١ إدارة الذاكرة

تعتبر ذاكرة الحاسب الرئيسية (RAM) من أهم الموارد التي يجب على نظام التشغيل إدارتها حيث يمثل المخزن الذي يقرأ المعالج منه الأوامر ويقوم بتنفيذها ، ولا يمكن للمعالج الوصول المباشر الى أحد الذواكر الثانوية مباشرة وإنما يتم تحميل البيانات والبرامج الى الذاكرة الرئيسية حتى تصبح متاحة للمعالج. وهنا يأتي دور مدير الذاكرة (Memory Manager) حيث يراقب هذا البرنامج أجزاء الذاكرة ويتعرف على ما هو مستخدم وما هو غير مستخدم كما يوفر دوالا لحجز مقاطع الذاكرة وتحريرها ، كذلك من الممكن أن يقوم بعملية إعادة تجزئة لمقاطع الذاكرة وذلك لتوفير مساحة واستغلال أفضل.وفي هذا الفصل سيتم دراسة مدير الذاكرة الفيزيائية والتخيلية وكيفية برمجتهم كذلك سيتم عرض بعض الطرق لحساب المساحة الكلية للذاكرة وعرض مقاطع الذاكرة المستخدمة من قبل النظام.

١.١ إدارة الذاكرة الفيزيائية Physical Memory Management

الذاكرة الفيزيائية (Physical Memoery) هي كتلة ذاكرية توجد بداخل شريحة الذاكرة الرئيسية RAM وتختلف طريقة حفظ البيانات فيها بحسب نوع الذاكرة (فمثلا تستخدم المكثفات والترانزستورات لحفظ البتات في ذاكرة DRAM) ، ويتم التحكم في هذه الكتلة بواسطة شريحة داخل اللوحة الأم تسمى بمتحكم الذاكرة أو بالجسر الشمالي (North Bridge) حيث يقوم هذا المتحكم بعملية إنعاش المكثفات بداخل الذاكرة حتى تحافظ على محتوياتها من الضياع كذلك يعنون هذا المتحكم الذاكرة الفيزيائية بداية من أول ٨ بتات حيث تأخذ العنوان الفيزيائي 0x0 ويليها العنوان 0x1 للثمانية بتات التالية ، وهكذا تتم عنونة الذاكرة من قبل المتحكم. وعند وصول طلب قراءة أو كتابة الى متحكم الذاكرة فان المتحكم يقوم بقراءة العنوان من مسار العناوين وبالتالي يتمكن من تفسير العنوان وتوجيهه الى المكان الصحيح حيث غالبا ما توجد أكثر من شريحة رام بداخل الحاسب. وتشكل مجموعة كل العناوين في كل شرائح الذاكرة مساحة عناوين فيزيائية (Physical Address Space) .لكن قد يكون هناك عددا من هذه العناوين غير مستخدمة فعليا وذلك في حالة وجود شريحة ذاكرة في المنفذ الأول والثالث وعدم وجود شريحة ذاكرة في المنفذ الثاني وفي هذه الحالة اذا كان حجم كل شريحة ذاكرة هي n فان العناوين من n-1 الى 2n-1 هي عناوين غير موجودة حقيقة وتسمى فتحات (Holes) ، وعملية الكتابة في هذه الفتحات لا تؤثر شيء على النظام بينما عند القراءة من هذه الفتحات فان البيانات الموجودة في مسار البيانات هي التي يتم قرائتها على الرغم من أنها غير صحيحة. وقبل الخوض في برمجة مدير الذاكرة يجب توفير عددا من المعلومات مثل جحم الذاكرة الكلي ومناطق الذاكرة المستخدمة وغير المستخدمة و الفتحات الموجودة ان كانت هناك فتحات ، وهذا حتى يتمكن مدير الذاكرة من إدارتها على النحو المطلوب.

١.١.١ حساب حجم الذاكرة

توجد عدة طرق لحساب حجم الذاكرة بعضها تعتمد على النظام والأخرى لا تعتمد ، وفي بداية إقلاع الحاسب يقوم نظام البايوس بالتخاطب مع متحكم الذاكرة وأخذ حجم الذاكرة ويقوم بحفظها في أحد العناوين في منطقة بيانات البايوس في الذاكرة. وتعتبر هذه الطريقة هي الأكثر دقة في الحصول على حجم الذاكرة ولكن تبقى مشكلة أن مقاطعات البايوس لا يمكن استدعائها بداخل النواة والتي تعمل في النمط المحمي. لذلك سيتم استخدام المقاطعة الخاصة بجلب حجم الذاكرة قبل أن تبدأ النواة في عملها وتمرير هذا الحجم مع بعض المعلومات الاخرى و تسمى معلومات الإقلاع المتعدد الى النواة.

المقاطعة int 0x12

تقوم هذا المقاطعة بجلب حجم الذاكرة من منطقة بيانات البايوس (بالتحديد من العنوان 0x413) لكنها لا تستخدم حاليا نظرا لأنها تجلب حجم ٦٤ كيلوبايت كحد أقصى وبالتالي اذا كانت الذاكرة لديك أكبر من هذا الحجم فانها لا تجلب الحجم الصحيح ، لذلك لا تستخدم هذه المقاطعة الان. والمثال ١.١ يوضح كيفية استخدامها

Example ١.١: Using int 0x12 to get size of memory

```
١
٢ ;————————————————————————
٣ ; get_conventional_memory_size
٤ ; ret ax=KB size from address 0
٥ ;————————————————————————
٦ get_conventional_memory_size:
٧   int 0x12
٨   ret
```

المقاطعة 0x15 الدالة 0xe801

تجلب هذه المقاطعة الحجم الصحيح وتستخدم دائما لهذا الغرض ، وتعود بالقيم:

- العلم CF: صفر في حالة نجاح عمل المقاطعة.
- المسجل EAX: عدد الكيلوبايتات من العنوان 1 MB الى 16 MB.
- المسجل EBX: عدد الوحدات المكونة من ٦٤ كيلوبايت بدئا من العنوان 16 MB، ويجب ضربها لاحقا بالعدد ٦٤ حتى يتم تحويلها الى عدد الكيلوبايتات.

وفي بعض الأنظمة يستخدم البايوس المسجلين ECX و EDX بدلا من المسجلين EAX و EBX. والمثال ١.٢ يوضح كيفية استخدام هذه المقاطعة.

Example ١.٢: Using int 0x15 Function 0xe801 to get size of memory

```
١
٢ ; ——————————————————————
٣ ; get memory size:
٤ ; get a total memory size.except the first MB.
٥ ; return:
٦ ;   ax=KB between 1MB and 16MB
٧ ;   bx=number of 64K blocks above 16MB
٨ ;   bx=0 and ax= —1 on error
٩ ; ——————————————————————
١٠ get_memory_size:
١١
١٢     push ecx
١٣     push edx
١٤     xor ecx,ecx
١٥     xor edx,edx
١٦
١٧     mov ax,0x801    ; BIOS get memory size
١٨     int 0x15
١٩
٢٠     jc .error
٢١     cmp ah,0x86
٢٢     je .error
٢٣     cmp ah,0x80
٢٤     je .error
٢٥
٢٦     jcxz .use_eax
٢٧
٢٨     mov ax,cx
٢٩     mov bx,dx
٣٠
٣١   .use_eax:
٣٢     pop edx
٣٣     pop ecx
٣٤     ret
٣٥
٣٦   .error:
```

```
٣٧      mov ax,-1
٣٨      mov bx,0
٣٩      pop edx
٤٠      pop ecx
٤١      ret
```

ويجدر ملاحظة أن هذا المقاطعة لا تحسب أول ميجا بايت من الذاكرة ، لذلك عند استخدام هذه الدالة يجب زيادة الحجم الكلي بمقدار ١٠٢٤ كيلوبايت (الزيادة تكون في مسجل ax) ، كذلك يجب ضرب المسجل bx بالعدد ٦٤ نظرا لان القيمة التي تضعها الدالة هي عدد الوحدات المكونة من ٦٤ كيلوبايت ، وهذا يعني ان كان العدد هو ٢ مثلا فان النتيجة يجب أن تكون 64*2 وتساوي 128 كيلوبايت.

٢.١.١ خريطة الذاكرة Memory Map

بعد أن تم حساب حجم الذاكرة يجب الانتباه الى أن عددا منها محجوز لبعض الأغراض (راجع فقرة خريطة الذاكرة في فصل إقلاع الحاسب) وهنا يأتي دور خريطة الذاكرة التي تعرف وتحدد لنا مقاطع الذاكرة المستخدمة والغير مستخدمة. ويمكن الحصول على خريطة الذاكرة بواسطة مقاطعة البايوس int 0x15 الدالة e820 التي تأخذ المدخلات التالية:

- العلم CF: صفر في حالة نجاح عمل المقاطعة.
- المسجل eax: القيمة 0x534d4150 وتساوي SMAP بترميز ASCII، وفي حالة حدوث خطأ فان رقم الخطأ سيحفظ في المسجل ah.
- المسجل ebx: عنوان السجل التالي أو صفر في حالة الإنتهاء.
- المسجل ecx: طول المقطع بالبايت.
- المسجلان es:di: سجل واصفة المقطع.

وتعود بالمخرجات:

- المسجل eax: رقم الدالة وهي 0xe820.
- المسجل ebx: عنوان البداية.
- المسجل ecx: حجم الذاكرة المؤقتة (buffer) وتساوي ٢٠ بايت على الأقل.
- المسجل edx: القيمة 0x534d4150 وتساوي SMAP بترميز ASCII.
- المسجلان es:di: عنوان الذاكرة buffer التي ستحفظ بها النتائج.

وبعد تنفيذ المقاطعة فان الذاكرة Buffer يتم ملئها بأول سجل وهو بطول ٢٤ بايت ويتم تكرار استدعاء المقاطعة الى أن تكون قيمة المسجل ebx مساوية للصفر. ومحتويات كل سجل يوضحها المثال ١.٣.

Example ١.٣: Memory Map Entry Structure

```
١
٢ ; Memory Map Entry Structure
٣ struc memory_map_entry
٤    .base_addr  resq  1
٥    .length    resq  1
٦    .type     resd  1
٧    .acpi_null  resd  1
٨ endstruc
```

وهي توصف مقطع الذاكرة حيث تحوي عنوان بداية المقطع وطوله ونوع المقطع وأخيرا بيانات محجوزة. ونوع المقطع يحدد ما إذا كان المقطع مستخدما أو محجوزا ويأخذ عدة قيم:

- القيمة 1: تدل على أن المقطع متوفرا.
- القيمة 2: تدل على أن المقطع محجوزا.
- القيمة 3: ACPI Reclaim Memory وهي منطقة محجوزة لكي يستخدمها النظام.
- القيمة 4:ACPI NVS Memory كذلك هي منقطة محجوزة للنظام.
- بقية القيم الأخرى تدل على أن المقطع غير معرف أو غير موجود.

والمثال ١.٤ يوضح كيفية جلب مقاطع الذاكرة لكي يستفيد منها مدير الذاكرة لاحقا.

Example ١.٤: Get Memory Map

```
١ ; ------------------------------------------------
٢ ; get_memory_map:
٣ ; Input:
٤ ;    EAX = 0x0000E820
٥ ;    EBX = continuation value or 0 to start at beginning of map
٦ ;    ECX = size of buffer for result (Must be >= 20 bytes)
٧ ;    EDX = 0x534D4150h ('SMAP')
٨ ;    ES:DI = Buffer for result
٩ ;
١٠ ; Return:
١١ ;    CF = clear if successful
```

```
;   EAX = 0x534D4150h ('SMAP')
;   EBX = offset of next entry to copy from or 0 if done
;   ECX = actual length returned in bytes
;   ES:DI = buffer filled
;   If error, AH containes error code
; ------------------------------------------------
get_memory_map:

    pushad
    xor ebx,ebx
    xor bp,bp
    mov edx,'PAMS'    ; 0x534D4150
    mov eax,0xe820
    mov ecx,24
    int 0x15      ; BIOS get memory map.

    jc .error
    cmp eax,'PAMS'
    jne .error

    test ebx,ebx
    je .error

    jmp .start

  .next_entry:
    mov edx,'PAMS'    ; 0x534D4150
    mov eax,0xe820
    mov ecx,24
    int 0x15      ; BIOS get memory map.

  .start:
    jcxz .skip_entry

    mov ecx,[es:di + memory_map_entry.length]
    test ecx,ecx
    jne short .good_entry
    mov ecx,[es:di + memory_map_entry.length+4]
    jecxz .skip_entry

  .good_entry:
```

```
٥٣      inc bp
٥٤      add di,24
٥٥
٥٦    .skip_entry:
٥٧      cmp ebx,0
٥٨      jne .next_entry
٥٩      jmp .done
٦٠
٦١    .error:
٦٢      stc         ; set carry flag
٦٣
٦٤    .done:
٦٥      popad
٦٦      ret
٦٧
٦٨ endstruc
```

٣.١.١ مواصفات الإقلاع المتعدد

العديد من محملات النظام (Bootloader) تدعم الإقلاع المتعدد لمختلف أنظمة التشغيل وذلك عبر مواصفات ومقاييس محددة يجب أن يلتزم بها محمل النظام عند تحميل نواة النظام. ومن ضمن هذا المقياس تمرير بيانات الإقلاع المتعدد (Multiboot Information) من محمل النظام الى نواة نظام التشغيل. وما يهمنا حاليا هو تمرير حجم الذاكرة وخريطة الذاكرة الى النواة حتى يتمكن مدير الذاكرة من إدارتها بناءا على خريطة الذاكرة وحجمها. حيث ذكرنا سابقا أنه أثناء عمل النواة لا توجد طريقة مبسطة لتحديد حجم الذاكرة ومخطط المقاطع فيها ، لذلك تم اللجوء الى مقاطعات البايوس والإستفادة من خدماته ومن ثم تمرير النتائج الى نواة النظام عن طريق هيكلة قياسية[١].

حالة الحاسب

من ضمن هذه المقاييس أيضا توفر حالة معينة للحاسب (Machine State) ،وهي تنص على أنه عند تحميل نواة أي نظام تشغيل فان بعض المسجلات يجب أن تأخذ قيما محددة كالاتي:

- المسجل eax: يجب أن يأخذ الرقم 0x2BADB002 وهي إشارة لنواة النظام بأن محمل النظام يدعم الإقلاع المتعدد.
- المسجل ebx: تحتوي على عنوان بداية هيكلة الإقلاع المتعدد.

[١]على الرغم من أنه يمكن تمرير هذه البيانات بأي طريقة إلا أن الإلتزام بهيكلة قياسية سيفيد لاحقا عند دعم الإقلاع المتعدد.

- المسجل cs: واصفة الشفرة يجب أن تكون ٣٢ بت قراءة/تنفيذ بدءا من العنوان 0x0 وانتهاءا بالعنوان 0xffffffff.
- المسجلات ds,es,fs,gs,ss: يجب أن تكون مقاطع القراءة والكتابة ٣٢ بت وتبدأ من العنوان 0x0 وتنتهي بالعنوان 0xffffffff.
- يجب تفعيل بوابة a20.
- مسجل التحكم cr0: البت 0 يجب أن يفعل (تفعيل النمط المحمي) والبت ٣١ يجب أن يعطل (تعطيل التصفيح).

معلومات الإقلاع المتعدد

تعتبر هيكلة معلومات المتعدد من أهم الهياكل التي يجب تمريرها الى النواة ، ويتم حفظ عنوانها في المسجل ebx وهي الطريقة القياسية التي يتم بها تمرير الهيكلة الى النواة ، لكن بسبب أننا في هذه المرحلة لا ندعم الإقلاع المتعدد فيمكن أن نمرر هذه البيانات بأي شكل كان كدفع عنوانها الى المكدس (stack). والمثال ١.٥ يوضح هيكلة معلومات الإقلاع المتعدد.

Example ١.٥: Multiboot Inforamtion Structure

```
١ boot_info:
٢
٣ istruc multiboot_info
٤   at multiboot_info.flags,        dd 0
٥   at multiboot_info.mem_low,        dd 0
٦   at multiboot_info.mem_high,       dd 0
٧   at multiboot_info.boot_device,      dd 0
٨   at multiboot_info.cmd_line,       dd 0
٩   at multiboot_info.mods_count,     dd 0
١٠  at multiboot_info.mods_addr,      dd 0
١١  at multiboot_info.sym0,         dd 0
١٢  at multiboot_info.sym1,         dd 0
١٣  at multiboot_info.sym2,         dd 0
١٤  at multiboot_info.mmap_length,      dd 0
١٥  at multiboot_info.mmap_addr,      dd 0
١٦  at multiboot_info.drives_length,    dd 0
١٧  at multiboot_info.drives_addr,      dd 0
١٨  at multiboot_info.config_table,     dd 0
١٩  at multiboot_info.bootloader_name,    dd 0
٢٠  at multiboot_info.apm_table,      dd 0
```

```
٢١   at multiboot_info.vbe_control_info,   dd 0
٢٢   at multiboot_info.vbe_mode_info,    dw 0
٢٣   at multiboot_info.vbe_interface_seg,  dw 0
٢٤   at multiboot_info.vbe_interface_off,  dw 0
٢٥   at multiboot_info.vbe_interface_len,  dw 0
٢٦ iend
```

ويحدد المقياس استخدام المتغير flags لتحديد البيانات المستخدمة في هيكلة معلومات الإقلاع المتعدد من غير المستخدمة ، فمثلا في حالة كانت قيمة البت flags[0] هي ١ فان المتغيرات mem_low و mem_high يمكن استخدامها وهكذا.وحاليا لن يتم التركيز على المتغير flags وسيتم وضع قيم للمتغيرات mem_low و mem_high والتي تحوي حجم الذاكرة الفيزيائية والتي تم جلبها بواسطة البايوس ، وكذلك المتغيرات mmap_addr و mmap_length والتي توضح عنوان بداية مخطط الذاكرة الذي تم جلبه أيضا بواسطة البايوس وكذلك طولها.والمثال ١.٦ يوضح الأوامر التي تم إضافتها الى المرحلة الثانية من محمل النظام لدعم الإقلاع المتعدد وكيفية حفظ معلومات الإقلاع المتعدد وارسالها الى نواة نظام التشغيل.

Example ١.٦: snippet from stage2 bootloader

```
١
٢ ...
٣ ; when stage2 begin started, BIOS put drive number where stage1 are
      loaded from in dl
٤       mov [boot_info+multiboot_info.boot_device],dl
٥ ...
٦
٧ ;------------------
٨ ; Get Memory Size
٩ ;------------------
١٠      xor eax,eax
١١      xor ebx,ebx
١٢      call get_memory_size
١٣
١٤      mov [boot_info+multiboot_info.mem_low],ax
١٥      mov [boot_info+multiboot_info.mem_high],bx
١٦
١٧ ...
١٨
١٩ ;-----------------------------------------------
٢٠ ; Pass MultiBoot Info to the Kernel
٢١ ;-----------------------------------------------
٢٢      mov eax,0x2badb002
```

```
٢٣      mov ebx,0
٢٤      mov edx,[kernel_size]
٢٥      push dword boot_info
٢٦
٢٧      call ebp     ; Call Kernel
```

وعند استدعاء النواة فان البيانات التي تم دفعها الى المكدس تعتبر كوسيط للدالة ، وتم انشاء هيكلة بلغة السي (بداخل النواة) بنفس حجم الهيكلة التي تم دفعها الى المكدس وذلك لقراءة محتوياتها بشكل مبسط ومقروء.والمثال ١.٧ يوضح كيفية استقبال النواة لهذه الهيكلة.

Example ١.٧: Kernel Entry

```
١
٢ void _cdecl kernel_entry (multiboot_info* boot_info)
٣ {
٤
٥ #ifdef i386
٦   _asm {
٧     cli
٨
٩     mov ax, 10h   // select data descriptor in GDT.
١٠    mov ds, ax
١١    mov es, ax
١٢    mov fs, ax
١٣    mov gs, ax
١٤  }
١٥ #endif
١٦
١٧  // Execute global constructors
١٨  init_ctor();
١٩
٢٠  // Call kernel entry point
٢١  main(boot_info);
٢٢
٢٣  // Cleanup all dynamic dtors
٢٤  exit();
٢٥
٢٦ #ifdef i386
٢٧  _asm {
٢٨    cli
٢٩    hlt
```

```
٣٠    }
٣١ #endif
٣٢
٣٣    for(;;);
٣٤ }
```

٤.١.١ مدير الذاكرة الفيزيائية

٢.١ إدارة الذاكرة التخيلية Virtual Memory Management