يجب تعديل قانون التحويل السابق واعتبار أن كل حرف يأخذ بايتين من الذاكرة وليس بايت واحد.
البايت الثاني للحرف يحدد لون الحرف وكثافة اللون (غامق وفاتح) والجدول التالي يوضح البتات فيه:

- البتات 0-2: لون الحرف:
  - البت 0: أحمر.
  - البت 1: أخضر.
  - البت 2: أزرق.
- البت 3: كثافة لون الحرف (0 غامق ، 1 فاتح).
- البت 4-6: لون خلفية الحرف:
  - البت 0: أحمر.
  - البت 1: أخضر.
  - البت 2: أزرق.
- البت 7: كثافة لون خلفية الحرف (0 غامق ، 1 فاتح).

وهكذا توجد 4 بت لتحديد اللون ، والجدول التالي يوضح هذه الألوان:

- 0: Black.
- 1: Blue.
- 2: Green.
- 3: Cyan.
- 4: Red.
- 5: Magneta.
- 6: Brown.
- 7: Light gray.
- 8: Dark Gray.
- 9: Light Blue.
- 10: Light Green.

- 11: Light Cyan.
- 12: Light Red.
- 13: Light Magneta.
- 14: Light Brown.
- 15: White.

اذاً لطباعة حرف على النمط Mode 7 فانه يجب ارسال الحرف وخصائصه الى ذاكرة العرض ، كما يجب مراعاة بعض الامور من تحديث المؤشر Cursor (هو خط underline يظهر ويختفي للدلالة على الموقع الحالي) و الانتقال الى الصف التالي في حالة الوصول الى اخر حرف في العمود أو في حالة كان الحرف المراد طباعته هو حرف الانتقال الى سطر جديد 0xa . والمثال التالي يوضح الدالة putch32 والتي تستخدم لطباعة حرف على الشاشة في النمط المحمي PMode.

Listing ٤١٠.: Some Code

```
; ******************************************
; putch32: print character in protected mode.
;   input:
;       bl: character to print.
; ******************************************

bits  32

%define VIDEO_MEMORY  0xb8000   ; Base Address of Mapped
   Video Memory.
%define COLUMNS     80      ; text mode (mode 7) has 80
   columns,
%define ROWS      25      ; and 25 rows.
%define CHAR_ATTRIBUTE  31      ; white on blue.

x_pos   db    0     ; current x position.
y_pos   db    0     ; current y position.

putch32:

    pusha      ; Save Registers.

  ;-----------------------------
  ; Check if bl is new line ?
```

```
;-----------------------------

  cmp bl,0xa    ; if character is newline ?
  je new_row    ; yes, jmp at end.

;-----------------------------------
; Calculate the memory offset
;-----------------------------------
  ; because in text mode every character take 2 bytes: one
      for the character and one for the attribute, we must
      calculate the memory offset with the follwing
     formula:
  ; offset = x_pos*2 + y_pos*COLUMNS*2

  xor eax,eax

  mov al,2
  mul byte[x_pos]
  push eax      ; save the first section of formula.

  xor eax,eax
  xor ecx,ecx

  mov ax,COLUMNS*2    ; 80*2
  mov cl,byte[y_pos]
  mul ecx

  pop ecx
  add eax,ecx

  add eax,VIDEO_MEMORY  ; eax = address to print the
     character.

;----------------------
; Print the chracter.
;----------------------

  mov edi,eax

  mov byte[edi],bl          ; print the character,
  mov byte[edi+1],CHAR_ATTRIBUTE    ; with respect to the
     attribute.
```

```
;_____________________
; Update the postions.
;_____________________

  inc byte[x_pos]
  cmp byte[x_pos],COLUMNS
  je new_row

  jmp putch32_end


new_row:

  mov byte[x_pos],0       ; clear the x_pos.
  inc byte[y_pos]         ; increment the y_pos.

putch32_end:

  popa      ; Restore Registers.

  ret
```

وتبدأ هذه الدالة بفحص الحرف المراد طباعته (موجود في المسجل bl) مع حرف الانتقال الى السطر الجديد 0xa وفي حالة التساوي يتم نقل التنفيذ الى آخر جسم الدالة والذي يقوم بتصفير قيمة x وزيادة قيمة y دلالة على الانتقال الى السطر الجديد. أما في حالة كان الحرف هو أي حرف آخر فانه يجب حساب العنوان الذي يجب ارسال الحرف اليه حتى يمكن طباعته ، وكما ذكرنا أن النمط النصي Mode 7 يستخدم بايتين لكل حرف لذا سيتم استخدام العلاقة التالية للتحويل ما بين (x,y) الى العنوان المطلوب.

$$videomemory = 0xb0000$$
$$videomemory+ = x * 2 + y * 80 * 2$$

وكما يظهر في الكود السابق فقد تم حساب هذا العنوان وحفظه في المسجل eax وبعد ذلك تم طباعة الحرف المطلوب بالخصائص التي تم تحديدها مسبقا كثابت. وآخر خطوة في الدالة هي زيادة قيم (x,y) للدالة الى المكان التالي ، وهذا يتم بزيادة x فقط وفي حالة تساوت القيمة مع قيمة آخر عمود في الصف فانه يتم زيادة قيمة y وتصفير x دلالة على الانتقال الى الصف التالي.

٤.٣.٣. طباعة السلاسل النصية strings

لطباعة سلسلة نصية سنستخدم دالة طباعة الحرف وسنقوم بأخذ حرف حرف من السلسة وارسالها الى دالة طباعة الحرف حتى تنتهي السلسلة ، والشفرة التالية توضح الدالة puts32 لطباعة سلسلة نصية.

Listing ٤١١.: Some Code

```
; ****************************************
; puts32: print string in protected mode.
;   input:
;        ebx: point to the string
; ****************************************

bits  32

puts32:

    pusha     ; Save Registers.

    mov edi,ebx

  @loop:
    mov bl,byte[edi]  ; read character.

    cmp bl,0x0      ; end of string ?
    je  puts32_end    ; yes, jmp to end.

    call putch32    ; print the character.

    inc edi       ; point to the next character.

    jmp @loop

  puts32_end:

  ;-----------------------------
  ; Update the Hardware Cursor.
  ;-----------------------------
    ; After print the string update the hardware cursor.
```

```
mov bl,byte[x_pos]
mov bh,byte[y_pos]

call move_cursor

popa      ; Restore Registers.

ret
```

في هذه الدالة سيتم قراءة حرف حرف من السلسة النصية وطباعته الى أن نصل الى نهاية السلسلة (القيمة 0x0) ، وبعد ذلك سيتم تحديث المؤشر وذلك عن طريق متحكم CRT Controller ونظراً لان التعامل معه بطئ قليلا فان تحديث المؤشر سيكون بعد طباعة السلسلة وليس بعد طباعة كل حرف .

٤.٣.٤. تحديث المؤشر Hardware Cursor

عند طباعة حرف او سلسلة نصية فان مؤشر الكتابة لا يتحرك من مكانه الا عند تحديده يدويا ، وهذا يتم عن طريق التعامل مع متحكم CRT Controller . هذا المتحكم يحوي العديد من المسجلات ولكننا سوف نركز على مسجل البيانات Data Register ومسجل نوع البيانات Index Register.
ولارسال بيانات الى هذا المتحكم ، فيجب اولا تحديد نوع البيانات وذلك بارسالها الى مسجل Index Register ومن ثم ارسال البيانات الى مسجل البيانات Data Register ، وفي حواسيب x86 فان مسجل البيانات يأخذ العنوان 0x3d5 ومسجل Index Register يأخذ العنوان 0x3d4.
والجدول التالي يوضح القيم التي يمكن ارسالها الى مسجل نوع البيانات Index Register.

- 0x0: Horizontal Total.
- 0x1: Horizontal Display Enable End.
- 0x2: Start Horizontal Blanking.
- 0x3: End Horizontal Blanking.
- 0x4: Start Horizontal Retrace Pulse.
- 0x5: End Horizontal Retrace.
- 0x6: Vertical Total.
- 0x7: Overflow.
- 0x8: Preset Row Scan.
- 0x9: Maximum Scan Line.

- 0xa: Cursor Start.
- 0xb: Cursor End.
- 0xc: Start Address High.
- 0xd: Start Address Low.
- 0xe: Cursor Location High.
- 0xf : Cursor Location Low.
- 0x10: Vertical Retrace Start.
- 0x11: Vertical Retrace End.
- 0x12: Vertical Display Enable End.
- 0x13: Offset.
- 0x14: Underline Location.
- 0x15: Start Vertical Blanking.
- 0x16: End Vertical Blanking.
- 0x17: CRT Mode Control.
- 0x18: Line Compare.

وعند ارسال أي من القيم السابقة الى مسجل Index Reigster فان هذا سيحدد نوع البيانات التي سترسل الى مسجل البيانات Data Register. ومن الجدول السابق سنجد أن القيمة `0xf` ستحدد قيمة x للمؤشر ، والقيمة `0xe` ستحدد قيمة y للمؤشر.وبعد ذلك يجب ارسال قيم x,y الى مسجل البيانات على التوالي مع ملاحظة أن متحكم CRT يتعامل مع بايت واحد لكل حرف وهذا يعني أننا سنستخدم القانون التالي للتحويل من قيم (x,y) الى عناوين.

$$videomemory = x + y * 80$$

والشفرة التالية توضح عمل الدالة move cursor والتي تعمل على تحريك المؤشر.

Listing ٤١٢.: Some Code

```
; ****************************************
; move_cursor: Move the Hardware Cursor.
;   input:
;       bl: x pos.
;       bh: y pos.
```

```
; ****************************************

bits  32

move_cursor:

    pusha     ; Save Registers.

  ;------------------------
  ; Calculate the offset.
  ;------------------------
    ; offset = x_pos + y_pos*COLUMNS

    xor ecx,ecx
    mov cl,byte[x_pos]

    mov eax,COLUMNS
    mul byte[y_pos]

    add eax,ecx
    mov ebx,eax

  ;----------------------
  ; Cursor Location Low.
  ;----------------------

    mov al,0xf
    mov dx,0x3d4
    out dx,al

    mov al,bl
    mov dx,0x3d5
    out dx,al

  ;-----------------------
  ; Cursor Location High.
  ;-----------------------

    mov al,0xe
    mov dx,0x3d4
    out dx,al
```

```
    mov al,bh
    mov dx,0x3d5
    out dx,al


    popa        ; Restore Registers.

    ret
```

٤.٣.٥. تنظيف الشاشة Clear Screen

تنظيف الشاشة هي عملية ارسال حرف المسافة بعدد الحروف الموجودة (80*25 في نمط Mode 7) و تصفير قيم (x,y) . والشفرة التالية توضح كيفية تنظيف الشاشة وتحديد اللون الازرق كخلفية لكل حرف.

Listing ٤١٣.: Some Code

```
; ***********************************************
; clear_screen: Clear Screen in protected mode.
; ***********************************************

bits  32

clear_screen:

    pusha      ; Save Registers.
    cld

    mov edi,VIDEO_MEMORY     ; base address of video memory.
    mov cx,2000    ; 25*80
    mov ah,CHAR_ATTRIBUTE    ; 31 = white character on blue
       background.
    mov al,' '

    rep stosw

    mov byte[x_pos],0
    mov byte[y_pos],0
```

```
popa      ; Restore Registers.
ret
```

٤.٤. تحميل النواة

الى هنا تنتهي مهمة المرحلة الثانية من محمل النظام Second Stage Bootloader ويتبقى فقط البحث عن النواة ونقل التحكم اليها[٤]. وفي هذا الجزء سيتم كتابة نواة تجريبية بهدف التأكد من عملية نقل التحكم الى النواة وكذلك بهدف إعادة كتابة شفرة محمل النظام بشكل أفضل.
وسيتم استخدام لغة التجميع لكتابة هذه النواة التجريبية حيث أن الملف الناتج سيكون Pure Binary ولا يحتاج الى محمل خاص ، وابتداءاً من الفصل القادم سنترك لغة التجميع جانبا ونبدأ العمل بلغة السي والسي++ .
وبما أننا نعمل في النمط المحمي PMode فلا يمكننا أن نستخدم مقاطعة البايوس int 0x13 لتحميل قطاعات النواة الى الذاكرة ، ويجب أن نقوم بكتابة درايفر لمحرك القرص المرن أو نقوم بتحميل النواة الى الذاكرة قبل الانتقال الى النمط المحمي وهذا ما سنفعله الان ، وسنترك جزئية برمجة محرك القرص المرن لاحقا.
وحيث أن النمط المحمي يسمح باستخدام ذاكرة حتى 4 جيجا ، فان النواة سنقوم بتحميلها على العنوان 0x100000 أي عند 1 ميجا من الذاكرة .لكن علينا التذكر بأن النمط الحقيقي لا يدعم الوصول الى العنوان 0x100000 لذلك سنقوم بتحميل النواة أولا في أي عنوان خالي وليكن 0x3000 وعند الانتقال الى النمط المحمي سنقوم بنسخها الى العنوان 0x100000 ونقل التنفيذ والتحكم اليها.
والشفرة التالية توضح نواة ترحيبية.

Listing ٤١٤.: Some Code

```
org   0x100000      ; kernel will load at 1 MB.
bits  32            ; PMode.
jmp kernel_entry
%include "stdio.inc"
```

[٤] الفصل التالي سيتناول موضوع النواة وكيفية برمجتها بالتفصيل.

```
kernel_message  db    0xa,0xa,0xa,"                eqraOS
   v0.1 Copyright (C) 2010 Ahmad Essam"
        db    0xa,0xa,    "          University of Khartoum
           — Faculty of Mathematical Sceinces.",0


logo_message  db    0xa,0xa,0xa,"                  ---  ---
   --------  -  /  --  \  /  --/"
        db    0xa,          "               / —_)  _  `/  --/  _
           `/  /  /_/  /_\  \   "
        db    0xa,          "               \__/\_,  /_/   \_,_/
             \____//___/   "
        db    0xa,          "                     /_/
                                    ",0
;*********************
; Entry point.
;*********************

kernel_entry:

  ;________________
  ; Set Registers
  ;________________

    mov ax,0x10     ; data selector.
    mov ds,ax
    mov es,ax
    mov ss,ax
    mov esp,0x90000   ; set stack.

  ;_____________________________________
  ; Clear Screen and print message.
  ;_____________________________________

    call clear_screen

    mov ebx,kernel_message
    call puts32

    mov ebx,logo_message
    call puts32
```

```
;--------------------
; Halt the system.
;--------------------

  cli
  hlt
```

والمرحلة الثانية من محمل النظام ستكون هي المسؤولة عن البحث عن النواة وتحميلها ونقل التنفيذ اليها ، وسيتم تحميلها الى الذاكرة قبل الانتقال الى النمط المحمي وذلك حتى نتكمن من استخدام مقاطعة البايوس `int 0x13` وعند الانتقال الى النمط المحمي سيتم نسخ النواة الى عنوان 1 ميجا ونقل التحكم الى النواة.
ولتحميل النواة الى الذاكرة يجب أولا تحميل Root Directory الى الذاكرة والبحث عن ملف النواة وفي حالة كان الملف موجودا سيتم قراءة عنوان أول كلستر له ، هذا العنوان سيعمل ك index في جدول FAT (والذي يجب تحميله الى الذاكرة ايضا) وسيتم قراءة القيمة المقابلة لهذا ال index والتي ستخبرنا هل ما اذا كان هذا الكلستر هو آخر كلستر للملف أم لا[٥].
والشفرة التالية توضح ملف المرحلة الثانية من المحمل stage2.asm ، وتم تقسيم الكود بشكل أكثر تنظيما حيث تم نقل أي دالة تتعلق بالقرص المرن الى الملف floppy.inc (ملف inc. هو ملف للتضمين في ملف آخر) ، والدوال المتعلقة بنظام الملفات موجودة على الملف fat12.inc ودوال الاخراج موجودة في stdio.inc ودوال تفعيل بوابة A20 موجودة على الملف a20.inc ودالة تعيين جدول الواصفات العام وكذلك تفاصيل الجدول موجودة في الملف gdt.inc ، اخيرا تم انشاء ملف common.inc لحفظ بعض الثوابت المستخدمة دائما[٦].

Listing ٤١٥.: Some Code

```
bits 16   ; 16-bit real mode.
org 0x500

start:  jmp stage2

;*************************
; include files:
;*************************
%include "stdio.inc"    ; standard I/O routines.
%include "gdt.inc"      ; GDT load routine.
%include "a20.inc"      ; Enable A20 routines.
```

[٥] راجع الفصل السابق لمعرفة التفاصيل.
[٦] جميع شفرات الملفات مرفقة مع البحث في مجلد example/ch3/boot/ وشفرة المحمل النهائية ستكون ملحقة في نهاية البحث.

```
%include "fat12.inc"    ; FAT12 driver.
%include "common.inc"   ; common declarations.

;*********************
; data and variable
;*********************

hello_msg   db    0xa,0xd,"Welcome to eqraOS Stage2",0xa,0xd
   ,0
fail_message  db    0xa,0xd,"KERNEL.SYS is Missing. press
   any key to reboot...",0



; ************************************
; entry point of stage2 bootloader.
; ************************************

stage2:

  ;-----------------
  ; Set Registers.
  ;-----------------

    cli

    xor ax, ax
    mov ds, ax
    mov es, ax

    mov ax, 0x0
    mov ss, ax
    mov sp, 0xFFFF

    sti

  ;-----------------------
  ; Load gdt into gdtr.
  ;-----------------------

    call load_gdt
```

```
;-----------------------
; Enable A20.
;-----------------------
  call enable_a20_keyboard_controller_output_port

;-------------------
; Display Message.
;-------------------
  mov si,hello_msg
  call puts16

;-------------------------
; Load Root Directory
;-------------------------
  call load_root

;-------------------------
; Load Kernel
;-------------------------
  xor ebx,ebx
  mov bp,KERNEL_RMODE_BASE    ; bx:bp buffer to load
      kernel

  mov si,kernel_name
  call load_file

  mov dword[kernel_size],ecx
  cmp ax,0
  je enter_stage3

  mov si,fail_message
  call puts16

  mov ah,0
  int 0x16      ; wait any key.
  int 0x19      ; warm boot.
  cli         ; cannot go here!
  hlt


;-------------------
; Go to PMode.
```

```
  ;————————————————

  enter_stage3:

    ; just set bit 0 from cr0 (Control Register 0).

    cli         ; important.
    mov eax,cr0
    or eax,0x1
    mov cr0,eax     ; entering pmode.


  ;———————————————————
  ; Fix CS value
  ;———————————————————
    ; select the code descriptor
    jmp CODE_DESCRIPTOR:stage3


;*************************
; entry point of stage3
;*************************

bits 32     ; code now 32—bit

stage3:

  ;————————————————;
  ; Set Registers.
  ;————————————————;

    mov ax,DATA_DESCRIPTOR      ; address of data
       descriptor.
    mov ds,ax
    mov ss,ax
    mov es,ax
    mov esp,0x90000   ; stack begin from 0x90000.

  ;————————————————————————————————
  ; Clear Screen and print message.
  ;————————————————————————————————
```

```
  call clear_screen

  mov ebx,stage2_message
  call puts32

  mov ebx,logo_message
  call puts32



;————————————————————
; Copy Kernel at 1 MB.
;————————————————————
  mov eax,dword[kernel_size]
  movzx ebx,word[bytes_per_sector]
  mul ebx
  mov ebx,4
  div ebx

  cld

  mov esi,KERNEL_RMODE_BASE
  mov edi,KERNEL_PMODE_BASE
  mov ecx,eax
  rep movsd

;————————————————————
; Execute the kernel.
;————————————————————
  jmp CODE_DESCRIPTOR:KERNEL_PMODE_BASE

;————————————————————;
; Hlat the system.
;————————————————————;
  cli      ; clear interrupt.
  hlt      ; halt the system.
```

النتيجة:

شكل ٤.١.: محمل النظام أثناء العمل

شكل ٤.٢.: بدء تنفيذ النواة

القسم III.

النواة Kernel

٥. مقدمة حول نواة نظام التشغيل

أحد أهم المكونات في نظام التشغيل هي نواة النظام (Kernel) حيث تدير هذه النواة عتاد وموارد الحاسب وتوفر واجهة برمجية عالية تسمح لبرامج المستخدم من الاستفادة من هذه الموارد بشكل جيد.وتعتبر برمجة نواة النظام من أصعب المهمات البرمجية على الاطلاق ، حيث تؤثر هيكلته وتصميمه على كافة نظام التشغيل وهذا ما يميز بعض الانظمة ويجعلها قابلة للعمل في أجهزة معينة. وفي هذا الفصل سنلقي نظرة على النواة وبرمجتها باستخدام لغة السي و السي++ وكذلك سيتم الحديث عن طرق تصميم النواة وميزات وعيوب كلٌ على حدة.

٥.١. نواة نظام التشغيل

تعرَّف نواة نظام التشغيل بأنها الجزء الأساسي في النظام والذي تعتمد عليه بقية مكونات نظام التشغيل. ويكمن دور نواة النظام في التعامل المباشر مع عتاد الحاسب وإدارته بحيث تكوِّن طبقة برمجية تبعد برامج المستخدم من تفاصيل وتعقيدات العتاد ، ولا تقتصر على ذلك بل توفر واجهة برمجية مبسطة (يمكن استخدامها من لغة البرمجة المدعومة على النظام) بحيث تمكن برامج المستخدم الاستفادة من موارد الحاسب . وفي الحقيقة لا يوجد قانون ينص على إلزامية وجود نواة للنظام ، حيث يمكن لبرنامج ما (يعمل في الحلقة صفر) التعامل المباشر مع العتاد ومع كل الجداول في الحاسب والوصول الى أي بايت في الذاكرة لكن هذا ما سيجعل عملية كتابة البرامج عملية شبه مستحيلة ! حيث يجب على كل مبرمج يريد كتابة تطبيق بسيط أن يجيد برمجة العتاد وأساسيات الاقلاع حتى يعمل برنامجه ، اضافة على ذلك لا يمكن تشغيل أكثر من برنامج في نفس الوقت نظرا لعدم وجود بنية تحتية توفر مثل هذه الخصائص ، ولاننسى اعداد وتهيئة جداول النظام وكتابة وظائف التعامل مع المقاطعات والأخطاء، ودوال حجز وتحرير الذاكرة وغيرها من الخصائص الضرورية لأي برنامج.كل هذا يجعل عملية تطوير برنامج للعمل على حاسب ما بدون نواة له أمراً غير مرغوبا ، خاصة إذا ذكرنا أن البرنامج يجب تحديثه مجدداً عند نقله الى منصة أخرى ذات عتاد مختلف. اذاً يمكن أن نقول أن نواة النظام هي الجزء الاهم في نظام التشغيل ككل ،حيث تدير النواة عتاد الحاسب من المعالج والذاكرة الرئيسية والأقراص الصلبة والمرنة وغيرها من الأجهزة المحيطة بالحاسب.

وحتى نفهم علاقة النواة مع بقية أجزاء النظام ، فانه يمكن تقسيم الحاسب الى عدة مستويات من التجريد بحيث كل مستوى يخدم المستوى الذي يليه .

١.١.٥. مستويات التجريد

العديد من البرمجيات يتم بنائها على شكل مستويات ، وظيفة كل مستوى هو توفير واجهة للمستوى الذي يليه بحيث تخفي هذه الواجهة العديد من التعقيدات والتفاصيل وكذلك ربما يحمي مستوى ما بعض الخصائص من المستوى الذي يليه ، وغالبا ما يتبع نظام التشغيل لهذا النوع من البرمجيات حيث يمكن تقسيم النظام ككل الى عدة مستويات.

المستوى الأول: مستوى العتاد

مستوى العتاد هو أدنى مستوى يمكن أن نعرفه ويظهر على شكل متحكمات لعتاد الحاسب ، حيث يرتبط متحكم ما في اللوحة الأم مع متحكم آخر في العتاد نفسه. وظيفة المتحكم في اللوحة الأم هي التخاطب مع المتحكم الاخر في العتاد والذي بدوره يقوم بتنفيذ الأوامر المستقبلة. كيف يقوم المتحكم بتنفيذ الأوامر ؟ هذا هو دور المستوى الثاني.

المستوى الثاني: مستوى برامج العتاد Firmware

برامج العتاد (Firmware) هي برامج موجودة على ذاكرة بداخل المتحكم (غالبا ذاكرة EEPROM) ، وظيفة هذه البرامج هي تنفيذ الأوامر المرسلة الى المتحكم. ومن الامثلة على مثل هذه البرمجيات برنامج البايوس وأي برنامج موجود في أي متحكم مثل متحكم لوحة المفاتيح.

المستوى الثالث: مستوى النواة (الحلقة صفر)

النواة وهي أساس نظام التشغيل ، وظيفتها ادارة موارد الحاسب وتوفير واجهة لبقية أجزاء النظام ، وتعمل النواة في الحلقة صفر ، اي أنه يمكن تنفيذ أي أمر والوصول المباشر الى أي عنوان في الذاكرة.

المستوى الرابع: مستوى مشغلات الأجهزة (الحلقة ١ و ٢)

مشغلات الأجهزة هي عبارة عن برامج للنظام وظيفتها التعامل مع متحكمات العتاد (وذلك عن طريق النواة) سواءا لقراءة النتائج او لارسال الأوامر ، هذه البرامج تحتاج الى أن تعمل في الحلقة ١ و ٢ حتى تتمكن من تنفيذ العديد من الأوامر ، وفي حالة تم تنفيذها على الحلقة صفر فان هذا قد يؤدي الى خطورة تعطل النظام في حالة كان هناك عطل في احد المشغلات كذلك ستكون صلاحيات المشغل عالية فقد يقوم أحد المشغلات بتغيير أحد جداول المعالج مثل جدول الواصفات العام (GDT) والذي بدوره قد يعطل النظام.

المستوى الخامس: مستوى برامج المستخدم (الحلقة ٣)

المستوى الاخير وهو مستوى برامج المستخدم حيث لا يمكن لهذه البرامج الوصول الى النواة وانما تتعامل فقط مع واجهة برمجة التطبيقات (Application Progeamming Interface) والتي تعرف بدوال (API).

٢.٥. وظائف نواة النظام

تختلف مكونات ووظائف نواة نظام التشغيل تبعاً لطريقة التصميم المتبعة ،فهناك العديد من الطرق لتصميم الانوية بعضاً منها يجعل ما هو متعارف عليه بأنه يتبع لنواة النظام ببرنامج للمستخدم (User Program)[١] والبعض الاخر عكس ذلك . لذلك سنذكر حالياً المكونات الشائعة في نواة النظام وفي القسم التالي عند الحديث عن هيكلة وطرق تصميم الأنوبة سنفصل أكثر في هذه المكونات ونقسمها بحسب طريقة التصميم.

١.٢.٥. إدارة الذاكرة

أهم وظيفة لنواة النظام هي إدارة الذاكرة حيث أن أي برنامج يجب ان يتم تحمليه على الذاكرة الرئيسية قبل أن يتم تنفيذه ، لذلك من مهام مدير الذاكرة هي معرفة الأماكن الشاغرة ، والتعامل مع مشاكل التجزئة (Fragmentation) حيث من الممكن أن تحوي الذاكرة على الكثير من المساحات الصغيرة والتي لا تكفي لتحميل أي برنامج أو حتى حجز مساحة لبرنامج ما. أحد المشاكل التي على مدير الذاكرة التعامل معها هي معرفة مكان تحميل البرنامج ، حيث يجب أن يكون البرنامج مستقلا عن العناوين (Position Independent) لكي يتم تحمليه وإلا فلن نعرف ما هو عنوان البداية (Base Address) لهذا البرنامج. فلو فرضنا ان لدينا برنامج binary ونريد تحميله الى الذاكرة فهنا لن نتمكن من معرفة ما هو العنوان الذي يجب أن يكون عليه البرنامج ، لذلك عادة فان الناتج من عملية ترجمة وربط أي برنامج هو انها تبدأ من العنوان 0x0، وهكذا سنتمكن دوما من تحميل أي برنامج في بداية الذاكرة. بهذا الشكل لن نتمكن من تنفيذ أكثر من برنامج واحد ، حيث سيكون هناك برنامجا واحدا فقط يبدأ من العنوان 0x0 ، والحل لهذه المشاكل هو باستخدام مساحة العنونة التخيلية (Virtual Address Space) حيث يتم تخصيص مساحة تخيلية من الذاكرة لكل برنامج بحيث تبدأ العنونة تخيليا من 0x0 وبهذا تم حل مشكلة تحميل أكثر من برنامج وحل مشكلة relocation. ومساحة العنوان التخيلية (VAS) هي مساحة من العناوين لكل برنامج بحيث تيدأ من ال 0x0 ومفهوم هذه المساحة هو أن كل برنامج سيتعامل مع مساحة العناوين الخاصة به وهذا ما يؤدي الى حماية الذاكرة ، حيث لن يستطيع أي برنامج الوصول الى أي عنوان آخر بخلاف العناوين الموجودة في VAS. ونظراً لعدم ارتباط ال VAS مع الذاكرة الرئيسية فانه يمكن ان يشير عنوان تخيلي الى ذاكرة اخرى بخلاف الذاكرة الرئيسية (مثلا القرص الصلب). وهذا يحل مشكلة انتهاء المساحات الخالية في

[١] المقصود أنها برامج تعمل في الحلقة ٣.

الذاكرة. ويجدر بنا ذكر أن التحويل بين العناوين التخيليه الى الحقيقية يتم عن طريق العتاد بواسطة وحدة ادارة الذاكرة بداخل المعالج (Memory Management Unit).وكذلك مهمة حماية الذاكرة والتحكم في الذاكرة Cache وغيرها من الخصائص والتي سيتم الإطلاع عليها في الفصل الثامن – بمشيئة الله–.

٢.٢.٥. إدارة العمليات

٣.٢.٥. نظام الملفات

٣.٥. هيكلة وتصميم النواة

توجد العديد من الطرق لتصميم الأنوية وسنستعرض بعض منها في هذا البحث ، لكن قبل ذلك يجب الحديث عن طريقة مفيدة في هيكلة وتصميم الأنوية الا وهي تجريد العتاد (Hardware Abstraction) أي بمعنى فصل النواة من التعامل المباشر مع العتاد ، وانشاء طبقة برمجية (Software Layer) تسمى طبقة HAL (اختصارا لكلمة Hardware Abstraction Layer) بين النواة وبين العتاد ، وظيفة طبقة HAL هي توفير واجهة لعتاد الحاسب بحيث تمكِّن النواة من التعامل مع العتاد.

فصل النواة من العتاد تُتيح العديد من الفوائد ،أولاً شفرة النواة ستكون أكثر مقروئية وأسهل في الصيانة والتعديل لأن النواة ستتعامل مع واجهة أخرى أكثر سهولة من تعقيدات العتاد ، الميزة الثانية والأكثر أهمية هي امكانية نقل النواة (Porting) لأجهزة ذات عتاد مختلف (مثل SPARC,MIPS,...etc) بدون التغيير في شفرة النواة ، فقط سيتم تعديل طبقة HAL من ناحية التطبيق (Implementation) بالاضافة الى إعادة كتابة مشغلات الأجهزة (Devcie Drivers) مجدداً[٢].

١.٣.٥. النواة الضخمة Monolithic Kernel

تعتبر الأنوية المصممة بطريقة Monolitic[٣] أسرع وأكفأ أنوية في العمل وذلك نظرا لان كل برامج النظام (System Process) تكون ضمن النواة وتعمل في الحلقة صفر ، والشكل التالي يوضح مخطط لهذه الأنوية.

المشكلة الرئيسية لهذا التصميم هو انه عند حدوث أي مشكلة في أي من برامج النظام فان النظام سوف يتوقف عن العمل وذلك نظرا لانها تعمل في الحلقة صفر وكما ذكرنا سابقا أن أي خلل في هذا المستوى يؤدي الى توقف النظام عن العمل. مشكلة اخرى يمكن ذكرها وهي ان النواة غير مرنة بمعنى أنه لتغيير نظام الملفات مثلا يجب اعادة تشغيل النظام مجددا.

[٢]أغلب أنوية أنظمة التشغيل الحالية تستخدم طبقة HAL، هل تسائلت يوما كيف يعمل نظام جنو/لينوكس على أجهزة سطح المكتب والأجهزة المضمنة!

[٣]كلمة Mono تعني واحد ، أما كلمة Lithic فتعني حجري ، والمقصود بأن النواة تكون على شكل كتلة حجرية ليست مرنة وتطويرها وصيانتها معقد.

وكأمثلة على أنظمة تشغيل تعمل بهذا التصميم هي أنظمة يونكس ولينوكس ، وأنظمة ال DOS القديمة ووينـدوز ما قبل NT.

٢.٣.٥. النواة المصغرة MicroKernel

الأنوية MicroKernel هي الأكثر ثباتا واستقرار ومرونة والأسهل في الصيانة والتعديل والتطوير وذلك نظرا لان النواة تكون أصغر ما يمكن ، حيث أن الوظائف الأساسية فقط تكون ضمن النواة وهي ادارة الذاكرة وادارة العمليات (مجدول العمليات،أساسيات IPC)، أما بقية برامج النظام مثل نظام الملفات ومشغلات الأجهزة وغيرها تتبع لبرامج المستخدم وتعمل في نمط المستخدم (الحلقة ٣) ، وهذا يعني في حالة حدوث خطأ في هذه البرامج فان النظام لن يتأثر كذلك يمكن تغيير هذه البرامج (مثلا تغيير نظام الملفات) دون الحاجة الى اعادة تشغيل الجهاز حيث أن برامج النظام تعمل كبرامج المستخدم . والشكل التالي يوضح مخطط هذه الأنوية.
المشكلة الرئيسية لهذا التصميم هو بطئ عمل النظام وذلك بسبب أن برامج النظام عليها أن تتخاطب مع بعضها البعض عن طريق تمرير الرسائل (Message Passing) أو مشاركة جزء من الذاكرة (Shared Memory) وهذا ما يعرف ب Interprocess Communication.
وأشهر مثال لنظام تشغيل يتبع هذا التصميم هو نظام مينكس الاصدار الثالث.

٣.٣.٥. النواة الهجينة Hybrid Kernel

هذا التصميم للنواة ما هو إلا مزيج من التصميمين السابقين ، حيث تكون النواة MicroKernel لكنها تطبق ك Monolithic Kernel ، ويسمى هذا التصميم Hybrid Kernel أو Modified MicroKernel.
والشكل التالي يوضح مخطط لهذا التصميم.
وكأمثلة على أنظمة تعمل بهذا التصميم هو أنظمة ويندوز التي تعتمد على معمارية NT ، ونظام BeOS و Plane 9.

٤.٥. برمجة نواة النظام

يمكن برمجة نواة نظام التشغيل بأي لغة برمجة ، لكن يجب التأكد من أن اللغة تدعم استخدام لغة التجميع (Inline Assemlby) حيث أن النواة كثيرا ما يجب عليها التعامل المباشر مع أوامر لغة التجميع (مثلا عند تحميل جدول الواصفات العام وجدول المقاطعات وكذلك عند غلق المقطاعات وتفعيلها وغيرها).
الشيء الاخر الذي يجب وضعه في الحسبان هو أنه لا يمكن استخدام لغة برمجة تعتمد على مكتبات في وقت التشغيل (ملفات dll مثلا) دون إعادة برمجة هذه المكتبات(مثال ذلك لا يمكن استخدام لغات دوت نت دون إعادة برمجة إطار العمل). وكذلك لا يمكن الإعتماد على دوال النظام الذي تقوم بتطوير نظامك

الخاص فيه (مثلا لن تتمكن من استخدام new لحجز الذاكرة وذلك لانها تعتمد كليا على نظام التشغيل، أيضا دوال الادخال والاخراج تعتمد كليا على النظام).
لذلك غالبا تستخدم لغة السي والسي++ لبرمجة أنوية أنظمة التشغيل نظرا لما تتمتع به اللغتين من ميزات فريدة تميزها عن باقي اللغات ، وتنتشر لغة السي بشكل أكبر لاسباب كثيرة منها هو أنها لا تحتاج الى مكتبة وقت التشغيل (RunTime Library) حتى تعمل البرامج المكتوبة بها على عكس لغة سي++ والتي تحتاج الى (RunTime Library) لدعم الكثير من الخصائص مثل الاستثنائات و دوال البناء والهدم.
وفي حالة استخدام لغة سي أو سي++ فانه يجب إعادة تطوير اجزاء من مكتبة السي والسي++ القياسية (Standard C/C++ Library) وهي الأجزاء التي تعتمد على نظام التشغيل مثل دوال printf و scanf و دوال حجز الذواكر malloc/new وتحريرها free/delete.
ونظرا لاننا بصدد برمجة نظام 32 بت ، فان النواة أيضا يجب أن تكون 32 بت وهذا يعني أنه يجب استخدام مترجم سي أو سي++ 32 بت . مشكلة هذه المترجمات أن المخرج منها (البرنامج) لا يأتي بالشكل الثنائي فقط (Flat Binary) [٤]، وانما يضاف على الشفرة الثنائية العديد من الأشياء Headers,...etc.
ولتحميل مثل هذه البرامج فانه يجب البحث عن نقطة الإنطلاق للبرنامج (main routine) ومن ثم البدء بتنفيذ الأوامر منها.
وسيتم استخدام مترجم فيجوال سي++ لترجمة النواة ، وفي الملحق سيتم توضيح خطوات تهيئة المشروع وازالة أي اعتمادية على مكتبات أو ملفات وقت التشغيل.
وسنعيد كتابة النواة التي قمنا ببرمجتها بلغة التجميع في الفصل السابق ولكن بلغة السي والسي++ ، وسنناقش كيفية تحميل وتنفيذ هذه النواة حيث أن المخرج من مترجم الفيجوال سي++ هو ملف تنفيذي (Portable Executable) ولديه صيغة محددة يجب التعامل معها حتى نتمكن من تنفيذ الدالة الرئيسية للنواة (main()) ، كذلك سنبدأ في تطوير ملفات وقت التشغيل للغة سي++ وذلك حتى يتم دعم بعض خصائص اللغة والتي تحتاج الى دعم وقت التشغيل مثل دوال البناء والهدم والدوال الظاهرية (Pure Virtual Function) ، وفي الوقت الحالي لا يوجد دعم للإستثنائات (Exceptions) في لغة السي++ .

١.٤.٥. تحميل وتنفيذ نواة PE

بما أننا سنستخدم مترجم فيجوال سي++ والذي يخرج لنا ملف تنفيذي (Portable Executable) فانه يجب أن نعرف ما هي شكل هذه الصيغة حتى نتمكن عند تحميل النواة أن ننقل التنفيذ الى الدالة الرئيسية وليست الى أماكن أخرى.ويمكن استخدام مترجمات سي++ أخرى (مثل مترجم g++) لكن يجب ملاحظة أن هذا المترجم يخرج لنا ملف بصيغة ELF وهي صيغة الملفات التنفيذية على نظام جنو/لينوكس. والشكل التالي يوضح صيغة ملف PE الذي نحن بصدد التعامل معه.
يوجد أربع اضافات(headers) لصيغة PE سنطلع عليها بشكل سريع وفي حالة قمنا بتطوير محمل خاص لهذه الصيغة فسيتم دراستها بالتفصيل. و يمكن أن نصف هذه الاضافات بلغة السي++ كالتالي.

[٤] كبرنامج محمل النظام الذي قمنا بتطويره في بداية هذا البحث.

Listing ٥١.: Global new/delete operator

```
// header information format for PE files

typedef struct _IMAGE_DOS_HEADER {  // DOS .EXE header
    unsigned short e_magic;  // Magic number (Should be MZ
    unsigned short e_cblp;    // Bytes on last page of file
    unsigned short e_cp;      // Pages in file
    unsigned short e_crlc;     // Relocations
    unsigned short e_cparhdr;  // Size of header in
       paragraphs
    unsigned short e_minalloc;   // Minimum extra paragraphs
        needed
    unsigned short e_maxalloc;   // Maximum extra paragraphs
        needed
    unsigned short e_ss;    // Initial (relative) SS value
    unsigned short e_sp;    // Initial SP value
    unsigned short e_csum;   // Checksum
    unsigned short e_ip;     // Initial IP value
    unsigned short e_cs;    // Initial (relative) CS value
    unsigned short e_lfarlc;   // File address of relocation
        table
    unsigned short e_ovno;    // Overlay number
    unsigned short e_res[4];    // Reserved words
    unsigned short e_oemid;   // OEM identifier (for
       e_oeminfo)
    unsigned short e_oeminfo;  // OEM information; e_oemid
       specific
    unsigned short e_res2[10];    // Reserved words
    long   e_lfanew;    // File address of new exe header
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;


// Real mode stub program

typedef struct _IMAGE_FILE_HEADER {
    unsigned short  Machine;
    unsigned short  NumberOfSections;
    unsigned long   TimeDateStamp;
    unsigned long   PointerToSymbolTable;
    unsigned long   NumberOfSymbols;
```

```
    unsigned short  SizeOfOptionalHeader;
    unsigned short  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

typedef struct _IMAGE_OPTIONAL_HEADER {
    unsigned short  Magic;
    unsigned char   MajorLinkerVersion;
    unsigned char   MinorLinkerVersion;
    unsigned long   SizeOfCode;
    unsigned long   SizeOfInitializedData;
    unsigned long   SizeOfUninitializedData;
    unsigned long   AddressOfEntryPoint;   // offset of
       kernel_entry
    unsigned long   BaseOfCode;
    unsigned long   BaseOfData;
    unsigned long   ImageBase;   // Base address of
       kernel_entry
    unsigned long   SectionAlignment;
    unsigned long   FileAlignment;
    unsigned short  MajorOperatingSystemVersion;
    unsigned short  MinorOperatingSystemVersion;
    unsigned short  MajorImageVersion;
    unsigned short  MinorImageVersion;
    unsigned short  MajorSubsystemVersion;
    unsigned short  MinorSubsystemVersion;
    unsigned long   Reserved1;
    unsigned long   SizeOfImage;
    unsigned long   SizeOfHeaders;
    unsigned long   CheckSum;
    unsigned short  Subsystem;
    unsigned short  DllCharacteristics;
    unsigned long   SizeOfStackReserve;
    unsigned long   SizeOfStackCommit;
    unsigned long   SizeOfHeapReserve;
    unsigned long   SizeOfHeapCommit;
    unsigned long   LoaderFlags;
    unsigned long   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

ما نريد الحصول عليه هو عنوان الدالة الرئيسية للنواة (`kernel entry()`) والتي سيبدأ تنفيذ النواة

منها ، هذا العنوان موجود في أحد المتغيرات في آخر إضافة (header) وهي IMAGE OPTIONAL HEADER ، وحتى نحصل على عنوان هذه الأضافة يجب أن نبدأ من أول إضافة وذلك بسبب أن الاضافة الثانية ذات حجم متغير وليست ثابته مثل بقية الاضافات.

وبالنظر الى أول إضافة IMAGE DOS HEADER وبالتحديد الى المتغير lfanew e حيث يحوي عنوان الإضافة الثالثة IMAGE FILE HEADER والتي هي اضافة ثابته الحجم ، ومنها نصل الى آخر إضافة ونقرأ المتغير AddressOfEntryPoint الذي يحوي عنوان offset للدالة الرئيسية وكذلك نقرأ المتغير ImageBase والذي يحوي عنوان البداية للدالة ويجب اضافته لقيمة ال offset ، وبعد ذلك يتم نقل التنفيذ الى الدالة بواسطة الامر `call`. والشفرة التالية توضح طريقة ذلك (ويتم تنفيذها في المرحلة الثانية من محمل النظام مباشرة بعدما يتم تحميل النواة الى الذاكرة على العنوان KERNEL PMODE BASE).

Listing ٥٢.: Global new/delete operator

```
mov ebx,[KERNEL_PMODE_BASE+60]
add ebx,KERNEL_PMODE_BASE      ; ebx = _IMAGE_FILE_HEADER

add ebx,24       ; ebx = _IMAGE_OPTIONAL_HEADER

add ebx,16       ; ebx point to AddressOfEntryPoint

mov ebp,dword[ebx]  ; epb = AddressOfEntryPoint

add ebx,12       ; ebx point to ImageBase

add ebp,dword[ebx]  ; epb = kernel_entry

cli

call ebp
```

٢.٤.٥. تطوير بيئة التشغيل للغة سي++

حتى نتمكن من استخدام جميع خصائص لغة سي++ فانه يجب كتابة بعض الشفرات التشغيلية (startup) والتي تمهد وتعرف العديد من الخصائص في اللغة ، وفي هذا الجزء سيتم تطوير مكتبة وقت التشغيل للغة سي++ (C++ Runtime Library) وذلك نظراً لأننا قد الغينا الإعتماد على مكتبة وقت التشغيل التي تأتي مع المترجم المستخدم في بناء النظام (النظام الخاص بنا) حيث أن هذه المكتبة تعتمد على نظام التشغيل المستخدم في عملية التطوير مما يسبب مشاكل استدعاء دوال ليست موجودة.

وبدون تطوير هذه المكتبة فلن يمكن تهيئة الكائنات العامة (Global Object) و حذف الكائنات ، وكذلك لن يمكن استخدام بعض المعاملات (new,delete) و RTTI والاستثنائات (Exceptions).

المعاملات العامة Global Operator

سيتم تعريف معامل حجز الذاكرة (new) وتحريرها (delete) في لغة السي++ ، ولكن لاننا حاليا لم نبرمج مديراً للذاكرة فان التعريف سيكون خاليا.والمقطع التالي يوضح ذلك.

Listing ٥٣.: Global new/delete operator

```
void* __cdecl ::operator new (unsigned int size){return 0;}
void* __cdecl operator new[] (unsigned int size){return 0;}
void __cdecl ::operator delete (void * p){}
void __cdecl operator delete[] (void * p){}
```

Pure virtual function call handler

ايضا يجب تعريف دالة للتعامل مع الدوال الظاهرية النقية (Pure virtual function)[٥]، حيث سيقوم المترجم باستدعاء الدالة purecall()[٦] أينما وجد عملية استدعاء لدالة Pure virtual ، لذلك أن أردنا دعم الدوال Pure virtual يجب تعريف الدالة purecall ، وحاليا سيكون التعريف كالاتي.

Listing ٥٤.: Global new/delete operator

```
int __cdecl ::_purecall() { for (;;); return 0; }
```

دعم الفاصلة العائمة Floating Point Support

لدعم الفاصلة العائمة (Floating Point) في سي++ فانه يجب تعيين القيمة 1 للمتغير fltused ، وكذلك يجب تعريف الدالة ftol2_sse() والتي تحول من النوع float الى النوع long كالتالي.

[٥] عند تعريف دالة بأنها Pure virtual داخل أي فئة فإن هذا يدل على أن الفئة مجردة (Abstract) ويجب إعادة تعريف الدالة (Override) في الفئات المشتقة من الفئة التي تحوي هذه الدالة، والا ستكون الفئة المشتقة .

[٦] هذه الدالة يقوم مترجم الفيجوال سي++ باستدعائها.وقد تختلف من مترجم لآخر.

Listing ٥٥.: Global new/delete operator

```
extern "C" long __declspec (naked) _ftol2_sse() {
  int a;
#ifdef i386
  _asm {
    fistp [a]
    mov ebx, a
  }
#endif
}

extern "C" int _fltused = 1;
```

تهيئة الكائنات العامة والساكنة

عندما يجد المترجم كائنا فانه يضيف مهيئاً ديناميكيا له (Dynamic initializer) في قسم خاص من البرنامج[٧] وهو القسم crt.. وقبل أن يعمل البرنامج فان وظيفة مكتبة وقت التشغيل هي استدعاء وتنفيذ كل المهيئات وذلك حتى تأخذ الكائنات قيمها الابتدائية (عبر دالة البناء Constructor). وبسبب أننا أزلنا مكتبة وقت التشغيل فانه يجب انشاء القسم crt. وهذا يتم عن طريق موجهات المعالج التمهيدي(Preprocessor) الموجودة في المترجم.
هذا القسم crt. يحوي مصفوفة من مؤشرات الدوال (Function Pointer) ، ووظيفة مكتبة وقت التشغيل هي استدعاء كل الدوال الموجودة وذلك بالمرور على مصفوفة المؤشرات الموجودة . و يجب أن نعلم أن مصفوفة المؤشرات موجودة حقيقة داخل القسم crt:xcu. حيث أن الجزء الذي يلي العلامة dollar sign يحدد المكان بداخل القسم ، وحتى نتمكن من استدعاء وتنفيذ الدوال عن طريق مصفوفة المؤشرات فانه يجب انشاء مؤشر الى بداية القسم crt:xcu. وفي نهايته ، مؤشر البداية سيكون في القسم crt:xca. وهو يسبق القسم crt:xcu. مباشرة ، ومؤشر النهاية سيكون في القسم crt:xcz. ويلي القسم crt:xcu. مباشرة .
وبخصوص القسم crt. الذي سننشئه فاننا لا نملك صلاحيات قراءة وكتابة فيه ، لذلك الحل في أن نقوم بدمج هذا القسم مع قسم البيانات data. .
والشفرة التالية توضح ما سبق.

Listing ٥٦.: Global new/delete operator

[٧] في أي برنامج تنفيذي يوجد العديد من الأقسام، مثلا قسم البيانات data. وقسم الشفرة code. والمكدس stack. وغيرها.

```
// Function pointer typedef for less typing
typedef void (__cdecl *_PVFV)(void);

// __xc_a points to beginning of initializer table
#pragma data_seg(".CRT$XCA")
_PVFV __xc_a[] = { 0 };

// __xc_z points to end of initializer table
#pragma data_seg(".CRT$XCZ")
_PVFV __xc_z[] = { 0 };

// Select the default data segment again (.data) for the
   rest of the unit
#pragma data_seg()

// Now, move the CRT data into .data section so we can read/
   write to it
#pragma comment(linker, "/merge:.CRT=.data")


// initialize all global initializers (ctors, statics,
   globals, etc..)
void __cdecl _initterm ( _PVFV * pfbegin, _PVFV * pfend ) {

  //! Go through each initializer
    while ( pfbegin < pfend )
    {
    //! Execute the global initializer
      if ( *pfbegin != 0 )
            (**pfbegin) ();

      //! Go to next initializer inside the initializer
         table
        ++pfbegin;
    }
}

// execute all constructors and other dynamic initializers
void _cdecl init_ctor() {

   _atexit_init();
```

```
    _initterm(__xc_a, __xc_z);
}
```

حذف الكائنات

لكي يتم حذف الكائنات (Objects) يجب انشاء مصفوفة من مؤشرات دوال الهدم (deinitializer array) ، وذلك بسبب أن المترجم عندما يجد دالة هدم فانه يضيف مؤشراً الى دالة الهدم بداخل هذه المصفوفة وذلك حتى يتم استدعائها لاحقا (عند استدعاء الدالة exit())، ويجب تعريف الدالة atexit حيث أن مترجم الفيجوال سي++ يقوم باستدعائها عندما يجد أي كائن ، وظيفة هذه الدالة هي اضافة مؤشر لدالة هدم الكائن الى مصفوفة المؤشرات ،وبخصوص مصفوفة المؤشرات فانه يمكن حفظها في أي مكان على الذاكرة . والشفرة التالية توضح ما سبق.

Listing ٥٧.: Global new/delete operator

```
/! function pointer table to global deinitializer table
static _PVFV * pf_atexitlist = 0;

// Maximum entries allowed in table. Change as needed
static unsigned max_atexitlist_entries = 32;

// Current amount of entries in table
static unsigned cur_atexitlist_entries = 0;

//! initialize the de—initializer function table
void __cdecl _atexit_init(void) {

    max_atexitlist_entries = 32;

  // Warning: Normally, the STDC will dynamically allocate
     this. Because we have no memory manager, just choose
  // a base address that you will never use for now
  pf_atexitlist = (_PVFV *)0x5000;
}

//! adds a new function entry that is to be called at
   shutdown
int __cdecl atexit(_PVFV fn) {
```

```
  //! Insure we have enough free space
  if (cur_atexitlist_entries>=max_atexitlist_entries)
    return 1;
  else {

    //! Add the exit routine
    *(pf_atexitlist++) = fn;
    cur_atexitlist_entries++;
  }
  return 0;
}

//! shutdown the C++ runtime; calls all global de-
   initializers
void _cdecl exit () {

  //! Go through the list, and execute all global exit
     routines
  while (cur_atexitlist_entries--) {

      //! execute function
      (*(--pf_atexitlist)) ();
  }
}
```

٣.٤.٥. نقل التنفيذ الى النواة

بعد أن قمنا بعمل تحليل (Parsing) لصيغة ملف PE ونقل التنفيذ الى الدالة `kernel_entry()` والتي تعتبر أول دالة يتم تنفيذها في نواة النظام ، وأول ما يجب تنفيذه فيها هو تحديد قيم مسجلات المقاطع وانشاء مكدس (Stack) وبعد ذلك يجب تهئية الكائنات العامة ومن ثم استدعاء الدالة `main()` التي تحوي شفرة النواة ، واخيرا عندما تعود الدالة `main()` يتم حذف الكائنات وايقاف النظام (Hang). والشفرة التالية توضح ذلك

Listing ٥٨.: Global new/delete operator

```
extern void _cdecl main ();   // main function.
extern void _cdecl init_ctor();   // init constructor.
extern void _cdecl exit ();     // exit.
```

```
void _cdecl kernel_entry ()
{

#ifdef i386
  _asm {
    cli

    mov ax, 10h       // select data descriptor in GDT.
    mov ds, ax
    mov es, ax
    mov fs, ax
    mov gs, ax
    mov ss, ax        // Set up base stack
    mov esp, 0x90000
    mov ebp, esp      // store current stack pointer
    push ebp
  }
#endif

  // Execute global constructors
  init_ctor();

  // Call kernel entry point
  main();

  // Cleanup all dynamic dtors
  exit();

#ifdef i386
  _asm cli
#endif

  for(;;);
}
```

وتعريف الدالة main() حالياً سيكون خاليا.

٥.٥. نظرة على شفرة نظام إقرأ

أهم الخصائص التي يجب مراعتها أثناء برمجة نواة نظام التشغيل هي خاصية المحمولية على صعيد الأجهزة والمنصات[٨] وخاصية قابلية توسعة النواة (Expandibility) و لذلك تم الإتفاق على أن تصميم نواة نظام تشغيل إقرأ سيتم بنائها على طبقة HAL حتى تسمح لأي مطور فيما بعد إعادة تطبيق هذه الطبقة لدعم أجهزة وعتاد آخر. وحتى نحصل على أعلى قدر من المحمولية وقابلية التوسعة في نواة النظام فانه سيتم تقسيم الشفرات البرمجية للنواة الى وحدات مستقلة بحيث تؤدي كل وحدة وظيفة ما ، وفي نفس الوقت يجب أن تتوافر واجهة عامة (Interface) لكل وحدة بحيث نتمكن من الاستفادة من خدمات هذه الوحدة دون الحاجة لمعرفة تفاصيلها الداخلية. وفي بداية تصميم المشروع فان عملية تصميم الواجهة تعتبر أهم بكثير من عملية برمجة محتويات الوحدة أو ما يسمى بالتنفيذ (Impelmentation) نظراً لان التنفيذ قد لا يؤثر على هيكلة المشروع ومعماريته مثلما تؤثر الواجهة .

- eqraOS:
  - boot: first-stage and second-stage bootloader.
  - core:
    - kernel:Kernel program PE executable file type.
    - hal:Hardware abstraction layer.
    - lib:Standard library runtime and standard C/C++ library.
    - include:Standard include headers.
    - debug:Debug version of eqraOS.
    - release:Final release of eqraOS.

٦.٥. مكتبة السي القياسية

نظراّ لأنه قد تم إلغاء الاعتماد على مكتبة السي والسي++ القياسية أثناء تطوير نواة نظام التشغيل فانه يجب انشاء هذه المكتبة حتى نتمكن من استخدام لغة سي وسي++ ، وبسبب أن عملية إعادة برمجة هذه المكتبات يتطلب وقتاّ وجهدا فاننا سنركز على بعض الملفات المستخدمة بكثرة ونترك البقية للتطوير لاحقا.

[٨] على عكس محمل النظام Bootloader والذي يعتمد على معمارية العتاد والمعالج.

تعريف NULL

في لغة سي++ يتم تعريف NULL على أنها القيمة 0 بينما في لغة السي تعرف ب 0 (*void).

Listing ٥٩.: null.h:Definition of NULL in C and C++

```
#ifndef NULL_H
#define NULL_H

#if define (_MSC_VER) && (_MSC_VER > = 1020)
#pargma once
#endif

#ifdef NULL
#undev NULL
#endif

#ifdef __cplusplus
extern "C"
{
#endif

/* C++ NULL definition */
#define NULL 0

#ifdef __cplusplus
}
#else

/* C NULL definition */
#define NULL (void*)0

#endif

#endif //NULL_H
```

وعند ترجمة النواة بمترجم سي++ فان القيمة __cplusplus تكون معرَّفة لديه ، أما في حالة ترجمة النواة بمترجم سي فان المترجم لا يُعرِّف تلك القيمة.

تعريف size_t

يتم تعريف size_t على أنها عدد صحيح 32-bit بدون إشارة (unsigned).

Listing ٥١٠.: size_t.h:Definition of size_t in C/C++

```
#ifndef SIZE_T_H
#define SIZE_T_H

#ifdef __cplusplus
extern "C"
{
#endif

/* Stdandard definition of size_t */
typedef unsigned size_t;

#ifdef __cplusplus
}
#endif


#endif //SIZE_T_H
```

إعادة تعريف أنواع البيانات

أنواع البيانات (Data Types) تختلف حجمها بحسب المترجم والنظام الذي تم ترجمة البرنامج عليه ، ويفضل أن يتم اعادة تعريفها (typedef) لتوضيح الحجم والنوع في آن واحد .

Listing ٥١١.: stdint.h:typedef data type

```
#ifndef STDINT_H
#define STDINT_H`

#define __need_wint_t
#define __need_wchar_t

/* Exact-width integer type */
```

```
typedef   char          int8_t;
typedef   unsigned char      uint8_t;
typedef   short         int16_t;
typedef   unsigned short     uint16_t;
typedef   int           int32_t;
typedef   unsigned int     uint32_t;
typedef   long long      int64_t;
typedef   unsigned long long  uint64_t;


// to be continue..

#endif //STDINT_H
```

ولدعم ملفات الرأس للغة سي++ فان الملف السابق سيتم تضمينه في ملف cstdint وهي التسمية التي تتبعها السي++ في ملفات الرأس[٩].

Listing ٥١٢.: cstdint:C++ typedef data type

```
#ifndef CSTDINT_H
#define CSTDINT_H

#include <stdint.h>

#endif //CSTDINT_H
```

نوع الحرف

ملف ctype.h يحوي العديد من الماكرو (Macros) والتي تحدد نوع الحرف (عدد،حرف،حرف صغير،مسافة،حرف تحكم،...الخ).

Listing ٥١٣.: ctype.h:determine character type

```
#ifndef CTYPE_H
#define CTYPE_H
```

[٩]ملفات الرأس للغة سي++ تتبع نفس هذا الأسلوب لذلك لن يتم ذكرها مجددا وسنكتفي بذكر ملفات الرأس للغة سي.

```
#ifdef _MSC_VER
#pragma warning (disable:4244)
#endif

#ifdef __cplusplus
extern "C"
{
#endif

extern char _ctype[];

/* constants */

#define CT_UP     0x01  // upper case
#define CT_LOW    0x02  // lower case
#define CT_DIG    0x04  // digit
#define CT_CTL    0x08  // control
#define CT_PUN    0x10  // punctuation
#define CT_WHT    0x20  // white space (space,cr,lf,tab).
#define CT_HEX    0x40  // hex digit
#define CT_SP   0x80  // sapce.

/* macros */

#define isalnum(c)    ( (_ctype+1)[(unsigned)(c)] & (CT_UP|
   CT_LOW|CT_DIG) )
#define isalpha(c)    ((_ctype + 1)[(unsigned)(c)] & (CT_UP
   | CT_LOW))
#define iscntrl(c)      ((_ctype + 1)[(unsigned)(c)] & (
   CT_CTL))


// to be continue..

#ifdef __cplusplus
}
#endif

#endif // CTYPE_H
```

دعم الدوال بعدد غير محدود من الوسائط

٧.٥. دالة طباعة المخرجات للنواة

٦. المقاطعات Interrupts

المقاطعات هي طريقة لإيقاف المعالج بشكل مؤقت من تنفيذ عملية ما (Current Process) والبدء بتنفيذ أوامر أخرى . وكمثال على ذلك هو عند الضغط على أي حرف في لوحة المفاتيح فان هذا يولد مقاطعة (Interrupt) تأتي كإشارة الى المعالج بأن يوقف ما يعمل عليه حاليا ويحفظ كل القيم التي يحتاجها لكي يستطيع مواصلة ما تم قطعه ، وفي حالة وجود دالة للتعامل مع هذه المقاطعة (مقاطعة لوحة المفاتيح) وتسمى دالة معالجة المقاطعة (Interrupt Handler) أو دالة خدمة المقاطعة (Interrupt Service Rountine) فان التنفيذ يتنقل اليها تلقائيا ، و يتم فيها معالجة هذه المقاطعة (مثلا يتم قراءة الحرف الذي تم ادخاله من متحكم لوحة المفاتيح ومن ثم ارساله الى متغير في الذاكرة) وعندما تنتهي دالة معالجة المقاطعة من عملها فان المعالج يعود لِيُكْمِل تنفيذ العملية التي كان يعمل عليها. والمقاطعات إما تكون مقاطعات عتادية (Hardware Interrupt) وتصدر من عتاد الحاسب أو تكون برمجية (Software Interrupt) وتصدر من خلال البرامج عن طريق تعليمة `int n`. كذلك هناك مقاطعات يصدرها المعالج نفسه عند حدوث خطأ ما (مثلا عن القسمة على العدد صفر أو عند حدوث Page Fault) وتسمى هذه المقاطعات بأخطاء المعالج أو استثنائات المعالج (Exceptions) ويجب معالجة هذه الأخطاء (Error Handler) لأنها توقف عمل النظام في حالة لم تتوفر دالة لمعالجتها.

١.٦. المقاطعات البرمجية Software Interrupts

المقاطعات البرمجية هي مقاطعات يتم اطلاقها من داخل البرنامج (عن طريق الأمر `int n`) لِنقل التنفيذ الى دالة أخرى تعالج هذه المقاطعة (Interrupt handler)، وغالبا ما تستخدم هذه المقاطعات في برامج المستخدم (Ring3 user mode) للاستفادة من خدمات النظام (مثلا للقراءة والكتابة في أجهزة الإدخال والإخراج حيث لا توجد طريقة اخرى لذلك في نمط المستخدم).

١.١.٦. المقاطعات في النمط الحقيقي

في النمط الحقيقي عندما يتم تنفيذ أمر المقاطعة (وهو ما يسمى بطلب تنفيذ المقاطعة (Interrupt Request) وتختصر بـــIRQ) فان المعالج يأخذ رقم المقاطعة المطلوب تنفيذها ويذهب بها الى جدول المقاطعات (Interrupt Vector Table) ، هذا الجدول يبدأ من العنوان الحقيقي `0x0` وينتهي عند العنوان `0x3ff`

ويحوي كل سجل فيه على عنوان دالة معالجة المقاطعة (IR) والتي يجب تنفيذها لتخديم المقاطعة المطلوبة. حجم العنوان هو أربع بايت وتكون كالتالي:

- Byte 0: Low offset address of IR.
- Byte 1: High offset address of IR.
- Byte 2: Low Segment address of IR.
- Byte 3: High Segment Address of IR.

ويتكون الجدول من 256 مقاطعة (وبحسبة بسيطة يكون حجم الجدول هو 1024 بايت وهي ناتجة من ضرب عدد المقاطعات في حجم كل سجل)، بعض منها محجوز والبعض الاخر يستخدمه المعالج والبقية متروكة لمبرمج نظام التشغيل لدعم المزيد من المقطاعات. وبسبب أن الجدول يتكون فقط من عناوين لدوال معالجة المقاطعات فان هذا يمكننا من وضع الدالة في أي مكان على الذاكرة ومن ثم وضع عنوانها داخل هذا السجل (يتم هذا عن طريق مقاطعات البايوس)، والجدول التالي يوضح IVT والمقاطعات الموجودة فيه.

Base Address	Interrupt Number	Description
0x000	0	Divide by 0
0x004	1	Single step (Debugger)
0x008	2	Non Maskable Interrupt (NMI) Pin
0x00C	3	Breakpoint (Debugger)
0x010	4	Overflow
0x014	5	Bounds check
0x018	6	Undefined Operation Code
0x01C	7	No coprocessor
0x020	8	Double Fault
0x024	9	Coprocessor Segment Overrun
0x028	10	Invalid Task State Segment (TSS)
0x02C	11	Segment Not Present
0x030	12	Stack Segment Overrun
0x034	13	General Protection Fault (GPF)
0x038	14	Page Fault
0x03C	15	Unassigned
0x040	16	Coprocessor error
0x044	17	Alignment Check (486+ Only)
0x048	18	Machine Check (Pentium/586+ Only)
0x05C	19-31	Reserved exceptions
0x068 - 0x3FF	32-255	Interrupts free for software use

٢.١.٦. المقاطعات في النمط المحمي

في النمط المحمي يستخدم المعالج جدولاً خاصاً يسمى بجدول واصفات المقاطعات (Interrupt Descriptor Table) ويختصر ب IDT ، هذا الجدول يشابه جدول IVT حيث يتكون من 256 واصفة كل واصفة مخصصة لمقاطعة ما (اذاً الجدول يحوي 256 مقاطعة) ، حجم كل واصفة هو 8 بايت تحوي عنوان دالة معالجة المقاطعة (IR) و نوع الناخب (selector type: code or data) في جدول GDT الذي تعمل عليه دالة معالجة المقاطعة ، بالاضافة الى مستوى الحماية المطلوب والعديد من الخصائص توضحها التركيبة التالية.

- Bits 0-15:
  - Interrupt / Trap Gate: Offset address Bits 0-15 of IR
  - Task Gate: Not used.
- Bits 16-31:
  - Interrupt / Trap Gate: Segment Selector (Useually 0x10)
  - Task Gate: TSS Selector
- Bits 31-35: Not used
- Bits 36-38:
  - Interrupt / Trap Gate: Reserved. Must be 0.
  - Task Gate: Not used.
- Bits 39-41:
  - Interrupt Gate: Of the format 0D110, where D determins size
    - 01110 - 32 bit descriptor
    - 00110 - 16 bit descriptor
  - Task Gate: Must be 00101
  - Trap Gate: Of the format 0D111, where D determins size
    - 01111 - 32 bit descriptor
    - 00111 - 16 bit descriptor
- Bits 42-44: Descriptor Privedlge Level (DPL)
  - 00: Ring 0
  - 01: Ring 1
  - 10: Ring 2
  - 11: Ring 3

- Bit 45: Segment is present (1: Present, 0:Not present)
- Bits 46-62:
  - Interrupt / Trap Gate: Bits 16-31 of IR address
  - Task Gate: Not used

والمثال التالي يوضح انشاء واصفة واحدة بلغة التجميع حتى يسهل تتبع القيم ، وسيتم كتابة مثال كامل لاحقا بلغة السي.

Listing ٦١.: Example of interrupt descriptor

```
idt_descriptor:
   baseLow       dw    0x0
   selector        dw    0x8
   reserved        db    0x0
   flags         db    0x8e            ; 010001110
   baseHi        dw    0x0
```

المتغير الأول `baseLow` هو أول 16 بت من عنوان دالة معالجة المقاطعة IR ويكمل الجزء الاخر من العنوان المتغير `baseHi` وفي هذا المثال العنوان هو `0x0` بمعنى أن دالة تخديم المقاطعة ستكون في العنوان `0x0`. وبما أن دالة معالجة (تخديم) المقاطعة تحوي شفرة برمجية للتنفيذ وليست بيانات (Data) فان قيمة المتغير `selector` يجب أن تكون `0x8` للإشارة الى ناخب الشفرة (Code Selector) في جدول الواصفات العام (GDT). أما المتغير `flags` فان قيمته هي `010001110b` دلالة على أن الواصفة هي `32-bit` وأن مستوى الحماية هو الحلقة صفر (Ring0).
وبعد أن يتم انشاء أغلب الواصفات بشكل متسلسل (في أي مكان على الذاكرة) ، يجب أن ننشئ جدول IDT وهذا يتم عن طريق حفظ عنوان أول واصفة في متغير وليكن `idt_start` وعنوان نهاية الواصفات في المتغير `idt_end` ومن ثم انشاء مؤشراً يسمى `idt_ptr` والذي يجب أن يكون في صورة معينة بحيث يحفظ عنوان بداية الجدول ونهايته :

Listing ٦٢.: Value to put in IDTR

```
idt_ptr:
  limit dw idt_end - idt_start  ; bits 0-15 is size of idt
  base  dd idt_start    ; base of idt
```

هذا المؤشر يجب أن يتم تحميله الى المسجل `IDTR` (وهو مسجل داخل المعالج) عن طريق تنفيذ الامر `lidt`[1] بالشكل التالي `lidt [idt_ptr]`.

[1] بعد تنفيذ هذا الأمر فان جدول المقاطعات سيتم استبداله بالجدول الجديد والذي نجد عنوانه بداخل المسجل `idtr` ، وهذا الأمر لا يُنفَّذ إلاّ اذا كانت قيمة العلم (CPL flag) هي صفر.

وعند حدوث أي مقاطعة فان المعالج ينهي الأمر الذي يعمل عليه و يأخذ رقم المقاطعة ويذهب به الى جدول IDT (عنوان هذا الجدول يتواجد بداخل المسجل IDTR) ، وبعد ذلك يقوم بحساب مكان الواصفة بالمعادلة `int_num * 8` وذلك بسبب أن حجم كل واصفة في جدول IDT هو 8 بايت. وقبل أن ينقل التنفيذ الى دالة معالجة المقاطعة فانه يجب أن يقوم بعملية حفظ للمكان الذي توقف فيه حتى يستطيع أن يتابع عمله عندما تعود دالة معالجة المقاطعة . ويتم حفظ الأعلام `EFLAGS` ومسجل مقطع الشفرة `CS` ومسجل عنوان التعليمة التالية `IP` في المكدس (Stack) الحالي ، وفي حالة حدوث خطأ ما فانه يتم دفع شفرة الخطأ (Error Code) الى المكدس أيضا. وشفرة الخطأ هي بطول `32-bit` وتتبع التركيبة التالية.

- Bit 0: External event
  - 0: Internal or software event triggered the error.
  - 1: External or hardware event triggered the error.
- Bit 1: Description location
  - 0: Index portion of error code refers to descriptor in GDT or current LDT.
  - 1: Index portion of error code refers to gate descriptor in IDT.
- Bit 2: GDT/LDT. Only use if the descriptor location is 0.
  - 0: This indicates the index portion of the error code refers to a descriptor in the current GDT.
  - 1: This indicates the index portion of the error code refers to a segment or gate descriptor in the LDT.
- Bits 3-15: Segment selector index. This is an index into the IDT, GDT, or current LDT to the segment or gate selector bring refrenced by the error code.
- Bits 16-31: Reserved.

وعندما تنتهي دالة معالجة المقاطعة من عملها فانه يجب أن تنفذ الأمر `iret` أو `iretd` حتى يتم ارجاع القيم التي تم دفعها الى المكدس (قيم الأعلام FLAGS). وبالتالي يُكْمِل المعالج عمله.

٣.١.٦. أخطاء المعالج

خلال تنفيذ المعالج للأوامر فانه ربما يحدث خطأ ما مما يجعل المعالج يقوم بتوليد استثناء يعرف باستثناء المعالج ، ويوجد له عدة أنواع:

- الخطأ Fault: عندما تعمل دالة معالجة هذا النوع من الاستثناء فربما يتم اصلاح هذا الخطأ ، وعنوان العودة الذي يتم دفعه الى المكدس هو عنوان الأمر الذي تسبب في هذا الخطأ.
- الخطأ Trap: عنوان العودة هو عنوان التعليمة التي تلي الأمر الذي تسبب في الخطأ.
- الخطأ Abort: لا يوجد عنوان للعودة ، ولن يكمل البرنامج عمله بعد انتهاء دالة معالجة الخطأ.

والجدول التالي يوضح أخطاء المعالج والمقاطعات التي يقوم بتوليدها.

Interrupt Number	Class	Description
0	Fault	Divide by 0
1	Trap/Fault	Single step
2	Unclassed	Non Maskable Interrupt (NMI) Pin
3	Trap	Breakpoint
4	Trap	Overflow
5	Fault	Bounds check
6	Fault	Unvalid OPCode
7	Fault	Device not available
8	Abort	Double Fault
9	Abort	Coprocessor Segment Overrun
10	Fault	Invalid Task State Segment
11	Fault	Segment Not Present
12	Fault	Stack Fault Exception
13	Fault	General Protection Fault
14	Fault	Page Fault
15	-	Unassigned
16	Fault	x87 FPU Error
17	Fault	Alignment Check
18	Abort	Machine Check
19	Fault	SIMD FPU Exception
20-31	-	Reserved
32-255	-	Avilable for software use

ويجدر بنا الوقوف على ملاحظة كنّا قد ذكرناها في الفصول السابقة وهي إلغاء المقاطعات (بواسطة الأمر `cli`) عند الانتقال الى النمط المحمي حتى لا يتسبب في حدوث خطأ General Protection Fault وبالتالي توقف النظام عن العمل وسبب ذلك هو أن عدم تنفيذ الأمر `cli` يعني أن المقاطعات العتادية مفعلة وبالتالي أي عتاد يمكنه أن يرسل مقاطعة الى المعالج لكي ينقل التنفيذ الى دالة تخديمها . وعند بداية الانتقال الى النمط المحمي فان جدول المقاطعات IDT لم يتم انشائه وأي محاولة لاستخدامه سيؤدي الى هذا الخطأ.
أحد المتحكمات التي ترسل مقاطعات الى المعالج بشكل ثابت هو متحكم Prpgrammable Interval Timer وتختصر بمتحكم PIT وهي تمثل ساعة النظام System Timer بحيث ترسل مقاطعة بشكل دائم الى المعالج والذي بدوره ينقل التنفيذ الى دالة تخديم هذه المقاطعة . وبسبب أن جدول المقطاعات غير متواجد

في بداية المرحلة الثانية من محمل النظام وكذلك لا توجد دالة لتخديم هذه المقاطعة فان هذا يؤدي الى توقف النظام ، لذلك يجب ايقاف المقاطعات العتادية لحين انشاء جدول المقطاعات وكتابة دوال معالجة المقاطعات. كذلك توجد مشكلة أخرى لبعض المقاطعات العتادية حيث انها تستخدم نفس أرقام المقاطعات التي يستخدمها المعالج للإستثناءات وحلها هو بإعادة برمجة الشريحة المسؤولة عن استقبال الاشارات من العتاد وتحويلها الى مقاطعات وارسالها الى المعالج ، هذه الشريحة تسمى Programmable Interrupt Controller وتختصر ب PIC ويجب إعادة برمجتها وتغيير ارقام المقاطعات للأجهزة التي تستخدم أرقاماً متشابهة.
وفيما يلي سيتم إنشاء جدول المقاطعات (IDT) باستخدام لغة السي وتوفير ال 256 دالة لمعالجة المقطاعات وحاليا سيقتصر عمل الدوال على طباعة رسالة ، وقبل ذلك سنقوم بانشاء جدول الواصفات العام (GDT) مجددا (أي سيتم الغاء الجدول الذي قمنا بانشائه في مرحلة الاقلاع) وبعد ذلك سنبدأ في برمجة متحكم PIC واعادة ترقيم مقاطعات الأجهزة وكذلك برمجة ساعة النظام لارسال مقاطعة بوقت محدد.

٤.١.٦. إنشاء جدول الواصفات العام GDT

الهدف الرئيسي في نواة نظام التشغيل هي المحمولية على صعيد المنصات ، وهذا ما أدى الى اعتماد فكرة طبقة HAL والتي يقبع تحتها كل ما يتعلق بعتاد الحاسب وادارته وكل ما يجعل النظام معتمداً على معمارية معينة أيضا نجده تحت طبقة HAL ، وجدول الواصفات العام – كما ذكرنا في الفصول السابقة– يحدد ويقسم لنا الذاكرة الرئيسية كأجزاء قابلة للتنفيذ وأجزاء تحوي بيانات وغيرها ، ونظراً لأن إنشاء هذا الجدول يعتمد على معمارية المعالج والأوامر المدعومة فيه فانه يجب ان يقع تحت طبقة HAL[٢] وهذا يعني أن نقل النظام الى معمارية حاسوب آخر يتطلب فقط إعادة برمجة طبقة HAL .
بداية سنبدأ بتصميم الواجهة العامة لطبقة HAL ويجب أن نراعي أن تكون الواجهة مفصولة تماما عن التطبيق حتى يتمكن أي مطور من إعادة تطبيقها لاحقاً على معمارية حاسوب آخر.

Listing ٦٣.: include/hal.h:Hardware Abstraction Layer Interface

```
#ifndef HAL_H
#define HAL_H

#ifndef i386
#error "HAL is not implemented in this platform"
#endif

#include <stdint.h>

#ifdef _MSC_VER
```

[٢] من منظور آخر هذه الجداول (GDT,LDT and IDT) هي جداول للمعالج لذلك يجب أن تكون في طبقة HAL.

```
#define interrupt __declspec(naked)
#else
#define interrupt
#endif

#define far
#define near


/*  Interface */

extern int _cdecl hal_init();
extern int _cdecl hal_close();
extern void _cdecl gen_interrupt(int);


#endif // HAL_H
```

وحالياً واجهة طبقة HAL مكونة من ثلاث دوال تم الإعلان عنها بأنها `extern` وهذا يعني أن أي تطبيق (Implementation) لهذه الواجهة يجب أن يُعرِّف هذه الدوال. الدالة الاولى هي `hal_init()` والتي تقوم بتهيئة العتاد وجداول المعالج بينما الدالة الثانية `hal_close()` تقوم بعملية الحذف والتحرير وأخيرا الدالة `gen_interrupt` والتي تم وضعها لغرض تجربة إرسال مقاطعة برمجية والتأكد من أن دالة معالجة المقاطعة تعمل كما يرام.
نعود بالحديث الى جدول الواصفات العام (GDT) [٣] حيث سيتم انشائه بلغة السي وهذا ما سيسمح لنا باستخدام تراكيب عالية للتعبير عن الجدول و المؤشر مما يعطي وضوح ومقروئية أكثر في الشفرة.وسوف نحتاج الى تعريف ثلاث دوال[٤]:

- الدالة `i386_gdt_init`: تقوم بتهيئة واصفة خالية وواصفة للشفرة وللبيانات وكذلك انشاء مؤشر الجدول.
- الدالة `i386_gdt_set_desc`: دالة تهيئة الواصفة حيث تستقبل القيم وتعينها الى الواصفة المطلوبة.
- الدالة `gdt_install`: تقوم بتحميل المؤشر الذي يحوي حجم الجدول وعنوان بدايته الى المسجل GDTR.

والشفرة التالية توضح كيفية انشاء الجدول[٥].

[٣] راجع ١.١.٤.
[٤] لغرض التنظيم والتقسيم لا أكثر ولا أقل.
[٥] راجع شفرة النظام لقراءة ملف الرأس `hal/gdt.h`.

Listing ٦٤.: hal/gdt.cpp:Install GDT

```
#include <string.h>
#include "gdt.h"

static struct gdt_desc _gdt[MAX_GDT_DESC];
static struct gdtr _gdtr;


static void gdt_install();


static void gdt_install() {
#ifdef _MSC_VER
  _asm lgdt [_gdtr];
#endif
}

extern void i386_gdt_set_desc(uint32_t index,uint64_t base,
    uint64_t limit,uint8_t access,uint8_t grand) {

  if ( index > MAX_GDT_DESC )
    return;

  // clear the desc.
  memset((void*)&_gdt[index],0,sizeof(struct gdt_desc));

  // set limit and base.
  _gdt[index].low_base = uint16_t(base & 0xffff);
  _gdt[index].mid_base = uint8_t((base >> 16) & 0xff);
  _gdt[index].high_base = uint8_t((base >> 24) & 0xff);
  _gdt[index].limit = uint16_t(limit & 0xffff);

  // set flags and grandularity bytes
  _gdt[index].flags = access;
  _gdt[index].grand = uint8_t((limit >> 16) & 0x0f);
  _gdt[index].grand = _gdt[index].grand | grand & 0xf0;
}

extern gdt_desc* i386_get_gdt_desc(uint32_t index) {
  if ( index >= MAX_GDT_DESC )
```

```
    return 0;
  else
    return &_gdt[index];
}

extern int i386_gdt_init() {

  // init _gdtr
  _gdtr.limit = sizeof(struct gdt_desc) * MAX_GDT_DESC - 1;
  _gdtr.base = (uint32_t)&_gdt[0];

  // set null desc.
  i386_gdt_set_desc(0,0,0,0,0);

  // set code desc.
  i386_gdt_set_desc(1,0,0xffffffff,
    I386_GDT_CODE_DESC|I386_GDT_DATA_DESC|I386_GDT_READWRITE
       |I386_GDT_MEMORY,   // 10011010
    I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K
           // 11001111

  );

  // set data desc.
  i386_gdt_set_desc(2,0,0xffffffff,
    I386_GDT_DATA_DESC|I386_GDT_READWRITE|I386_GDT_MEMORY,
       // 10010010
    I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K      //
       11001111
  );

  // install gdtr
  gdt_install();

  return 0;
}
```

٥.١.٦. إنشاء جدول المقاطعات IDT

٢.٦. Programmable Interrupt Controller

٣.٦. Programmable Interval Timer

٤.٦. المقاطعات العتادية Hardware Interrupts

٧. إدارة الذاكرة

١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management

٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management

٨. مشغلات الاجهزة Device Driver

١.٨. برمجة مشغل لوحة المفاتيح Keyboard Driver

٢.٨. برمجة مشغل القرص المرن Floppy Disk Driver

انظر الى شفرة النظام.

٣.٨. برمجة متحكم DMAC

انظر الى شفرة النظام.

٩. أنظمة الملفات

انظر الى شفرة النظام.

١. ترجمة وتشغيل البرامج

لتطوير نظام التشغيل يجب استخدام مجموعة من الادوات واللغات التي تساعد وتيسير عملية التطوير وفي هذا الفصل سيتم عرض هذه الأدوات وكيفية استخدامها.

اعداد مترجم فيجوال سي++ لبرمجة النواة.

١.١. نظام ويندوز

١.٢. نظام لينوكس

Listing ١١.: Some Code

```
The following is a list of the standard BIOS interrupts
   used in a typical BIOS.

%http://www.bioscentral.com/misc/biosservices.htm
Interrupt   Address   Type    Description
00h       0000:0000h  Processor   Divide by zero
01h       0000:0004h  Processor   Single step
02h       0000:0008h  Processor   Non maskable interrupt (
   NMI)
03h       0000:000Ch  Processor   Breakpoint
04h       0000:0010h  Processor   Arithmetic overflow
05h       0000:0014h  Software  Print screen
06h       0000:0018h  Processor   Invalid op code
07h       0000:001Ch  Processor   Coprocessor not available
08h       0000:0020h  Hardware  System timer service
   routine
09h       0000:0024h  Hardware  Keyboard device service
   routine
```

```
0Ah       0000:0028h  Hardware  Cascade from 2nd
   programmable interrupt controller
0Bh       0000:002Ch  Hardware  Serial port service – COM
   post 2
0Ch       0000:0030h  Hardware  Serial port service – COM
   port 1
0Dh       0000:0034h  Hardware  Parallel printer service –
   LPT 2
0Eh       0000:0038h  Hardware  Floppy disk service
0Fh       0000:003Ch  Hardware  Parallel printer service –
   LPT 1
10h       0000:0040h  Software  Video service routine
11h       0000:0044h  Software  Equipment list service
   routine
12h       0000:0048H  Software  Memory size service routine
13h       0000:004Ch  Software  Hard disk drive service
14h       0000:0050h  Software  Serial communications
   service routines
15h       0000:0054h  Software  System services support
   routines
16h       0000:0058h  Software  Keyboard support service
   routines
17h       0000:005Ch  Software  Parallel printer support
   services
18h       0000:0060h  Software  Load and run ROM BASIC
19h       0000:0064h  Software  DOS loading routine
1Ah       0000:0068h  Software  Real time clock service
   routines
1Bh       0000:006Ch  Software  CRTL – BREAK service
   routines
1Ch       0000:0070h  Software  User timer service routine
1Dh       00000074h   Software  Video control parameter
   table
1Eh       0000:0078h  Software  Floppy disk parameter
   routine
1Fh       0000:007Ch  Software  Video graphics character
   routine
20h–3Fh     0000:0080f – 0000:00FCh   SOftware  DOS
   interrupt points
40h       0000:0100h  Software  Floppy disk revector
   routine
41h       0000:0104h  Software  hard disk drive C:
```

```
   parameter table
42h       0000:0108h  Software  EGA default video driver
43h       0000:010Ch  Software  Video graphics characters
44h       0000:0110h  Software  Novel Netware API
45h       0000:0114h  Software  Not used
46h       0000:0118h  Software  Hard disk drive D:
   parameter table
47h       0000:011Ch –  Software  Not used
48h         Software  Not used
49h       0000:0124h  Software  Not used
4Ah       0000:0128h  Software  User alarm
4Bh–63h     0000:012Ch –  Software  Not used
64h         Software  Novel Netware IPX
65h–66h       Software  Not used
67h         Software  EMS support routines
68h–6Fh     0000:01BCh  Software  Not used
70h       0000:01c0h  Hardware  Real time clock
71h       0000:01C4h  Hardware  Redirect interrupt cascade
72h–74h     0000:01C8h – 0000:01D0h   Hardware  Reserved –
   Do not use
75h       0000:01D4h  Hardware  Math coprocessor exception
76h       0000:01D8h  Hardware  Hard disk support
77h       0000:01DCh  Hardware  Suspend request
78h–79h     0000:01E0h –  Hardware  Not used
7Ah         Software  Novell Netware API
78h–FFh     0000:03FCh  Software  Not used
```

ب. المراجع

ب.١. المراجع

Bibliography

[1] William Stallings, *Operating System: Internals and Design Principles*. Prentice Hall, 5th Edition, 2004.

[2] Andrew S. Tanenbaum ,Albert S Woodhull, *Operating Systems Design and Implementation*. Prentice Hall, 3rd Edition, 2006.

[3] Michael Tischer, Bruno Jennrich, *PC Intern: The Encyclopedia of System Programming*. Abacus Software, 6th Edition, 1996.

[4] Hans-Peter Messmer, *The Indispensable PC Hardware Book*. Addison-Wesley Professional, 4th Edition, 2001.

[5] Andrew S. Tanenbaum, *Structured Computer Organization*. Prentice Hall, 4th Edition, 1998.

[6] Ytha Yu,Charles Marut, *Asssembly Language Programming and Organization IBM PC*. McGraw-Hill/Irwin, 1st Edition, 1992.

[7] Intel® Manuals, *Intel® 64 and IA-32 Architectures Software Developer's Manuals*. http://www.intel.com/products/processor/manuals/

[8] OSDev: http://wiki.osdev.org

[9] brokenthorn: http://brokenthorn.com

[10] Computer Sciense Community in Sudan: http://sudancs.com

ج. شفرة نظام إقرأ

كود النظام

د. إتفاقية ترخيص المستندات الحرة GNU FDL