المرحلة الثالثة : نواة ب C++

في الدروس السابقة، تعرفنا على كيفية كتابة قطاع الانطلاق، أوضاع العنونة في x86 ثم تحميل النواة من القرص المرن إلى الذاكرة.
القاسم المشترك بين الدروس السابقة كان استعمال الأسمبلي لتطوير البرامج. في هذا الدرس سنوضح كيف يمكن استعمال لغة من أعلى مستوى وهي C++، سنتخلى أيضا عن محمل النواة البدائي الذي طورناه في الدروس السابقة لفائدة محمل آخر أكثر تطورا و أكثر نضجا وهو كروب Grub (Grand unified boot loader).

في البداية سنوضح كيف يجب كتابة النواة لدعم التحميل من طرف كروب (1)، بعد ذلك سنوضح كيفية كتابة نواتنا ب C++ (2) ثم نستغل هذه اللغة في كتابة بعض العناصر للتعامل مع الشاشة النصية وأيضا تثبيت جدول للموصفات العام (3).

قبل البدء في الدرس، سنحتاج إلى بعض الأدوات بلإضافة إلى المجمع nasm:

- أدوات gcc: و تشمل المترجم و binutils. إذا كنت في بيئة linux، فهذه الأدوات موجودة سلفا، بالنسبة لمستعملي ويندوز يمكن الحصول على كل الأدوات بتثبيت cygwin. http://www.cygwin.com
- المحمل كروب grub: يمكن الحصول عليه من موقعه الأصلي http://www.gnu.org/software/grub

1- تحميل النواة بواسطة كروب

كروب برنامج لتحميل نظام تشغيل أو عدة أنظمة إلى الذاكرة، و هو تنجيز implementation لما يسمى ب "توصيفة الإقلاع المتعدد" multiboot specificaion. هذه الأخيرة توصف المهام التي ينبغي لمحمل النواة أن يتكفل بها وكذا الخدمات التي يجب عليه توفيرها للنظام المستفيد.

استعمال كروب يمنح العديد من الامتيازات من ضمنها : إمكانية التحميل من دعامات مختلفة كالأقراص الصلبة، المرنة أو عبر الشبكة – دعم ملفات تنفيذية مختلفة مثل elf, aout... – دعم ملفات مضغوطة ب gzip – التعرف على عدة أنظمة ملفات Fat – extfs...

لكتابة نواة قابلة للتحميل بكروب (أو أي محمل يحترم توصيفة الإقلاع المتعدد) :

1. النواة يجب أن تكون معدة للاشتغال في بيئة 32 بيت.
2. على الملف التنفيذي أن يحتوي على رأس header بهيأة محددة، وعلى عنوان بداية هذا الرأس أن يكون مسطرا على حدود 32 بيت و أن يتواجد في 8192 بايت الأولى من الملف. الرأس يتوفر على حقول إجبارية و أخرى اختيارية:

البعد	الإسم	الوصف	إجباري؟
0	"كلمة السر" Magic field	هذا الحقل يستعمل كعلامة على بداية الرأس داخل الملف	نعم
4	العلامات flags	تحدد البيتات المعلومات المطلوب توفيرها	نعم
8	مجموع التحقق checksum	عدد من 32 بيت بحيث إذا تمت إضافته إلى الحقلين السابقين يكون المجموع 0	نعم
12	عنوان الرأس header adress	العنوان الذي يجب نقل الرأس إليه في الذاكرة	فقط إذا كان البيت 16 من flags يساوي 1
16	عنوان التحميل load adress	عنوان بداية فقرة الشفرة (text) في الذاكرة	فقط إذا كان البيت 16 من flags يساوي 1
20	نهاية عنوان التحميل	عنوان نهاية فقرة البيانات	فقط إذا كان البيت 16

	load end adress	في الذاكرة (data)	من flags يساوي 1
24	عنوان نهاية فقرة bss	عنوان نهاية فقرة bss إي فقرة البيانات غير المجهزة يقوم uninitialized data المحمل بملء المساحة المخصصة لهذه البيانات ب 0	فقط إذا كان البيت 16 من flags يساوي 1
28	عنوان المدخل entry adress	العنوان الذي ينبغي أن يقفز إليه المحمل في النواة بعد إتمام التحميل	فقط إذا كان البيت 16 من flags يساوي 1
من 29 إلى 44	حقول خاصة بتوفير معلومات عن أوضاع الفيديو التي يفضل تثبيتها لبرنامج التشغيل		فقط إذا كان البيت 2 من flags يساوي 1

بالنسبة لحقل العلامات flags، يحتوي على 32 بيت، لكن ما يهم فيه هو:

- البيت 0: 1 تعني عنوان التحميل يجب أن يكون مسطرا على 4 كيلوبايت
- البيت 1: 1 تعني أنه ينبغي على المحمل إدخال المعلومات التي تشمل حجم الذاكرة في البنية المسماة multiboot_info. هذه الأخيرة تمد النواة بمعلومات مختلفة ويكون عنوانها مخزنا في ebx بعد التحميل. ندرج فيما يلي تعريف البنية (أدخلت فقط الأربعة عناصر الأولى من هذه البنية وهي الأهم والتي سنستعملها في درسنا، لمعرفة كامل عناصر البنية يرجى قراءة mutiboot specification (http://www.gnu.org/software/grub/manual/multiboot/

```
typedef struct multiboot_info
{
   unsigned long flags;
   unsigned long mem_lower;
   unsigned long mem_upper;
   unsigned long boot_device;
} multiboot_info_t;
```

الحقل الأول flags يخبرنا بالمعلومات التي تتوفر عليها البنية، إذا كان البيت 0 من هذا الحقل يساوي 1 فهذا يعني أن الحقلين التاليين mem_lower و mem_upper متوفران ويحملان: الأول حجم الذاكرة السفلى (الموجودة في أول ميغابايت أقصى قيمة لهذا الحقل هي 640 كيلوبايت) و الثاني حجم الذاكرة العليا (الموجودة بعد أول ميغابايت). جميع الأحجام بالكيلوبايت.
إذا كان البيت 1 من هذا الحقل flags يساوي 1 فهذا يعني أن الحقل boot_device متوفر و يعطينا نوع الدعامة التي تم إقلاع النواة منها: 0 تعني القرص المرن الأول، و 0x80 تعني القرص الصلب الأول.

- البيت 2: 1 تعني أن الرأس يحتوي على البيانات الخاصة بأوضاع الفيديو.
- البيت 16: 1 تعني أن الرأس يحتوي على البيانات الخاصة بالعناوين.

لفهم الحقول المتعلقة بالعناوين ينبغي شرح البنية العامة للملفات التنفيذية.
لنفترض مثلا أن لدينا ملفين لشفرة ب مكتوبين ب C و اسمهما *one.c* و *tow.c* على التوالي

one.c	tow.c

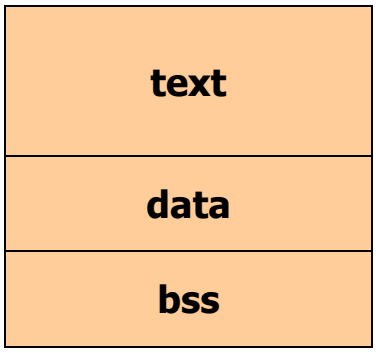
`int a;`
`extern void func();`
`void main() {`
`  func();`
`}`
	`extern int a;`
	`int b=10;`
	`void func() {`
	`  a=5;`
	`  b=2;`
	`}`

عندما نقوم بترجمة الملفين يقوم المترجم في مرحلة أولى بترجمة كل ملف مصدري إلى ملف يسمى ملف عنصر *object file*. كما يرى من الشفرتين فالملف *one.c* يقوم بتعريف متغير عام اسمه a و دالة اسمها *main*، في حين يُعرف *tow.c* متغيرا *b* و دالة *func*. كل هذه الأسماء المعرفة تشكل ما يسمى رموزا في ميدان الترجمة. إذا نظرنا إلى الملف *one.c* نجد أنه يقوم بنداء الدالة *func()* إي يقوم باستعمال الرمز *func* رغم أنه غير معرف داخله، السبب يكمن في التعليمة *extern void func()* التي تخبر المترجم أن الرمز *func* معرفة في ملف آخر فيقوم هذا الأخير بالترجمة الجزئية ويحتفظ في الملف الموضوع بمعلومة أن الملف يحتوي على رمز خارجي اسمه *func*. نفس الشيء بالنسبة للملف *tow.c* الذي يحتوي على إشارة لرمز خارجي و هو *a*.

بعد مرحلة الترجمة التي تنتج ملفات العناصر، يقو برنامج اسمه محرر الروابط linker بإنتاج الملف التنفيذي، يتم ذلك عن طريق جمع التعليمات التي في ملفات العناصر و دمجها في ملف واحد، لأجل ذلك يجب فك الروابط بين مختلف الملفات، مثلا في ملف عنصر *one.o* ناتج عن ترجمة *one.c* على محرر الروابط فك الرابط *func()*. بما أن محرر الروابط يعرف جميع الرموز في كل الملفات فإنه يعرف عناوينها في الملف التنفيذي النهائي (و أيضا في الذاكرة عند تحميل الملف التنفيذي لأنه يجب إخبار محرر الروابط بموقع التحميل) بذلك يقوم بكتابة العناوين الصحيحة في جميع التعليمات التي كانت تحتوي على إشارات خارجية.

يقوم محرر الروابط بتقسيم الملف التنفيذي النهائي إلى فقرات، كل فقرة معدة لاحتضان نوع متجانس من المعلومات. عادة نجد الفقرات التالية:

1. فقرة تحتوي على الشفرة وتسمى عادة ب text، في مثالنا السابق الفقرة ستشمل كل التعليمات في الدالتين main و func.
2. فقرة تحتوي على البيانات المجهزة تشمل المتغيرات التي تعرف قيمتها ساعة الترجمة. تكون عادة بعد الشفرة و تسمى data في حالتنا ستشمل المتغير b. في بعض الأحيان تكون أيضا هناك فقرة إضافية تسمى (read only data) rodata أو rdata لتخزين البيانات الثابتة const.
3. فقرة تحتوي على البيانات غير المجهزة تشمل المتغيرات التي لم تحدد أي قيمة لها ساعة الترجمة تكون عادة بعد data و تسمى bss في حالتنا ستشمل المتغير a.



في درسنا نحن نستعمل محرر الروابط المرفق مع gcc و يسمى ld. لتحديد الفقرات التي يجب أن يحتويها الملف التنفيذي يجب علينا كتابة ملف صغير يسمى linker script، لاحقا سنقول مخطوط الروابط. هذا المخطوط يصف محتوى كل الفقرات المطلوب إدماجها وأيضا عناوينها أثناء تحميلها في الذاكرة، هذا بالإضافة إلى بعض المعلومات الأخرى، يمرر المخطوط إلى محرر الروابط عبر سطر الأوامر باستعمال الاختيار –Tsrcipt_file حيث script_file هو اسم ملف المخطوط. في حالة عدم تحديد هذا الملف فإن ld يستعمل مخطوطا داخليا ويكون معدا –من قبل- لملائمة نظام التشغيل الذي يتم الاشتغال فيه. لكن في حالتنا نحن لا نستند إلى أي نظام مسبق بالإضافة إلى أننا نريد ملفا تنفيذيا بمواصفات خاصة لذلك سنستعمل مخطوطنا الخاص بنا.

في مخطوط الروابط يمكننا تعريف الفقرات التي ستؤلف الملف التنفيذي، ترتيبها و مواقع بدايتها في الذاكرة، فيما يلي سنقدم المخطوط الذي سنستعمله مع شرح موجز، إذا أردت معرفة المزيد عن كتابة مخطوط لمحرر الروابط ld أنظر موقع ld

```
ENTRY(entry)

SECTIONS
{
        . = 0x00100000;
        code = .;

        .text  . : {
                *(.text*)
        }

        . = ALIGN(4096);

        data = .;
        .data . :  {
                *(.*data*)

        }

        bss = .;
        .bss :  {
                *(COMMON)
                *(.bss*)
                . = ALIGN(4096);
                *(.stack)
        }
        end = .;
        /DISCARD/ :{
                *(.note*)
                *(.indent)
                *(.comment)
                *(.stab)
                *(.stabstr)
            }
}
```

.text

*الفقرة text. تشمل كل الفقرات التي تكتب على شاكلة *.text معدة للشفرة*

.data

*الفقرة data. تشمل كل الفقرات التي تكتب على شاكلة *.data معدة للبيانات المجهزة*

.bss

*الفقرة bss. تشمل كل الفقرات التي تكتب على شاكلة *.bss أو COMMON معدة للبيانات غير المجهزة*

bss. تحتوي أيضا على فقرة خاصة اسمها stack. المسجل esp سيشير في البداية إلى نهاية هذه الفقرة

Entry(start) تعرف التعليمة الأولى التي سيتم تنفيذها بعد التحميل أي مدخل الملف التنفيذي.

Sections تفتح باب تعريف الفقرات المؤلفة للملف، أولا نقوم بتغيير قيمة متغير خاص "." (النقطة في المخطوط متغير خاص يسمى "عداد الموقع" location counter و تشير إلى الموقع الحالي – أي أثناء عملية تحرير الروابط - في الملف) مما يعني أننا نجعل العنوان الحالي يبدأ في 0x100000 و هو عنوان التحميل في الذاكرة، نقوم بتعريف رمز اسمه *code* و نعطيه قيمة المتغير "." (أي أن code تشير حاليا إلى بداية فقرة الشفرة المعرفة مباشرة بعده والتي هي text.) الفقرة الأولى إذن تسمى *text*. وتبدأ في 0x100000، التعليمة *(.text*)* تعني إدماج محتويات جميع الفقرات الموجودة في جميع ملفات العناصر و التي يبدأ اسمها ب text. داخل هذه الفقرة. بعد ذلك نقوم بتغيير الموقع الحالي عبر تغيير قيمة عداد الموقع. التعليمة ALIGN(4096) تنتج لنا عنوانا مسطرا على 4 كيلوبايت بعد الموقع الحالي و هدفها جعل فقرة البيانات و التي تلي فقرة الشفرة تبدأ في عنوان مسطر على 4 كيلوبايت.

بعد الفقرة text. نعرف فقرة data. المعدة لإيواء البيانات المجهزة، المتغير data يشير إلى بداية الفقرة والتي سندمج فيها محتويات جميع الفقرات التي يكتب اسمها على شاكلة *data*. مثل data. أو rdata. أو rodata.، بنفس الطريقة نقوم بتعريف الفقرة bss.، هذه الأخيرة نضيف إليها محتوى فقرة خاصة (.stack) سنعرفها فيما بعد هدفها توفير مساحة للكومة التي ستستعملها النواة. في آخر الفقرات المتغير end سيشير إلى نهاية الملف. التعليمات التي بعد DISCARD هدفها إزاحة محتويات الفقرات التي لا حاجة لنا بها والتي ينتجها المترجم في ملفات العناصر.

في هذا الدرس، سنحتاج أيضا للأسمبلي، الملف التاليmultiboot.s يتضمن الرأس الخاص بكروب بالإضافة إلى التعليمات التي تقوم بإعطاء اليد للنواة الفعلية المكتوبة ب C++.

فيما يلي نص الشفرة بالأسمبلي في ملف multiboot.s

```
%define MULTIBOOT_HEADER_MAGIC 0x1BADB002        ; magic header
%define MULTIBOOT_HEADER_FLAGS 0x00010003
                  ;; flags:
                     ;; align to 4 kb
                     ;; give us memory info
                     ;; header contains address fields
%define MULTIBOOT_CHECKSUM -(MULTIBOOT_HEADER_MAGIC +
MULTIBOOT_HEADER_FLAGS)
                        ;; remember: sum of this field and the tow first
fields must be zero

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; code section

SECTION .text
[BITS 32]

      GLOBAL entry      ; must be global to be visible to the linker
      entry:
        jmp start
       ; extern symbols defined in the linker script file
       EXTERN code, bss, end
       ALIGN 4    ; mutiboot header must be 4 bytes aligned

       multiboot_header:
         dd MULTIBOOT_HEADER_MAGIC
         dd MULTIBOOT_HEADER_FLAGS
         dd MULTIBOOT_CHECKSUM
         dd multiboot_header
         dd code
         dd bss
         dd end
         dd entry

       ; program entry point, the code of kernel start here
       GLOBAL entry; must be global to be visible to the linker

       start:
         mov esp, stack
         ; we declare tow extern symbols because we don't know for the
       moment
         ; wich symbol the compiler will generate from our c++ file:
       kmain (linux compiler)
         ; or _kmain (cygwin compiler)
```

```
         extern _kmain, kmain
        ; Push kmain args and then call it
         push ebx
         push MULTIBOOT_HEADER_MAGIC
        %ifdef LEADING_USCORE
          call _kmain
        %else
          call kmain
        %endif
        jmp $

SECTION .stack
    resb 0x4000   ; we reserve space for our stack
    stack:
```

الأسطر الثلاثة الأولى تعرف بعض القيم المستخدمة فيما بعد، القيمة الأولى مزدوجة (double word) تعرف "كلمة السر" الضرورية في أول الرأس. التعريف الثاني خاص بالعلامات flags الموجهة للمحمل كروب: تسطير النواة على عنوان 4 كيلوبايت+توفير معلومات عن الذاكرة+الرأس يحتوي على حقول العناوين مباشرة بعد الحقول الثلاثة الإجبارية. يأتي بعدهما التعريف الثالث لمجموع التحقق الذي كما قلنا من المفروض أن يساوي ناقص مجموع الحقول السابقة.

```
Section .text
```

تقوم بتعريف فقرة اسمها text. سنستخدمها لإيواء الشفرة.

```
Bits 32
```

الشفرة على 32 بيت.

```
EXTERN code, bss, end
```

تشير إلى الرموز الخارجية المعرفة في مخطوط الروابط. كما ترى داخل الشفرة يمكن أن نستخدم أيضا الرموز المعرفة في المخطوط.

```
ALIGN 4
```

لأن الرأس يجب أن يبدأ في عنوان مسطر على 4 بايت

بعد ذلك تأتي التعريفات الخاصة بالرأس، كتمرين يمكنك مطابقتها مع الجدول أعلاه.

```
start:
  mov esp, stack
```

هنا يبدأ مدخل الملف التنفيذي، قبل كل شيء يجب تثبيت الكومة التي سيستعملها البرنامج، يتم ذلك عبر جعل esp يشير إلى العنوان *stack* المعرف في آخر الملف.

```
    extern _kmain, kmain
   ; Push kmain args and then call it
    push ebx
    push MULTIBOOT_HEADER_MAGIC
  %ifdef LEADING_USCORE
     call _kmain
  %else
     call kmain
```

```
%endif

jmp $
```

بعد تثبيت الكومة، نقوم بنداء الدالة *kmain* والتي ستعرف في ملف خارجي مكتوب ب C++، هذه الدالة هي المدخل الفعلي لبرنامج النواة والتي ستتسلم مقاليد الأمور.لاحظ أننا نعرف رمزين خارجيين kmain و _kmain، ذلك لأننا لا نعرف الم الذي سيختاره المترجم للدالة kmain، مثلا مترجم cygwin يقوم دائما بإضافة "_" إلى أول اسم الدلة مما يعني أنك إذا كتبت دالة اسمها kmain فإن الاسم المنتج في ملف العنصر يكون _kmain، بالمقابل مترجم لينوكس (على الأقل في توزيعة ماندراك التي جربتها) يترك الاسم دون تغيير. لتجاوز هذه الاختلافات نستخدم التجميع الشرطي conditional assmebling أي أنه إذا تم تعريف الاسم LEADING_USCORE فإننا ننادي الدالة بالاسم الذي يبدأ ب "_"، في الحالة العكسية ننادي الدالة باسمها الأصلي. **الاسم LEADING_USCORE سيعرف في سطر الأوامر الموجه ل NASM فقط إذا كان المترجم المستخدم يغير اسم الدالة.** قبل نداء الدالة نقوم بوضع العوامل التي تخصها في الكومة. بما أنه في C++ العوامل تدخل في الكومة ابتداء من أقصى عامل في اليسار فإن تعريف الدالة kmain التي ناديها هنا يصبح كالتالي

```
void kmain(DWORD magic, multiboot_info *mbi)
```

هذا يعني أن التعليمة الأولى *push ebx* تقوم في الواقع بوضع مؤشر على بنية من نوع multiboot_info حيث عنوانها كما رأينا مخزن في ebx. أما التعليمة الثانية فتقوم بوضع العامل الثاني وهو "كلمة السر" magic number حى يتسنى للدالة kmain معرفة ما إذا كان التحميل قد تم فعلا من طرف كروب.

```
SECTION .stack
resb 0x4000   ; we reserve space for our stack
stack:
```

نقوم هنا بتعريف فقرة اسمها stack. مخصصة لإيواء الكومة، التعليمة resb تقوم فقط بتخصيص مساحة فارغة قيمتها 16 كيلوبايت ثم نعرف الرمز stack بحيث يشير إلى آخرها، بذلك عندما قمنا بنقل عنوان هذا الرمز في المسجل esp فإن ما فعلناه في الحقيقة هو صنع كومة تبدأ من آخر بايت في ملف النواة فما تحت.

2- برنامج النواة ب C++

قبل المرور إلى برنامج النواة لا بد من توضيح بعض الأمور: على عكس C، فبعض الوظائف المستعملة في لغة C++ تفترض وجود دعامة تنفيذية "Runtime support" أو بلغة أخرى مكتبات جاهزة و تستند في الغالب على نظام التشغيل الذي يجري الاشتغال فيه.

المشكلة أنه في حالتنا لا يوجد نظام تشغيل جاهز بعد يمكنه توفير الخدمات التي تحتاجها هذه المكتبات، خصوصا خدمات تدبير الذاكرة. هذا يضع قيودا على الوظائف التي يمكننا استعمالها في الوضع الراهن، باختصار الوظائف التالية لا يمكننا استعمالها مباشرة (نحن نفترض أن المترجم المستعمل هو GCC):

- صنع العناصر أو تدميرها عن طريق المعاملين التقليديين new و delete
- الاستثناءات exceptions
- وظائف التعرف RTTI: بالأخص المعاملين typid و dynamic_cast
- استعمال عناصر عامة أو ثابتة Global and static objects: تنشيط هذه الوظيفة يحتاج بعض التعديلات على مخطوط الروابط و على الشفرة، في الوقت الحالي سنستعمل

عناصر محلية مصنوعة في الكومة أي داخل الدوال لأن هذه الأخيرة لا تحتاج إلى إعداد مسبق.
- الدوال الافتراضية الخالصة pure virtual fuctions: تنشيطها يحتاج كتابة دوال خاصة تنادى من طرف المترجم ساعة الترجمة.

جميع الوظائف الأخرى مثل الفئات Classes الاشتقاق derivation والدوال الافتراضية لا تحتاج لدعم مسبق ويمكننا استعمالها بحرية.

بعد معرفة الوظائف المتاحة لنا يمكننا المرور إلى الشفرة.

في درسنا، برنامج النواة يتكون من الملفات التالية:

- ***main.cpp*** و هو الملف الرئيسي، يحتوي على الدالة *kmain* التي تشكل المدخل الفعلي للنواة و بعض الدوال المفيدة التي سنحتاجها.
- ***gdt.cpp*** يضم الدوال المتعلقة بتدبير الوضع المحمي: صناعة جدول للموصفات العام و تثبيته، نذكر هنا بأنه ساعة تنفيذ برنامج النواة، فإن كروب قد قام قبلا بالمرور بالمعالج إلى الوضع المحمي وتثبيت ج.م.ع GDT لكننا سنقوم باستعمال جدول خاص بنا.
- ***system.h*** حاليا يحتوي على تعريف بالدوال المفيدة المنجزة في *main.cpp* حتى يتسنى استعمالها من طرف الملفات الأخرى. يضم هذا الملف يضم أيضا دالتين للقراءة والكتابة في البوابات الفرعية باستعمال الأسمبلي المباشر inline assmbly و هي وظيفة تمكننا من استعمال تعليمات أسمبلر مباشرة في ملفات C++.
- ***Video.cpp/Video.h*** يحتوي على الفئة *Video Class* والتي توفر بعض الخدمات للكتابة على الشاشة النصية.

أولا لنلقي نظرة على الدوال المفيدة في ملف *system.h*

```
#ifndef SYSTEM_H_
#define SYSTEM_H_

#define MULTIBOOT_MAGIC      0x1BADB002

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef unsigned long QWORD;

inline unsigned char inb (unsigned short port)
{
    unsigned char rv;
    asm volatile("inb %1, %0" : "=a" (rv) : "dN" (port));
    return rv;
}


inline void outb (unsigned short port, unsigned char data)
{
    asm volatile("outb %1, %0" : : "dN" (port), "a" (data));
}

BYTE *memcpy(BYTE *dest, const BYTE *src, int count);
BYTE *memset(BYTE *dest, const BYTE val, int count);
void intToString (char *buf, char base, int number);
void GDTSetup();

#endif /*SYSTEM_H_*/
```

في أول الملف نجد بعض التعريفات من نوع *typedef* و التي تخص بعض أنواع البيانات التي سنحتاج التعامل معها. نعرف هنا BYTE أي عدد من 8 بيت، WORD من 16 بيت، DWORD 32 بيت و QWORD 64 بيت. هذا سيجعل الشفرة أكثر تعبيرية.
بعد التعريفات نجد الدالتين inb و outb، وهما لتأدية نفس وظائف التعليمتين in و out بالأسمبلي أي القراءة و الكتابة في البوابات الفرعية.
إذا كنت قد قرأت الدرس السابق فلا شك أنك تذكر أن البوابات الفرعية ما هي إلا خلايا للذاكرة من حجم 16 بيت وكل خلية محددة بعنوان من 16 بيت، و تستعمل للتعامل مع فروع الحاسب مثل لوحة المفاتيح و بطاقات الفيجا، لقراءة و كتابة الفروع بالأسمبلي تستعمل التعليمات in و out. المشكلة أن C و C++ لا تتوفران على تعليمات لهذه الوظائف، لهذا سنستعمل تقنية inline assmebly. لمن لا يعلم ما معنى ذلك، فهذه التقنية تمكننا من إدخال تعليمات بالأسمبلي داخل شفرة C/C++. سأكتفي هنا بشرح ما تفعله التعليمات وللقارئ الذي يريد أن يعرف أكثر عن هذه التقنية أن ينظر إلى هذا الموقع.

asm تخبر المترجم أن ما سيأتي لاحقا بين قوسين وصف لتعليمة بالأسمبلي، volatile استعملت هنا لمنع المترجم من إدخال أي تحسين (optimization) قد يؤدي إلى إزاحة هذه التعليمة من الكود النهائي. داخل القوسين نبدأ التعليمة inb تعني ببساطة in byte أي قراءة بايت واحد ووضعه في المسجل 0% انطلاقا من البوابة الفرعية 1%، الرمزان 0% و 1% يعنيان انه بدل إعطاء قيم جاهزة كما في تعليمة أسمبلر عادية، سنستعمل متغيران، بعد التعليمة يجب تحديد القيمة المرجعة output value لهذه التعليمة، :
"=a"(rv) تستعمل لتعويض 0% و معناها أنه يجب وضع قيمة المسجل al (أو ax أو eax حسب حجم التعليمة) في المتغير rv. "dN (port)" تعني أنه إذا كانت القيمة port ثابتة (مثلا 55) فيجب وضعها مباشرة مكان 1% أما إذا كانت متغيرة فيجب وضعها قبلا في المسجل edx ثم تعويض 1% ب edx. باختصار الدالة inb(port) تقوم بقراءة بايت من البوابة الفرعية المحددة بالرقم port و إرجاعه لنا.
الدالة outb تعني out byte أي قراءة بايت واحد من البوابة الفرعية 0% و وضعه في 1%. نفس الشرح السابق ينطبق على الرموز الأخرى الموجودة داخل القوسين. في النهاية الدالة outb(port, data) تقوم يكتابة بايت قيمته data في البوابة الفرعية port.

هاتان الدالتان ضروريتان لأننا سنحتاج لقراءة البوابات الفرعية من شفرة C++. وتم تعريفهما ك inline حتى يتم إقحامهما مباشرة مثل ال macros في مواقع الشفرة التي تناديهما.

التعريفات المتبقية هي لثلاث دوال يتم تنجيزهما في الملف main.cpp.

الدالة memcpy تقوم بنقل بايتات بعدد count من العنوان src إلى العنوان dest.

```
BYTE *memcpy(BYTE *dest, const BYTE *src, int count)
{
    int i;
    for(i=0; i<count; i++)
       dest[i] = src[i];
    return dest;
}
```

الدالة memset تقوم بكتابة البايت val بعدد مرات count انطلاقا من العنوان dest.

```
BYTE *memset(BYTE *dest, BYTE val, int count)
{
    int i;
    for(i=0; i<count; i++)
```

```
        dest[i] = val;
    return dest;
}
```

الدالة *intToString* تقوم بتحويل رقم إلى سلسلة حروف بالترميز العشري أو السداعشري. التعليق مرفق مع نص الشفرة.

كتابة رقم على شكل حروف: إذا كان base يساوي d أو u فإن الترميز العشري هو المستخدم، أما إذ كان يساوي 'x' فإن الترميز السداعشري هو المستخدم.

```
 void intToString (char *buf, char base, int number)
{

        const char digits[] = "0123456789abcdef";


        char *p = buf;
        unsigned long uns = number;
        int divisor = 10;
```

إذا كنا في الترميز العشري و number عدد سلبي على سلسلة الحروف أن تبدأ ب '-'

```
        if (base == 'd' && d < 0) {
            *p = '-';
            p++;    buf++;
            uns = -number;
        }
```

إذا كنا في الترميز السداعشري على سلسلة الحروف أن تبدأ ب '0x'

```
       else if (base == 'x') {
            divisor = 16;
            *p = '0';
            *(p+1) = 'x';
            p +=2;
            buf +=2;
        }
```

الترميزات المتحملة فقط هي u أو d أو x

```
        else if(base != 'u')  {
            *buf=0;
            return;
        }
```

لتحويل العدد نقوم باستخراج أرقامه واحدا بواحد بدءا من اليمين، uns%divisor تعطينا أقصى رقم في اليمين، بينما uns/=divisor تزيح الرقم المستخرج من العدد. مثلا بالعدد 12987 في الترميز العشري العملية الأولى تعطينا 7 بينما العملية الثانية تحول العدد إلى 1298 فنقوم بتكرار العملية على جميع الحروف

```
        do {
            long remainder = uns % divisor;
```

نستخرج رمز الرقم من الجدول المعرف في بداية الدالة

```
            *p = digit[remainder];
            *p++;
        }  while (uns /= divisor);
```

نضيف 0 علامة على نهاية السلسلة

```
       *p = 0;
```

الحلقة السابقة أعطتنا تمثيل العدد لكن بالمقلوب في مثالنا "78921"، لإعادة ترتيبه نقوم ببساطة بتبديل أمكنة الحروف من الأطراف، في مثالنا تبديل أمكنة 1 و 7 (تعطينا "18927") ثم 8 و 2 (تعطينا "12987")

```
        char *head, *tail;
        head = buf;
        tail = p - 1;
```

```
        while (head < tail) {
            char tmp = *head;
            *head = *tail;
            *tail = tmp;
            head++;
            tail--;
        }
}
```

الدالة ()GDTSetup تقوم بتثبيت ج.م.ع ، تنجيزها موجود في الملف gdt.cpp كما سنرى لاحقا.

الفئة The class Video:

سنقوم بشرح كيفية تنجيز فئة للتعامل مع الشاشة النصية، الملف video.h يحتوي على التعريف فيما الملف video.cpp يقوم بتنجيز الفئة

فيما يلي محتوى الملف video.h

```
#ifndef VIDEO_H_
#define VIDEO_H_

#include<system.h>

class Video
{
private:
  // cursor's position
  int _x, _y;

  // current attribute
  BYTE _attribute;

  // pointer to video memory
  BYTE *mem;

  // screen width and height
  const int width, height;

public:
  Video();

  // clears the screen
  void clear();

  //puts a character at the current location
  // c The character's ASCII code
  void put(char c);

  // A simple printf function
  void printf (const char *format, ...);

  // moves the cursor to the given position
  // x The cursor's new x coordinate
  // y The cursor's new y coordinate
  void moveTo(int x, int y);

  // update the cursor's position in screen to
```

```
  // reflect the actual (x,y) position
  void updateCursor();

  // scrolls the screen by the given vertical amount
  // dy The numbers of lines to scroll
  void scroll(int dy);

  // sets the attribute for the next writes
  // attribute The character's attribute
  void attribute(BYTE attribute);

  // return the current attribute
  BYTE attribute() const;

  // return the cursor's current horizontal position
  int x() const;

  // the cursor's current vertical position
  int y() const;
}
;

#endif /*VIDEO_H_*/
```

كما ترى، الملف يقوم بتعريف الفئة Video : هذه الأخيرة تحتوي على خواص (private members) لتعريف إحداثيات (_x, _y) موقع الكتابة، اللون (_attribute) المستعمل للكتابة و الخلفية، مؤشر (mem) على ذاكرة بطاقة الفيديو عند الوضع النصي، و طول وعرض (width, height) الشاشة بالحروف.

بعد ذلك، الفقرة الموالية تقوم بتعريف عوام (public members) الفئة:
Video() هو مصنع الفئة (Constructor).
الطرق (methods) المتبعة للتعامل مع الشاشة:
clear() لمحو الشاشة، **put(char c)** لكتابة حرف واحد في الموقع الحالي،
void printf(const char *format,...);تنجيز مبسط ل printf.
moveTo(int x,int y) لتغيير موقع الكتابة و مؤشر الشاشة Cursor.
moveBy(int dx, int dy) لتغيير الموقع نسبة إلى الموقع الحالي.
scroll(int dy) لتمرير السطور من أسفل إلى أعلى.
باقي الطرق هي فقط للولوج إلى خاصيات الفئة attribute قراءة و كتابة، و x,y قراءة فقط.

و الآن لنمر إلى كيفية تنجيز الطرق أعلاه

```
Video::Video()
  : _x(0),
    _y(0),
    _attribute(0x07),
    mem(reinterpret_cast<BYTE*>(0xb8000)),
    width(80),
    height(25)
{}
```

في المصنع Video() نقوم بتحميل خاصيات الفئة بالقيم الابتدائية: الإحداثيات x,y يبدآن بالقيمة 0، أي في أول موقع على الشاشة و هو أعلى موقع في أقصى اليسار بالنسبة للغات اللاتينية. الخاصية _attribute حملت بالقيمة 7 مما يعني لون ابيض غير كثيف للكتابة و أسود للخلفية. mem مجهز بالقيمة 0xB8000 أي يؤشر على بداية موقع

الشاشة في الذاكرة. width, height يجهزان تباعا ب 25 و 80 وهما طول و عرض الشاشة النصية.

فيما يلي نص الطريقة put(char c) مع التعليق

```
الطريقة put(char c) تقوم بكتابة حرف على الشاشة في الموقع الحالي باستعمال الخاصية _attribute

void Video::put(char c)
{
    في البداية اختبار بسيط لقيمة الحرف c لمعرفة ما إذا كان يحتوي على حرف من حروف التحكم – أي
    الحروف التي لا تطبع على الشاشة بل تحتوي على أمر مثل الرجوع إلى السطر –

    في حال مصادفة الحرف 0x8 و المطابق لحرف الرجوع إلى الوراء backspace character فإننا نقوم
    بتحريك x خطوة واحدة إلى الوراء شرط ألا نكون في أول السطر

    if(c == 0x08)
    {
        if(_x != 0) _x--;
    }

    في حال '\r'، نقوم بالتحرك إلى بداية السطر

    else if(c == '\r')
    {
        _x= 0;
    }

    في حال '\n'، نقوم بالتحرك إلى بداية السطر الموالي

    else if(c == '\n')
    {
        _x= 0;
        _y++;
    }

    في حال مصادفة حرف طباعة عادي (حروف الطباعة تبدأ من الحرف ' ') نقوم بكتابة الحرف في الموقع
    الحالي (x,y) و الذي يطابق في الذاكرة البعد 2*(y*80+x) انطلاقا من بداية موقع الشاشة mem

    else if(c >= ' ')
    {
        int index = (_y*width + _x) * 2;

        // puts the character
        mem[index] = c;
        mem[index+1] = _attribute;

        بعد كتابة الحرف نقوم بالتحرك خطوة إلى الأمام
        _x++;
    }

    الآن علينا اختبار قيمة الإحداثي x و تصحيحه في حالة ما إذا تجاوز عرض الشاشة

    if(_x >= width)
    {
        _x= 0;
        _y++;
    }

    بعد تصحيح x نقوم باختبار قيمة y و تصحيحها عبر تمرير السطور من أسفل إلى أعلى في حالة تجاوز
    طول y لطول الشاشة
```

```
    if(_y >= height)
       scroll(_y-height+1);

الآن يجب تحريك مؤشر الكتابة Cursor إلى الموقع الحالي

    updateCursor();
}
```

الملف يحتوي أيضا على تنجيز مبسط للدالة printf

```
الدالة printf تأخذ عددا غير معروف من العوامل arguments، عدد هذه الأخيرة يحدد بعدد المغيرات
modifiers التي تبدأ ب % في داخل سلسلة الحروف format. مثلا إذا كانت format تساوي "a = %d
and b = %x" لدينا مغيرين d% و x% مما يعني أن الدالة ستأخذ عاملين إضافيين على format. الدالة
المبسطة أسفله تقبل فقط المغيرات التالية
d% يعني تمثيل العامل المطابق كعدد عشري موجب أو سالب
u% يعني تمثيل العامل المطابق كعدد عشري موجب فقط unsigned
x% يعني تمثيل العامل المطابق كعدد سداعشري
s% يعني تمثيل العامل المطابق كمؤشر على سلسلة حروف
لطباعة الحرف % نكتبه مرتين %%

void Video::printf (const char *format, ...)
{
العوامل الإضافية تأتي في الكومة بعد format مباشرة و لاستخراجها نقوم باعتبارها كجدول تشكل
arg = {format, arg1, arg2, ...) format أول عناصره

        const char **arg = reinterpret_cast<const char **> (&format);

        char c;
        int num;
        char buf[20];

نـقـوم بزيادة arg ليؤشر على العامل الذي بعد format
        arg++;

فيما يلي نقوم بمسح السلسلة format حرفا بحرف عند مصادفة حرف عادي نطبعه على الفور وفي حال
ملاقاة % فهذا يعني أننا أمام مغير
        while ((c = *format++) != 0)
        {
في حال مصادفة مغير نقوم أولا باستخراج الحرف الموالي
           if (c == '%') {
               const char *p;
               format++;
               c = *format;
ثم نقوم باختبار قيمته
               switch (c) {
في حال مصادفة % مرة أخرى نقوم بطباعته
                     case '%':
                        put('%');
                        break;
في حال مصادفة أحد المغيرات الرقمية d, u, x نقوم باستخراج قيمته من الجدول arg و من ثم تحويلها
إلى سلسلة حروف بالترميز المطلوب عبر نداء الدالة intToString و التي رأيناها سابقا. بعد ذلك نقوم
بطباعة سلسلة الحروف الناتجة عن هذا النداء

                     case 'd':
                     case 'u':
                     case 'x':
                         num = *(reinterpret_cast<int *> (arg));
                         arg++;
                         intToString (buf, c, num);
                         for (p = buf; *p != 0; p++)
                             put(*p);
```

```
                        break;
في حال المغير s نستخرج العامل المطابق من الكومة و نستخدمه كمؤشر على سلسلة الحروف المراد طبعها
                    case 's':
                        for (p = *arg; *p != 0; p++)
                           put(*p);
                        arg++;
                        break;
في حالة مغير غير معروف نقوم بطباعة العامل المكافئ كما هو
                    default:
                        num = *(reinterpret_cast<int *> (arg));
                        arg++;
                        put(num);
                        break;
                }
            } else {
حرف عادي و ليس مغيرا، نطبعه مباشرة على الشاشة
                    put(c);
            }
        }
}
```

لتحريك مؤشر الكتابة نستعمل الطريقة updateCursor()

```
void Video::updateCursor()
{

  // calculate the linear location
  WORD loc = (_y * width) + _x;

  // first we write the low byte of the new cursor's position
  outb(0x3D4, 0x0F);
  outb(0x3D5,  loc & 0xFF);
  // the we write the high byte of the new cursor's position
  outb(0x3D4, 0x0E);
  outb(0x3D5,   (loc>>8) &0xFF);
}
```

الشفرة أعلاه تحتاج إلى بعض الشرح، عند التعامل مع الشاشة النصية، ومن أجل تحديد موقع مؤشر الكتابة على الشاشة الرئيسية علينا أن نزود بطاقة الفيديو – ليس بالعنوان الحقيقي على شاكلة (x,y) – بل بالعنوان الخطي في ذاكرة الشاشة النصية و هو كلمة من 16 بيت.
لتغيير الموقع علينا برمجة مسجلات ال CRT Controller أو CRTC. هذا الأخير يتوفر على مسجلين: النوع الأول يدعى مسجل التحكم و النوع الثاني مسجل البيانات. تتم برمجة CRTC دائما على مرحلتين: في المرحلة الأولى يتم إخباره بالعملية المراد تنفيذها عبر وضع كود في مسجل التحكم، ثم بعد ذلك يتم تحمل مسجل البيانات الخاصة بالعملية في مسجل البيانات.
البوابة الفرعية لمسجل التحكم تكون – مسبقا – في العنوان 0x3D4، بينما بوابة مسجل البيانات تكون – مسبقا – في العنوان 0x3D5.
على ضوء ما سبق ذكره، فتغيير موقع مؤشر الشاشة يتم عبر أربع مراحل:
1- نحمل مسجل التحكم بالقيمة 0xF لإخباره أننا نود تزويده بال 8 بيتات الأضعف من القيمة الجديدة لموقع مؤشر الشاشة (انتبه، الموقع الخطي).

```
outb(0x3D4, 0x0F);
```

2- نقوم بتحميل مسجل البيانات بال8 بيتات الأضعف من قيمة الموقع الجديد

```
outb(0x3D5,  loc & 0xFF);
```

3- نحمل مسجل التحكم بالقيمة 0xE لإخباره أننا نود تزويده بال 8 بيتات الأقوى من القيمة الجديدة لموقع مؤشر الشاشة

```
outb(0x3D4, 0x0E);
```

4- نقوم بتحميل مسجل البيانات بال8 بيتات الأقوى من قيمة الموقع الجديد

```
outb(0x3D5,  (loc>>8) & 0xFF);
```

في الطريقة scroll() نقوم بتمرير السطور إلى أعلى في حال ما إذا تجاوز y السطر الأخير

```
void Video::scroll(int dy)
{
    if(dy > 0)
    {
        // Move the block from the second line to up
        memcpy (mem, mem + (dy * width * 2),
                  (height - dy) * width * 2);

        // blank the bottom lines of the screen
        int loc = (height-dy)*width*2;
        for (int i=0; i < dy*2; i+=2)
        {
          mem[loc+i] = ' ';
          mem[loc+i+1] = _attribute;
        }
  }
}
```

في النص أعلاه نقوم بنقل جميع محتوى الذاكرة انطلاقا من آخر سطر نريد نقله إلى السطر الأول. بعد ذلك نقوم بمحو محتوى السطور التي تبقت خاوية بعد عملية التمرير عبر طباعة الحرف ' '.

```
void Video::clear()
{
  for (int x=0; x<width; ++x)
  {
    for (int y=0; y<height; ++y)
    {
      put(' ');
    }
  }

  _x = 0;
  _y = 0;
  updateCursor();
}
```

الطريقة clear() تقوم بمحو كل محتوى الشاشة عبر طباعة الحرف ' ' في كل المواقع، ثم تغير الموقع الحالي إلى أول حرف.

تثبت جدول الموصفات العام

عندما يقوم كروب بتحميل النواة، يكون قد مر بالمعالج قبلا إلى الوضع المحمي وثبت جدولا للموصفات للعام، لكننا لن نعتمد في نواتنا على هذا الجدول و من الأفضل أن نقوم بتثبيت جدولنا الخاص.

الملف gdt.cpp يحتوي الشفرة التي تقوم بتثبيت ج.م.ع الجديد:

```
#include <system.h>
```

في البداية نقوم بتعريف بنية مطابقة لبيانات نموذج الموصف و الذي رأيناه في الدرس السابق، من الضروري استعمال التعريف الخاص __attribute__((packed)) حتى يكون حجم البنية المنتجة من طرف المترجم مطابقا لحجم البنية المعرفة (= منع تمديد حجم البنية بغرض تحسين الأداء)

```
struct GDTEntry
{
    WORD limit_1;               // segment limit bits 0->15
    WORD base_1;                // segment base address bits 0->15
    BYTE base_2;            // segment base address bits 16->23

    BYTE type:4;            // segment type
    BYTE system:1;          // descriptor type
    BYTE dpl:2;             // descriptor privilige level
    BYTE present:1;         // segment present in memory

    BYTE limit_2:4;             // segment limit bits 16->19
    BYTE avl:1;             // available bit
    BYTE zero:1;            // zero field for 32 bit architecture
    BYTE db:1;              // D/B field
    BYTE granularity:1;     // Granularity field

    BYTE base_3;                // segment base address bits 24->31
} __attribute__((packed)); /* this attribute ensures the structure
                              generated by the compiler will be exactly
                              the same size as the above (ie no optimization) */
```

تعريف البنية التي سيتم تحميلها إلى المسجل GDTR

```
struct GDTR {
    WORD limit;
    DWORD base;
} __attribute__((packed));
```

تعريف جدول موصفات من عشرة مداخل

```
// Declare a GDT with 10 entries, and the GDT pointer
#define MAX_ENTRIES 10
GDTEntry gdt[MAX_ENTRIES];
GDTR gptr;
```

الدالة setGDTEntry تقوم بملء المدخل رقم num في ج.م.ع. و تكوين موصف شفرة أو بيانات حسب قيمة isCode، الموصف يتوفر على أفضلية مساوية ل dpl

```
void setGDTEntry(int num,  bool isCode, int dpl)
{       if(num>= MAX_ENTRIES)
                return;
    // Setup the descriptor base address: 0
    gdt[num].base_1 = 0;
    gdt[num].base_2 = 0;
    gdt[num].base_3 = 0;

    // Setup the descriptor limits : 0xFFFFF (*4 ko see granularity flag)
```

```
    gdt[num].limit_1 = 0xFFFF;
    gdt[num].limit_2 = 0xF;

    // setup flags

    gdt[num].type = (isCode)?0xb:0x3);
    gdt[num].system = 1;    // decriptor type = code/data
    gdt[num].dpl = (dpl & 3); // same as (dpl modulo 3) to ensure dpl always < 3
    gdt[num].present = 1;

    gdt[num].avl = 0;
    gdt[num].db = 1;  // D/B: 32 bit segment type
    gdt[num].zero = 0;
    gdt[num].granularity = 1;
}
```

الدالة GDTSetup تقوم بالبناء الفعلي للجدول بالاستعانة بالبنيات و الدالة السابقة

```
void GDTSetup()
{
```

في نقوم بإعداد بيانات البنية التي ستحمل في مسجل GDTR

```
    /* Setup the GDT pointer and limit */
    gptr.limit = (sizeof(GDTEntry) * MAX_ENTRIES)-1 ;
    gptr.base = (DWORD)&gdt;
```

بناء المدخل الأول: كله 0 تبعا لكتاب الأنتل

```
    //First gdt entry must be null
    BYTE *entry = reinterpret_cast<BYTE*>(gdt);
    memset(entry, 0, sizeof(GDTEntry));
```

بناء المدخل الثاني: قسم شفرة + أفضلية قصوى = 0

```
    // setup a code segment with dpl=0
    setGDTEntry(1, true, 0);
```

بناء المدخل الثاني: قسم بيانات + أفضلية قصوى = 0

```
    // setup a data segment with dpl=0
    setGDTEntry(2, false, 0);
```

الآن علينا تحميل gptr إلى مسجل GDTR، من أجل ذلك نستعمل inline assmbly، النحو المستعمل هنا مختلف قليلا لأننا نستعمل الأسمبلر الخاص ب gcc، حيث ترتيب المعاملات مقلوب.

lgdt %0: تقوم بتحميل 0%0 و هو كناية عن المتغير رقم 0 و الذي يتم تحديد قيمته بالتعريف الموجود في آخر التعليمة ((gptr) "=m")، نقوم بتحميله في المسجل GDTR، بعد ذلك نقوم بقفزة طويلة ljmp للتعليمة الموالية بغرض تحديث محدد قسم البيانات في cs. ثم أخيرا نحمل مسجلات الأقسام الأخرى بالمحدد الجديد لقسم البيانات و قسم الكومة

```
    //load the new GDT into GDTR and then update the segment registers
    asm volatile ("lgdt %0          \n\
                   ljmp $8,$1f      \n\
                   1:               \n\
                   movw $16, %%ax    \n\
                   movw %%ax,  %%ss \n\
                   movw %%ax,  %%ds \n\
                   movw %%ax,  %%es \n\
                   movw %%ax,  %%fs \n\
                   movw %%ax,  %%gs"
                  :: "=m"(gptr));
}
```

لدينا الآن جميع الأدوات المطلوبة ولم يتبقى لنا سوى الشفرة التي سينفذها برنامج النواة عند الانطلاق: الدالة kmain – إذا كانت ذاكرتك جيدة فأنت تعرف حتما أنها نفس الدالة التي يتم النداء عليها في الملف multiboot.s.

في هذا الدرس لن نقوم بطباعة مجرد جملة ترحيب على الشاشة. سنستعمل المعلومات التي يمدها لنا كروب عبر بنية *multiboot_info* التي رأينا سابقا والدالة *printf* المبسطة لطباعة المعلومات على الشاشة النصية.
فيما يلي نص شفرة *kmain*

```
extern "C" void kmain(DWORD magic, multiboot_info mbi)
{
	GDTSetup();
	Video v;
	char *devices[] = { "floppy A", "hard disk"};
	v.clear();
إذا كانت "كلمة السر" غير صحيحة فالمحمل ليس هو كروب
	 if(magic != MULTIBOOT_MAGIC) {
		v.printf("Assalamou Alaikoum but not from grub\n");
		v.printf("Invalid magic number %x\n", magic);
	 } else {
		v.printf("Assalamou Alaikoum from grub\n");
	 }

إذا كان البيت 0 من flags يساوي 1 إذن البنية mbi تحتوي على المعلومات الخاصة بالذاكرة
	if (mbi->flags & 1) {
	   v.printf ("Lower memory = %uKB\n",
		     (unsigned) mbi->mem_lower);
	    v.printf ("Upper memory = %uKB\n",
		     (unsigned) mbi->mem_upper);
	}
إذا كان البيت 1 من flags يساوي 1 إذن البنية mbi تحتوي على المعلومات الخاصة بدعامة الانطلاق
	char bootdvc = ((unsigned) mbi->boot_device >> 24) & 0xf;
	char *boot = (bootdvc)?devices[1]:devices[0];
	if (mbi->flags & 2)
	    v.printf ("Boot device = %s\n", boot);
	while(1);
}
```

3- تثبيت النواة و القرص

1- إنتاج الملف التنفيذي للنواة، الملفات التي نتوفر عليها:
multiboot.s: ملف أسمبلي
system.h: ملف رأس يحتوي على تعريف الدوال في main.cpp و gdt.cpp
video.h: ملف رأس تعريف الفئة Video
video.cpp: ملف تنجيز الفئة Video
gdt.cpp: ملف شفرة تثبيت جدول الموصفات العام GDT
main.cpp: الملف الأصلي يحتوي على الدالةkmain التي تشكل المدخل الفعلي للنواة و كذا بعض الدوال الأخرى المستعملة في الملفات الأخرى.
link.lds: ملف السكريبت الخاص بمحرر الروابط ld.

لأجل هذا الغرض سنستعمل ملف makefile، بالنسبة لمستعملي cygwin تحت بيئة ويندوز فالملف كما يلي

```
CPPFLAGS = -c -Wall -fno-builtin -nostdlib -nostdinc -ffreestanding

.PHONY:  clean

OBJS = multiboot.o main.o video.o gdt.o


all: kernel.elf
```

```
kernel.elf: $(OBJS) link.lds
   ld -T link.lds $(OBJS) -o kernel.tmp
   objcopy -O elf32-i386 kernel.tmp kernel.elf
   $(RM) kernel.tmp

clean:
   $(RM) $(OBJS)


%.o: %.cpp
   g++ $(CPPFLAGS) $< -o $@

video.cpp: system.h video.h
main.cpp: system.h video.h
gdt.cpp: system.h


multiboot.o: multiboot.s
nasm -f elf -DLEADING_USCORE $< -o $@
```

ملحوظة مهمة: التعليمات المسطرة بالخط الأحمر للاستعمال فقط إذا كان المترجم الذي تستعمله هو cygwin. بالنسبة للتعليمة

```
objcopy -O elf32-i386 kernel.tmp kernel.elf
```

فنخن نستعملها فقط لأن cygwin غير قادر على إنتاج ملف تنفيذي من نوع elf مباشرة، لذلك فنحن نستعمل البرنامج objcopy لتحويل الملف المنتج في ويندوز إلى ملف elf. باختصار إذا كنت في بيئة لينوكس فأنت لا تحتاج لهذه التعليمة ويمكنك محوها. لاحظ أنه في هذه الحالة عليك تغيير اسم الملف المنتج في التعليمة التي فوق من kernel.tmp إلى kernel.elf

```
ld -T link.lds $(OBJS) -o kernel.elf
```

الاختيار -DLEADING_USCORE هو فقط لتعريف هذا الاسم في ملف الأسمبلي multiboot.s. تذكر التعليمة %ifdef LEADING_USCORE التي تقوم باختبار ما إذا كان هذا الاسم معرفا لتحديد الرمز الذي ينبغي استعماله لمناداة الدالة kmain. إذا كنت في بيئة لينوكس على الأرجح لن تحتاج إلى هذه التعليمة و عليك بمحوها.
في جميع الأحوال إذا كنت لا تعرف ما هو تصرف مترجمك، جرب الترجمة بدون هذا الاختيار، و في حالة ما إذا واجهك خطأ مثل

```
undefined reference to `kmain'
```

فعليك في هذه الحالة إضافة هذا الاختيار إلى سطر أوامر nasm.

الآن لدينا الملف التنفيذي kernel.elf و ما علينا سوى تحميله بكروب. أولا علينا صناعة قرص انطلاق كروب، الحل السهل هو استعمال صورة قرص جاهز بكروب (ستجد صورة القرص مرفقة مع الدرس أو يمكنك تحميله من الموقع التالي {صورة كروب}) و من ثم كتابته على القرص المرن بالتعليمة dd. تحت لينوكس

```
dd if=image_file of=/dev/fd0 conv=notrunc
```

في بيئة ويندوز

```
ntrawrite -f image_file -d A
```

حيث image_file هو اسم صورة القرص الذي تم تحميله.

الآن و بعد تكوين قرص الانطلاق علينا تثبيت ملفنا التنفيذي عليه. انقل الملف kernel.elf في أصل القرص root directory (هذا يكافئ A:\ في ويندوز و / في لينوكس). بعد ذلك افتح الملف menu.cfg في المجلد boot بمحرر نصوص عادي و اكتب السطور التالية

```
default 0

#Entry 0:
title Arabian OS
root (fd0)
kernel /kernel.elf
```

بعد حفظ الملف أصبح الآن لدينا قرص مثبت و جاهز للانطلاق مع نواتنا المطورة في هذا الدرس. ما عليك سوى الإقلاع بهذا القرص على حاسوب حقيقي أو برنامج لمحاكاة الحاسب مثل bochs. في البدء ستظهر لك في الشاشة قائمة اختيار للنظام الذي تود الاقلاع منه، اختر Arabian OS ثم أكد الاختيار بضغط entry. عندها ستطبع في الشاشة النصية جملة من نوع

```
Assalamou Alaikoum from grub
Lower memory = 639KB
Upper memory = 31744KB
Boot device = floppy A
```

و طبعا حجم الذاكرة سيكون مختلفا حسب الجهاز.

خاتمة

في هذا الدرس رأينا كيف يمكن استعمال لغة من أعلى مستوى و هي C++ لتطوير برنامج النواة. وقمنا على هذا الأساس بتطوير تعاملنا مع الشاشة النصية عبر الفئة Video و بخاصة الطريقة printf. مما مكننا من طباعة المعلومات التي زودنا بها كروب بسهولة.

في الدرس القادم إن شاء الله سنتقدم خطوة أخرى إلى الأمام و سنتحدث عن كيفية تدبير الانقطاعات في الوضع المحمي.

إل اللقاء.