

بسم الله الرحمن الرحيم

الدرس الرابع: "تدبير الذاكرة الحقيقية"

◀ نبذه عن الذاكرة الحقيقية ▶

الذاكرة الحقيقية هي تلك الذاكرة المثبتة علي اللوحة الام ومنها انواع كثيرة منها ذاكرة ذات حجم ٥١٢ ميجا ومنها ٢٥٦ إلخ.

وتدبير الذاكرة أي تخصيص مساحات لبرنامج معين بحيث لا يتم استعمال هذه المساحة من أكثر من برنامج مثال:

لنفترض ان لدينا ذاكرة حجمها ١٠ بايت تبدأ عند العنوان 0x00 وتنتهي عند العنوان 0x10 وعندنا برنامجين يعملان لنفرض ان احدهما طلب مساحة من الذاكرة حجمها ١٠ بيتات فقام المدير او المخصص ب إرسال بدايه المساحة إليه (= 0x00) وقام البرنامج الثاني بطلب مساحة ما ولم يعد هناك مساحة اخري فهناك ٣ حلول

- ١- إما ان ينتقل إلي الذاكرة الوهميه او الافتراضيه (ليس موضوعنا)
- ٢- إم ان يجمد طلبات البرنامج وينتظر حتي ينتهي البرنامج الاخر من عمله ثم يعطي هذه المساحة له
- ٣- وهناك حل اخير ولكن غير مرغوب به وهو اعطاء نفس العنوان (0x00) للبرنامج وبهذا نكون *خلطنا بين معلومات البرنامجين

ومن هنا نستنتج ان المدير هو برنامج يعيد عناوين اماكن ما من الذاكرة علي احجام معينه بشرط ----- ان يأتي بالعنوان الخاص بالمساحة باسرع فتره ممكنه وان تكون هذه المساحة غير مستخدمه من برنامج اخر

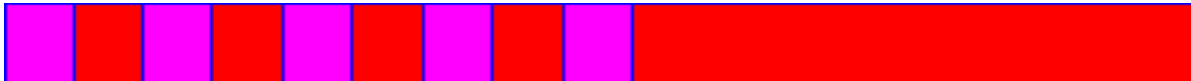
سياسات تدبير الذاكرة

لابد وانكم لاحظتم كلمه سياسات بدل من تقنيات لماذا؟
لانه لا يوجد طريقه معينه نقوم عن طريقها بتدبير الذاكرة أي نظام لتدبير الذاكرة ينفع بشرط تحقيق الشرطين السابقين
لنتعرف علي بعض المفاهيم قبل البدء

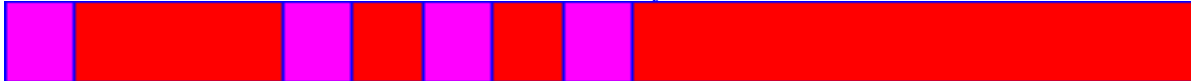
لنتخيل اننا نملك مساحة من الذاكرة قدرها ٦٤ بايت



ولنفرض ان هناك ٥ طلبات لذاكرة حجمها ٨ بايت



ولننظر ماذا سيحدث بعد تحرير احد القطاعات ونقل الثاني



الان هناك برنامج طلب ٦ بيتات سوف يقوم المدير بالبحث سوف يجد ٨ بايت غير مستعملين سوف ولننظر ماذا سيحدث



قارن هذه الصورة بالصورة رقم ٢ من اعلي سوف تجد القطاع الثاني في الصورة الاخير اصغر

ولنفرض ان بعد الطلب الاخير طلب اخر بحجم ٨ بايت سوف يقوم المدبر بالبحث وسيقابل او مكان حر ولكنه بحجم ٢ بايت ولكنه لا ينفع لذا سوف يستمر في البحث حتي يجد قطاع اخر بحجم ٨ بايت وهذا سوف يزيد من وقت البحث

الانقسام :عاده عندما يبدأ المخصص عمله من تخصيص وتحرير للمساحات او القطاعات ينتج عنها مساحات قليلة تزيد من وقت البحث وهذا ينتج مساحات فراغه متفرقه علي كامل الذاكره وهذا ما يسمى بالانقسام .

الانقسام نوعان الاول داخلي وخارجي

الداخلي : هو عكس ماكننا نقوله تماماً بدل من يقوم بقطع ٦ بتات ثم إعطاءها للبرنامج يقوم بتخصيص ال ٨ بتات كامله وبهذا يضيع ٢ بايت



وهنا ايضاً مشكله فنحن سنضيع الكثير من الذاكره

الخارجي : وهو المثال السابق فنحن سنخصص الحجم المطلوب وسنزيد وقت البحث

الدمج : في حالة وجود اكثر من قطاع حر متجاورين يتم دمجهم معاً ليكونوا قطاعاً واحداً

◀ بعض سياسات التخصيص ▶

الاسم	طريقته	مشكلاته	مميزاته
طريقه اول متسع	يتم البحث من اول القطاعات الحره حتي يجد مساحه بالحجم المطلوب او اكبر ثم يقوم بتخصيصها	انقسام خارجي اكبر بطء في التخصيص	امكانيه دمج
افضل متسع	يقوم بالبحث عن اصغر القطاعات حجماً وفي نفس الوقت يكون كاف لإستعاب الحجم المطلوب	المشكله في هذا المخصص هو عندما تكون حجم الذاكره كبير والانقسامات الخارجيه كبيره فانه سوف يلزمه وقت كبير حتي يجد قطاع فارغاً	تقليل نسبة الانقسام الخارجي
المتسع القادم	بدل البحث من بدايه القطاعات الحره يقوم بالبحث من اخر مكان توقف عنده	نسبه انقسام اكبر من السابقين	سرعه كبيره في البحث
متسلسه فيبرو ناتشي	يقوم بتقسيم القتااعات الحره الي احجام كهذه 2 2 4 6 10 16 26 42 byte ثم إختبار ما إذا كان الحجم المطلوب واحد من هذه المساحات	ماذا إذا كان الحجم المطلوب ١١ سوف يقوم بتخصيص واحده من مساحات ال ١٦ بايت وهنا ٥ بايت ضائعين أي نسبة انقسام داخلي اكبر	
خارطة البتات	لنفرض ان لدينا العناوين التاليه من الذاكره وتحميل قيم معينه 0x100: 1 0x101: 4 0x102: 5 0x103: 8 واردنا ان نقوم بتخصيص مساحه ما فانه يقوم بأخذ احد قيم العناوين السابقه ويختبر بتاتها فإذا كان احد البتات صفراً معناه ان هذا قطاع حر	زياده وقت البحث كلما زادت حجم الذاكره	

وقت بحث قصير	*****	واحد معناه ان هذا قطاع مستخدم هذا المخصص يقوم بتقليد بينه حواسبنا فه يقوم بتقسيم الذاكرة إلى قطاعات من حجم ٤ كيلو بايت وعندما نريد معرفه عنوان ما نقوم بضرب رقم الصفحه(المساحه ذات الحجم ٤ كيلو) * ١٠٩٦	آخر دخول اول خروج او المكس (هذا ما سنستخدمه
--------------	-------	---	--

مفاهيم إضافية:

القائمة هي مساحة من الذاكرة يتم حفظ فيها عناوين المساحات .

شرح تفصيلي للمدير الذي سنستخدمه

إن طريقه آخر دخول او خروج هي طريقه شبيهه بالمكدس فعندما نطلب صفحه(أي
مساحه ذات حجم ٤٠٩٦ بايت = ٤ كيلو) فإنه يقوم بإخراج اول عنوان من القائمه ثم
يعود به وهي تماثل إحدى تعليمات المكس (pop ax) وعندما يقوم البرنامج بتحرير
إحدى الصفحات يقوم بإضافتها إلى اول القائمه(push ax) ولنفرض ان هناك برنامج
طلب مساحه ما فإنه يعيد نفس عنوان الصفحه المحرره منذ زمن قليل
مثال:

لدينا مساحه ١٠ صفحات

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

وقمنا بتخصيص ٨ صفحات

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

ثم قمنا بتحرير الصفحه رقم ٦

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

عند طلب تخصيص لصفحه فإنه سوف يعيد عنوان الصفحه رقم ٦ لان عنوانها قد سجل
في اول القائمه.

الكود

في البدايه كنا نقوم بصنه متغيراتنا ديناميكياً أي عن طريق تعليمات التعريف

Char , Int ,;

ولكن عندما نصنع مؤشر لا نستطيع ان نعطي له عنواناً فهذه مشكله لاننا لانعلم حجم القائمه التي سنستعملها
ولكن حلها جاهز وهو استخدام المساحه التي تلي النواه مباشره لإعطاء عناوين للمؤشرات والسؤال هنا كيف
سنستعملها او كيف سنعرف نهايه النواه ؟

عد إلى ملف الربط وتذكر المتغير end الذي قمنا بتعريفه وقلت اننا سنستعمله فيما بعد والذي

يحدد نهايه النواه.

سنعرف إذن متغيراً kernelEnd لتخزين حدود نهاية النواة في الذاكرة. عندما يبدأ برنامج النواة التنفيذ نقوم بتهيئة القيمة الابتدائية لهذا المتغير

```
extern char end;
kernelEnd = (DWORD) &end;
```

عندما نريد صنع عنصر في الذاكرة ما علينا سوى تخصيص المساحة له انطلاقاً من kernelEnd ثم بعد ذلك نغير هذا الأخير ليشير إلى نهاية الحد الجديد. من أجل ذلك نقوم بتنفيذ دالة bootAlloc

```
void *bootAlloc(DWORD size) {
    DWORD addr = kernelEnd;

    // نقوم بحساب الحد الجديد للنواة
    DWORD newEnd = kernelEnd + size;
    // نتأكد من أننا نتوفر على ما يكفي من الذاكرة
    if(memorySize <= newEnd)
        return 0;

    // ثم نحدد نهاية النواة
    kernelEnd = newEnd;
    return (void *)addr;
}
```

نحن الآن نتوفر على دالة شبيهة ب malloc المستعملة في سي. لكننا نشغل في سي++ وهذه الأخيرة تعتمد بالأساس على المعامل new لصنع العناصر. لحسن الحظ، فسي++ تتيح لنا تعريف معاملنا new الخاص بنا.

```
inline void* operator new(DWORD size) {
    void *tmp = bootAlloc(size);
    return tmp;
}
```

بهذه الطريقة يمكننا استعمال المعامل new بطريقة طبيعية. مثلاً إذا أردنا صنع العناصر التي استعملناها في الدروس السابقة مثل Exception

```
Exception *exc = new Exception();
```

الآن بعد أن حللنا مؤقتاً مشكل تخصيص الذاكرة عند الإقلاع. يمكننا المرور إلى تنجز مخصص الذاكرة الحقيقية.

لتنجز الكومة التي ستخزن صور الصفحات سننشئ فئة جديدة نسميها PFrameStack

```
class PFrameStack
{
    DWORD top;
    DWORD *frames;

public:
    PFrameStack(DWORD nbFrames);

    // نقوم بإرجاع الصفحة التي في رأس الجدول. إذا كان الجدول فارغاً نعيد 0
    inline DWORD pop() {
        return (top) ? (frames[--top]) : 0;
    };

    // إضافة صفحة إلى رأس الجدول و تحديث مؤشر الكومة. index هو رقم الصفحة في الذاكرة
    // عنوان الصفحة = رقم الصفحة x حجم الصفحة (4096 بايت)
    inline void push(DWORD index) {
```

```
inline void push(DWORD index) {
    frames[top++] = index;
};
```

تُعطينا عدد العناصر في الكومة

```
inline DWORD size() {
    return top;
}
```

```
};
```

كما نلاحظ فكل طرق الفئة - باستثناء المصنّع - عرفت بطريقة سطرية inline. هذه التقنية تستعمل لتحسين الأداء حيث تدمج شفرة هذه الطرق مباشرة في المواقع التي يتم نداؤها منها. إذا استعملنا التعريف العادي سنخسر وقتاً أكبر لأننا سنضطر للمرور عبر call كلما تم نداء هذه الطرق.

في المصنّع نقوم بتهيئة الجدول الذي سيحتوي على عناوين صور الصفحات الحرة.

```
PFrameStack::PFrameStack(DWORD nbFrames) {
```

تخصيص المساحة اللازمة لاحتضان عناوين الصفحات = عدد الصفحات * حجم العنوان

```
top = 0;
frames = (DWORD *) bootAlloc(nbFrames*sizeof(DWORD));
}
```

بعد تنجيز الكومة. سننشئ فئة ستشكل المخصص الفعلي الذي ستتعامل معه باقي الفئات. الفئة PhysMManager هي التي ستكون الواجهة المستخدمة لتخصيص الذاكرة الفعلية

(سؤال: لماذا لا نستخدم PFrameStack كواجهة مباشرة بدل المرور عبر PhysMManager؟ الجواب: هكذا يمكننا عزل تنجيز المخصص عن الخارج. فيما بعد يمكننا استعمال بنيات غير الكومة أو حتى عدة مخصصات في آن واحد بدون أن نضطر لتغيير أي شفرة خارج PhysMManager. مبرمجو سي++ و جافا يعرفون جيداً هذه التقنية: الحجب (Encapsulation).

```
class PhysMManager
```

```
{
    PFrameStack *_pfStack;
    DWORD nbFrames;

    DWORD memTop;
    DWORD kernelTop;
```

```
public:
```

هذه الشارة flag تحدد ما إذا كان يلزم ملئ صورة الصفحة ب 0 قبل تخصيصها

```
static const DWORD ZEROED = 1;
```

```
PhysMManager();
```

DWORD allocFrame(DWORD flags);	تخصيص صورة صفحة واحدة
(address = index*4096) index	تقوم بتحرير الصفحة رقم
void freeFrame(DWORD index);	
DWORD freePages();	عدد الصفحات الحرة في الذاكرة
};	

التنفيذ:

extern DWORD kernelEnd;	يشير إلى حدود برنامج النواة
extern DWORD memorySize;	حجم الذاكرة بالبايت
PhysMManager::PhysMManager() {	
memTop = pageBase(memorySize+1);	رقم آخر صفحة في الذاكرة
nbFrames = memTop / PAGE_SIZE;	
_pfStack = new PFrameStack(nbFrames);	صنع مخصص الكومة
kernelTop = pageTop(kernelEnd);	تسطير نهاية النواة على 4096 بايت
for (int i = START_PAGE; i < START_VGA; i+=PAGE_SIZE) {	إضافة صور الصفحات في الذاكرة السفلى إلى القائمة الحرة
_pfStack->push(i>>PAGE_SHIFT);	
}	
for (int i = kernelTop; i < memTop; i+=PAGE_SIZE) {	إضافة صور الصفحات بعد النواة إلى القائمة الحرة
_pfStack->push(i>>PAGE_SHIFT);	
PhysMManager::PhysMManager() {	
memTop = pageBase(memorySize+1);	رقم آخر صفحة في الذاكرة
nbFrames = memTop / PAGE_SIZE;	
_pfStack = new PFrameStack(nbFrames);	صنع مخصص الكومة
kernelTop = pageTop(kernelEnd);	تسطير نهاية النواة على 4096 بايت
for (int i = START_PAGE; i < START_VGA; i+=PAGE_SIZE) {	إضافة صور الصفحات في الذاكرة السفلى إلى القائمة الحرة
_pfStack->push(i>>PAGE_SHIFT);	
}	
for (int i = kernelTop; i < memTop; i+=PAGE_SIZE) {	إضافة صور الصفحات بعد النواة إلى القائمة الحرة
_pfStack->push(i>>PAGE_SHIFT);	

```

    }
}

DWORD PhysMManager::allocFrame(DWORD flags) {
    // استرجاع صورة صفحة من الكومة
    BYTE *addr = (BYTE *)_pfStack->pop();

    // ثم في حالة تم التخصيص (addr != 0) و الشارة ZEROED flag محددة تملئ الصفحة بـ 0
    DWORD val = ((DWORD) addr) * PAGE_SIZE;

    if(addr && (flags & ZEROED)) {
        memset(val, 0, PAGE_SIZE);
    }

    // نعيد عنوان الصفحة = رقم الصفحة * حجم الصفحة
    return val;
}

void PhysMManager::freeFrame(DWORD index) {
    _pfStack->push(index);
}

DWORD PhysMManager::freePages() {
    return _pfStack->size();
}

```

الآن وقد أنجزنا مخصص الذاكرة الحقيقية. يتبقى لنا تجربته و التأكد من أنه يعمل كما ينبغي. برغم بساطته فقد ارتأيت أن أضيف دالة صغيرة لتجريب المخصص. فيما يلي سأقدم الدالة kmain حيث نقوم بصنع العناصر اللازمة باستخدام bootAlloc.

أريد فقط أن ألفت انتباه القارئ الذي تابع معي الدروس منذ البداية وحتى الآن أنني قد قمت ببعض التنظيم في الشفرة: لم أغير أيًا من الدوال التي تم إنجازها في الدروس السابقة بل فقط قُمت بتحويل مكانها داخل فئات جديدة حتى لا تتراكم لدينا مجموعة من الدوال و المتغيرات العامة التي تُعقد قراءة الشفرة. إذا كان القارئ قد قرأ الدروس السابقة فلن يجد مشكلة في قراءة الشفرة بالشكل الجديد. لبعض التوضيح أقدم عرضاً عاماً للبنية الجديدة للشفرة:

قبل المرور إلى شفرة Kmain. سأشرح بإيجاز كيف سنقوم بتجريب المخصص: التجربة تتم على مرحلتين في مرحلة أولى سنطلب تخصيص كل الصفحات الحرة. في كل صفحة مخصصة سنكتب بعض البيانات التي ستشكل "العلامة" على أن هذه الصفحة قد تم تخصيصها. عند تخصيص صفحة جديدة نقوم بالتأكد أن المخصص لم يعطنا صفحة تم تخصيصها مسبقاً و ذلك عبر التأكد من أنها لا تحتوي على العلامة السابق ذكرها. في المرحلة الثانية، نقوم بتحرير جميع الصفحات و نتأكد من أن جميع الصفحات قد حررت من طرف المخصص عبر مقارنة عدد الصفحات الحرة بعد التحرير بعددها قبل التحرير.

و الآن بعد هذا التوضيح ننتقل إلى شفرة الدالة kmain.

```

extern "C" void kmain(DWORD magic, multiboot_info *mbi) {
    extern char end;
    kernelEnd = (DWORD) &end;

    // العضو mem_upper في multiboot_info يعطى حجم "الذاكرة العليا" أي الموجود بعد
    // أول ميغ. الحجم موزن بالكيلوبايت لذلك نضربه في 1024 و نضيف عليه 1 ميغ.
    memorySize = mbi->mem_upper*1024+0x100000;

    video = new Video();
}

```

```

video->clear();

multiboot_info kernel::init
Kernel::init(magic, mbi);
printMemInfo(mbi);

الفتنة Machine تضم الشفرة اللازمة للتعامل مع المعالج
inb, outb, cli, sti ...
struct regs, GDTR, IDTR...
setupGDT, setupIDT

Machine::init();
PIC::init() تقوم بإعداد محكم الانقطاعات
PIC::init();
ثم نقوم بملئ ج.م.م. IDT بمديرات المرحلة الأولى (المكتوبة بالآسبيلي)
initISRS();
Machine::sti();

pageTest ستستعمل كما سنرى لتخزين عناوين صور الصفحات التي سنطلبها من المخصص
أثناء تجربته. بما أننا أثناء التجريب سنقوم بتخصيص كل الصفحات الحرة فإننا نحتاج
لمساحة = عدد الصفحات * حجم العنوان (DWORD)

pageTest = (DWORD*) bootAlloc(
    (memorySize/PAGE_SIZE)*sizeof(DWORD)
);

Exception *exc = new Exception();
for (int i = 0; i < 32; ++i) {
    IntManager::setHandler(i, exc);
}

Timer *t = new Timer();
t->setPhase(20);
IntManager::setHandler(32,t);
PIC::setPICIMask(0xFE);

video->printf("Init Physical Memory Manager");
Kernel::initPhysMManager();
video->printf("    [OK]\n");

هذه الدالة التي تقوم بتجريب المخصص الحقيقي. أنظر أسفله
testPhysMem();

while(1);
}

```

مجرى المخصص الحقيقي يتكون من ثلاث دوال.

الدالة allocateAll تقوم بتخصيص جميع الصفحات الحرة و تخزين عناوينها في الجدول pageTest

```

void allocateAll() {
    DWORD *addr;
    while(1) {
        نقوم بطلب صفحة من المخصص

        addr = (DWORD *)physMManager->allocFrame(0);

        إذا لم تعد هناك صفحة حرة نخرج من الحلقة
    }
}

```

```

        if (!addr)
            break;

        // نأكد من أن الصفحة لم يتم تخصيصها من قبلنا مسبقا
        if (*addr == (DWORD)addr && addr[(PAGE_SIZE-1)/4] == 0x55) {
            video->printf("Error, Page at %u already allocated\n",
                          (DWORD)addr);
            Machine::hlt();
        }

        // نكتب علامة التخصيص في الصفحة
        addr[0] = (DWORD)addr;
        addr[(PAGE_SIZE-1)/4] = 0x55;
        pageTest[nbAllocated] = (DWORD)addr;
        nbAllocated++;
        nbFree--;
        printMemState();
    }
}

```

عكسا، الدالة freeAll تقوم بتحرير كل الصفحات التي تم تخصيصها من قبل allocateAll

```

void freeAll() {
    DWORD *addr;
    while(1) {
        if (!nbAllocated)
            break;

        // نستعيد عنوان الصفحة التالية من الجدول pageTest
        addr = (DWORD *)pageTest[nbAllocated-1];

        // ثم نتأكد من أن الصفحة تحتوي على علامة التخصيص (الخذر واجب ☹)
        if (addr[0] != (DWORD)addr || addr[(PAGE_SIZE-1)/4] != 0x55) {
            video->printf("Error, Page at %u corrupted\n",
                          (DWORD)addr);
            Machine::hlt();
        }

        // نزيل علامة التخصيص ثم نعيد الصفحة إلى المخصص
        addr[0] = 0;
        addr[(PAGE_SIZE-1)/4] = 0;
        physMManager->freeFrame(((DWORD)(addr) >> PAGE_SHIFT));
        nbFree++;
        nbAllocated--;
        printMemState();
    }
}

```

و الآن الدالة الأساسية testPhysMManager

```

void testPhysMManager() {
    nbAllocated = 0;
    physMManager = Kernel::physMManager;
}

```

```
DWORD initialNbFree = nbFree = physMManager->freePages();  
video->printf("testing Physical Memory Manager  \n");
```

نخصص جميع الصفحات

```
allocateAll();
```

نتأكد من أنه تم فعلا تخصيص كل الصفحات

```
if(nbAllocated != initialNbFree) {  
    video->printf("There were %u free pages but\  
                only %u were allocated",  
                physMManager->freePages(), nbAllocated);  
  
    Machine::hlt();  
}
```

ثم نعيد للمخصص جميع الصفحات

```
freeAll();
```

و نتأكد من أنه قد استعاد منا جميع الصفحات و لم يضع شيئا

```
if(initialNbFree != physMManager->freePages()) {  
    video->printf("There were %u allocated pages but\  
                only %u were freed",  
                physMManager->freePages(), nbAllocated);  
  
    Machine::hlt();
```