١ 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.

١.١ 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 .

١.١.١ 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).

٢.١ 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.

١.٢.١ 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 الذاكرة. ويجدر بنا ذكر أن التحويل بين العناوين التخيليه الى الحقيقية يتم عن طريق العتاد بواسطة وحدة ادارة الذاكرة بداخل المعالج (Memory

[١] المقصود أنها برامج تعمل في الحلقة ٣.

Management Unit).وكذلك مهمة حماية الذاكرة والتحكم في الذاكرة Cache وغيرها من الخصائص والتي سيتم الإطلاع عليها في الفصل الثامن – بمشيئة الله–.

٣.١ 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].

١.٣.١ 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.

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

---

[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.

٢.٣.١ 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.

٣.٣.١ 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.

٤.١ 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 المقاطعات 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++ .

١.٤.١ 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.

Example ١.١: Portable Executable Header

```
١
٢ // header information format for PE files
٣
٤ typedef struct _IMAGE_DOS_HEADER {  // DOS .EXE header
٥     unsigned short e_magic;  // Magic number (Should be MZ
٦     unsigned short e_cblp;    // Bytes on last page of file
٧     unsigned short e_cp;      // Pages in file
```

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

```
٨      unsigned short e_crlc;     // Relocations
٩      unsigned short e_cparhdr;  // Size of header in paragraphs
١٠     unsigned short e_minalloc;   // Minimum extra paragraphs needed
١١     unsigned short e_maxalloc;   // Maximum extra paragraphs needed
١٢     unsigned short e_ss;    // Initial (relative) SS value
١٣     unsigned short e_sp;    // Initial SP value
١٤     unsigned short e_csum;   // Checksum
١٥     unsigned short e_ip;     // Initial IP value
١٦     unsigned short e_cs;    // Initial (relative) CS value
١٧     unsigned short e_lfarlc;   // File address of relocation table
١٨     unsigned short e_ovno;    // Overlay number
١٩     unsigned short e_res[4];    // Reserved words
٢٠     unsigned short e_oemid;   // OEM identifier (for e_oeminfo)
٢١     unsigned short e_oeminfo;  // OEM information; e_oemid specific
٢٢     unsigned short e_res2[10];    // Reserved words
٢٣     long   e_lfanew;    // File address of new exe header
٢٤   } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
٢٥
٢٦
٢٧ // Real mode stub program
٢٨
٢٩ typedef struct _IMAGE_FILE_HEADER {
٣٠     unsigned short  Machine;
٣١     unsigned short  NumberOfSections;
٣٢     unsigned long   TimeDateStamp;
٣٣     unsigned long   PointerToSymbolTable;
٣٤     unsigned long   NumberOfSymbols;
٣٥     unsigned short  SizeOfOptionalHeader;
٣٦     unsigned short  Characteristics;
٣٧ } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
٣٨
٣٩ typedef struct _IMAGE_OPTIONAL_HEADER {
٤٠     unsigned short  Magic;
٤١     unsigned char   MajorLinkerVersion;
٤٢     unsigned char   MinorLinkerVersion;
٤٣     unsigned long   SizeOfCode;
٤٤     unsigned long   SizeOfInitializedData;
٤٥     unsigned long   SizeOfUninitializedData;
٤٦     unsigned long   AddressOfEntryPoint;   // offset of kernel_entry
٤٧     unsigned long   BaseOfCode;
٤٨     unsigned long   BaseOfData;
```

```
٤٩     unsigned long   ImageBase;   // Base address of kernel_entry
٥٠     unsigned long   SectionAlignment;
٥١     unsigned long   FileAlignment;
٥٢     unsigned short  MajorOperatingSystemVersion;
٥٣     unsigned short  MinorOperatingSystemVersion;
٥٤     unsigned short  MajorImageVersion;
٥٥     unsigned short  MinorImageVersion;
٥٦     unsigned short  MajorSubsystemVersion;
٥٧     unsigned short  MinorSubsystemVersion;
٥٨     unsigned long   Reserved1;
٥٩     unsigned long   SizeOfImage;
٦٠     unsigned long   SizeOfHeaders;
٦١     unsigned long   CheckSum;
٦٢     unsigned short  Subsystem;
٦٣     unsigned short  DllCharacteristics;
٦٤     unsigned long   SizeOfStackReserve;
٦٥     unsigned long   SizeOfStackCommit;
٦٦     unsigned long   SizeOfHeapReserve;
٦٧     unsigned long   SizeOfHeapCommit;
٦٨     unsigned long   LoaderFlags;
٦٩     unsigned long   NumberOfRvaAndSizes;
٧٠     IMAGE_DATA_DIRECTORY DataDirectory[DIRECTORY_ENTRIES];
٧١ } 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 e lfanew 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).

Example ١.٢: Getting Kernel entry

```
١
٢
٣   mov ebx,[KERNEL_PMODE_BASE+60]
```

```
٤   add ebx,KERNEL_PMODE_BASE      ; ebx = _IMAGE_FILE_HEADER
٥
٦   add ebx,24       ; ebx = _IMAGE_OPTIONAL_HEADER
٧
٨   add ebx,16       ; ebx point to AddressOfEntryPoint
٩
١٠  mov ebp,dword[ebx]   ; epb = AddressOfEntryPoint
١١
١٢  add ebx,12       ; ebx point to ImageBase
١٣
١٤  add ebp,dword[ebx]   ; epb = kernel_entry
١٥
١٦  cli
١٧
١٨  call ebp
```

٢.٤.١ 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.

Example ١.٣: Global new/delete operator

```
١
٢ void* __cdecl ::operator new (unsigned int size){return 0;}
٣ void* __cdecl operator new[] (unsigned int size){return 0;}
٤ void __cdecl ::operator delete (void * p){}
٥ 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.

Example ١.٤: Pure virtual function call handler

```
١
٢ 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.

Example ١.٥: Floating Point Support

```
١
٢
٣ extern "C" long __declspec (naked) _ftol2_sse() {
٤   int a;
٥ #ifdef i386
٦   _asm {
٧     fistp [a]
٨     mov ebx, a
٩   }
١٠ #endif
١١ }
١٢
١٣ extern "C" int _fltused = 1;
```

[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.

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.

Example ١.٦: Object Initializer

```
١
٢ // Function pointer typedef for less typing
٣ typedef void (__cdecl *_PVFV)(void);
٤
٥ // __xc_a points to beginning of initializer table
٦ #pragma data_seg(".CRT$XCA")
٧ _PVFV __xc_a[] = { 0 };
٨
٩ // __xc_z points to end of initializer table
١٠ #pragma data_seg(".CRT$XCZ")
١١ _PVFV __xc_z[] = { 0 };
١٢
١٣ // Select the default data segment again (.data) for the rest of the
      unit
١٤ #pragma data_seg()
١٥
```

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

```
١٦ // Now, move the CRT data into .data section so we can read/write to
       it
١٧ #pragma comment(linker, "/merge:.CRT=.data")
١٨
١٩
٢٠ // initialize all global initializers (ctors, statics, globals, etc
       ..)
٢١ void __cdecl _initterm ( _PVFV * pfbegin, _PVFV * pfend ) {
٢٢
٢٣   //! Go through each initializer
٢٤     while ( pfbegin < pfend )
٢٥     {
٢٦     //! Execute the global initializer
٢٧       if ( *pfbegin != 0 )
٢٨             (**pfbegin) ();
٢٩
٣٠       //! Go to next initializer inside the initializer table
٣١         ++pfbegin;
٣٢     }
٣٣ }
٣٤
٣٥ // execute all constructors and other dynamic initializers
٣٦ void _cdecl init_ctor() {
٣٧
٣٨    _atexit_init();
٣٩    _initterm(__xc_a, __xc_z);
٤٠ }
```

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.

Example ١.٧: Delete Object

```
١
٢ /! function pointer table to global deinitializer table
٣ static _PVFV * pf_atexitlist = 0;
٤
٥ // Maximum entries allowed in table. Change as needed
٦ static unsigned max_atexitlist_entries = 32;
٧
٨ // Current amount of entries in table
٩ static unsigned cur_atexitlist_entries = 0;
١٠
١١ //! initialize the de—initializer function table
١٢ void __cdecl _atexit_init(void) {
١٣
١٤     max_atexitlist_entries = 32;
١٥
١٦   // Warning: Normally, the STDC will dynamically allocate this.
        Because we have no memory manager, just choose
١٧   // a base address that you will never use for now
١٨   pf_atexitlist = (_PVFV *)0x5000;
١٩ }
٢٠
٢١ //! adds a new function entry that is to be called at shutdown
٢٢ int __cdecl atexit(_PVFV fn) {
٢٣
٢٤   //! Insure we have enough free space
٢٥   if (cur_atexitlist_entries>=max_atexitlist_entries)
٢٦     return 1;
٢٧   else {
٢٨
٢٩     //! Add the exit routine
٣٠     *(pf_atexitlist++) = fn;
٣١     cur_atexitlist_entries++;
٣٢   }
٣٣   return 0;
٣٤ }
٣٥
٣٦ //! shutdown the C++ runtime; calls all global de—initializers
٣٧ void _cdecl exit () {
٣٨
٣٩   //! Go through the list, and execute all global exit routines
٤٠   while (cur_atexitlist_entries——) {
```

```
٤١
٤٢        //! execute function
٤٣        (*(--pf_atexitlist)) ();
٤٤    }
٤٥ }
```

٣.٤.١ 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

Example ١.٨: Kernel Entry routine

```
١
٢ extern void _cdecl main ();   // main function.
٣ extern void _cdecl init_ctor();   // init constructor.
٤ extern void _cdecl exit ();     // exit.
٥
٦ void _cdecl kernel_entry ()
٧ {
٨
٩ #ifdef i386
١٠   _asm {
١١     cli
١٢
١٣     mov ax, 10h        // select data descriptor in GDT.
١٤     mov ds, ax
١٥     mov es, ax
١٦     mov fs, ax
١٧     mov gs, ax
١٨     mov ss, ax         // Set up base stack
١٩     mov esp, 0x90000
٢٠     mov ebp, esp       // store current stack pointer
٢١     push ebp
٢٢   }
٢٣ #endif
```

```
٢٤
٢٥   // Execute global constructors
٢٦   init_ctor();
٢٧
٢٨   // Call kernel entry point
٢٩   main();
٣٠
٣١   // Cleanup all dynamic dtors
٣٢   exit();
٣٣
٣٤ #ifdef i386
٣٥   _asm cli
٣٦ #endif
٣٧
٣٨   for(;;);
٣٩ }
```

وتعريف الدالة `main()` حالياً سيكون خاليا.

٥.١ 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.

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

* 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.

٦.١ مكتبة السي القياسية

نظراً 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 وقتاً وجهدا فاننا سنركز على بعض الملفات المستخدمة بكثرة ونترك البقية للتطوير لاحقا.

تعريف NULL

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

Example ١.٩: null.h:Definition of NULL in C and C++

```
١
٢ #ifndef NULL_H
٣ #define NULL_H
٤
٥ #if define (_MSC_VER) && (_MSC_VER > = 1020)
٦ #pargma once
٧ #endif
٨
٩ #ifdef NULL
١٠ #undev NULL
١١ #endif
١٢
١٣ #ifdef __cplusplus
١٤ extern "C"
١٥ {
١٦ #endif
١٧
١٨ /* C++ NULL definition */
```

```
١٩ #define NULL 0
٢٠
٢١ #ifdef __cplusplus
٢٢ }
٢٣ #else
٢٤
٢٥ /* C NULL definition */
٢٦ #define NULL (void*)0
٢٧
٢٨ #endif
٢٩
٣٠ #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).

Example ١.١٠: size_t.h:Definition of size_t in C/C++

```
١
٢ #ifndef SIZE_T_H
٣ #define SIZE_T_H
٤
٥ #ifdef __cplusplus
٦ extern "C"
٧ {
٨ #endif
٩
١٠ /* Stdandard definition of size_t */
١١ typedef unsigned size_t;
١٢
١٣ #ifdef __cplusplus
١٤ }
١٥ #endif
١٦
١٧
١٨ #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 .

Example ١.١١: stdint.h:typedef data type

```
١
٢ #ifndef STDINT_H
٣ #define STDINT_H`
٤
٥ #define __need_wint_t
٦ #define __need_wchar_t
٧
٨ /* Exact-width integer type */
٩ typedef   char         int8_t;
١٠ typedef   unsigned char     uint8_t;
١١ typedef   short        int16_t;
١٢ typedef   unsigned short    uint16_t;
١٣ typedef   int          int32_t;
١٤ typedef   unsigned int    uint32_t;
١٥ typedef   long long      int64_t;
١٦ typedef   unsigned long long  uint64_t;
١٧
١٨
١٩ // to be continue..
٢٠
٢١ #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].

Example ١.١٢: cstdint:C++ typedef data type

```
١
٢ #ifndef CSTDINT_H
```

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

```
٣ #define CSTDINT_H
٤
٥ #include <stdint.h>
٦
٧ #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).

Example ١.١٣: ctype.h:determine character type

```
١
٢ #ifndef CTYPE_H
٣ #define CTYPE_H
٤
٥ #ifdef _MSC_VER
٦ #pragma warning (disable:4244)
٧ #endif
٨
٩ #ifdef __cplusplus
١٠ extern "C"
١١ {
١٢ #endif
١٣
١٤ extern char _ctype[];
١٥
١٦ /* constants */
١٧
١٨ #define CT_UP     0x01  // upper case
١٩ #define CT_LOW    0x02  // lower case
٢٠ #define CT_DIG    0x04  // digit
٢١ #define CT_CTL    0x08  // control
٢٢ #define CT_PUN    0x10  // punctuation
٢٣ #define CT_WHT    0x20  // white space (space,cr,lf,tab).
٢٤ #define CT_HEX    0x40  // hex digit
٢٥ #define CT_SP   0x80  // sapce.
٢٦
```

```
٢٧ /* macros */
٢٨
٢٩ #define isalnum(c)     ( (_ctype+1)[(unsigned)(c)] & (CT_UP|CT_LOW|
       CT_DIG) )
٣٠ #define isalpha(c)     ((_ctype + 1)[(unsigned)(c)] & (CT_UP | CT_LOW)
       )
٣١ #define iscntrl(c)       ((_ctype + 1)[(unsigned)(c)] & (CT_CTL))
٣٢
٣٣
٣٤ // to be continue..
٣٥
٣٦ #ifdef __cplusplus
٣٧ }
٣٨ #endif
٣٩
٤٠ #endif // CTYPE_H
```