OPERATING SYSTEM

Lecture Notes

Dr. Professor, J.M. Khalifeh

قسم المعلوماتية

الوحدة السادسة

النسخة العربية

ملاحظة هامة: النسخة الأساسية هي النسخة الإنكليزية

Unit-6

Memory Management

ملخص

الغرض الرئيسي من نظام الكمبيوتر هو تنفيذ البرامج. يجب أن تكون هذه البرامج، إلى جانب البيانات التي تستخدمها، في الذاكرة الرئيسية جزئيًا على الأقل أثناء التنفيذ. تحتفظ أنظمة الكمبيوتر الحديثة بعدة عمليات في الذاكرة أثناء التنفيذ. ومن أجل ذلك توجد العديد من مخططات إدارة الذاكرة، حيث تعكس الأساليب المختلفة، وتختلف فعالية كل أسلوب باختلاف الحالة. يعتمد اختيار نظام إدارة الذاكرة لنظام ما على العديد من العوامل، أهمها تصميم أجهزة النظام. حيث تتطلب معظم الخوارزميات شكلاً من أشكال دعم الأجهزة.

يمكن تعريف مصطلح الذاكرة على أنها مجموعة من البيانات المستخدمة بتنسيق معين. يتم استخدامها لتخزين التعليمات ومعالجة البيانات. تتكون الذاكرة من مجموعة كبيرة من الكلمات أو البايتات، ولكل منها موقعها الخاص. وبما أن الهدف الأساسي لنظام الكمبيوتر هو تنفيذ البرامج فيجب أن تكون هذه البرامج، إلى جانب المعلومات التي تصل إليها، في الذاكرة الرئيسية أثناء التنفيذ. تقوم وحدة المعالجة المركزية بجلب التعليمات من الذاكرة وفقًا لقيمة عداد البرنامج.

لتحقيق درجة من التعددية في المعالجة والاستخدام السليم للذاكرة، فمن المهم إدارة الذاكرة بشكل جيد، حيث توجد العديد من طرق إدارة الذاكرة، تعكس الأساليب المختلفة، وتعتمد فعالية كل خوارزمية على الحالة المعنية.

أهداف الوحدة

- شرح الفرق بين العنوان المنطقي والعنوان الفعلي.
- شرح دور وحدة إدارة الذاكرة (MMU) في ترجمة العناوين.
- شرح استراتيجيات تخصيص الذاكرة.
- تطبيق الإستراتيجيات الأولى والأفضل والأسوأ لتخصيص الذاكرة بشكل متواصل.
- التمييز بين التجزئة الداخلية والخارجية.
- ترجمة العناوين المنطقية إلى عناوين فعلية في نظام الترحيل.

خلفية

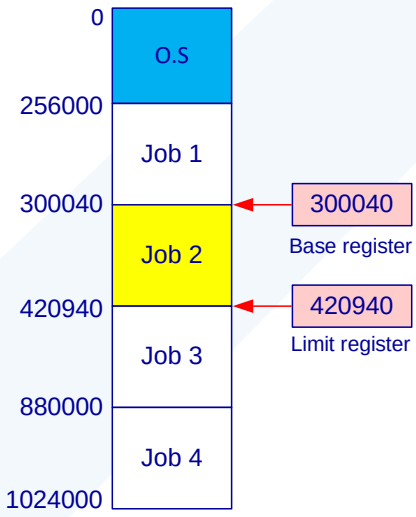
تعتبر الذاكرة أساسية لتشغيل نظام الكمبيوتر الحديث. تتكون الذاكرة من مجموعة كبيرة من البايتات، ولكل منها عنوانها الخاص. تقوم وحدة المعالجة المركزية بجلب التعليمات من الذاكرة وفقًا لقيمة عداد البرنامج. قد تتسبب هذه التعليمات في تحميل إضافي من عناوين ذاكرة معينة وتخزينها فيها.

في دورة تنفيذ التعليمات النموذجية، على سبيل المثال، يتم جلب التعليمات من الذاكرة أولاً. ويتم بعد ذلك فك تشفير التعليمات وقد يحتاج ذلك إلى جلب المعاملات من الذاكرة. وبعد تنفيذ التعليمات على المعاملات، قد يتم تخزين النتائج مرة أخرى في الذاكرة. وفي كل ذلك فإن وحدة الذاكرة ترى فقط دفقًا من عناوين الذاكرة ؛ لا يعرف كيف يتم إنشاؤها (من خلال عداد التعليمات، والفهرسة، والعناوين الحرفية، وما إلى ذلك) أو ما هي (التعليمات أو البيانات). وفقًا لذلك، يمكننا تجاهل كيفية إنشاء البرنامج لعنوان الذاكرة. وسنهتم فقط بتسلسل عناوين الذاكرة التي تم إنشاؤها بواسطة البرنامج قيد التشغيل.

الأجهزة الأساسية

الذاكرة الرئيسية والمسجلات المضمنة في كل نواة معالجة هي التخزين الوحيد للأغراض العامة الذي يمكن لوحدة المعالجة المركزية الوصول إليه مباشرة. هناك تعليمات خاصة بالآلة تأخذ عناوين الذاكرة كوسائط، لكن لا شيء يأخذ عناوين القرص. لذلك، يجب أن تكون أي تعليمات قيد التنفيذ، وأي بيانات يتم استخدامها بواسطة التعليمات، في أحد أجهزة التخزين ذات الوصول المباشر هذه. إذا لم تكن البيانات في الذاكرة، فيجب نقلها هناك قبل أن تتمكن وحدة المعالجة المركزية من العمل عليها.

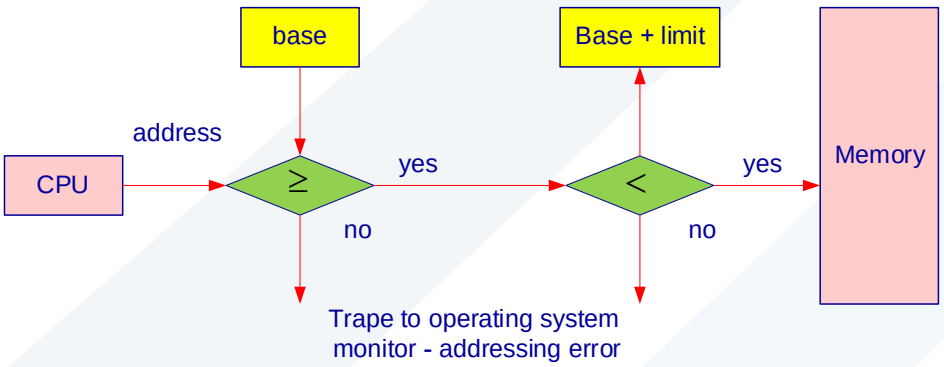
لتشغيل النظام بشكل صحيح، يجب علينا حماية نظام التشغيل من الوصول بواسطة عمليات المستخدم، وكذلك حماية عمليات المستخدم من بعضها البعض. يجب توفير هذه الحماية من خلال الأجهزة، لأن نظام التشغيل لا يتدخل عادة بين وحدة المعالجة المركزية والذاكرة الخاصة بها.



تحمي مساحة الذاكرة المنفصلة لكل عملية العمليات من بعضها البعض وهي أساسية لوجود عمليات متعددة محملة في الذاكرة للتنفيذ المتزامن. لفصل مساحات الذاكرة، نحتاج إلى القدرة على تحديد نطاق العناوين التي قد تصل إليها العملية والتأكد من أن العملية يمكنها الوصول إلى هذه العناوين فقط. ونستطيع توفير هذه الحماية باستخدام مسجلين، مسجل قاعدي ومسجل حدي.

يحتوي السجل القاعدي على أصغر عنوان ذاكرة فعلي صالح؛ يحدد المسجل الحدي حجم النطاق المستخدم من قبل العملية. على سبيل المثال، إذا كان المسجل القاعدي يحمل 300040 ومسجل الحد هو 120900، فيمكن للبرنامج الوصول بشكل صحيح إلى جميع العناوين من 300040 إلى 420939 (ضمناً).

يتم تحقيق حماية مساحة الذاكرة من خلال جعل أجهزة وحدة المعالجة المركزية تقارن كل عنوان تم إنشاؤه في وضع المستخدم مع السجلات. تؤدي أي محاولة من قبل برنامج يتم تنفيذه في وضع المستخدم للوصول إلى ذاكرة نظام التشغيل أو ذاكرة مستخدمين آخرين إلى الإعلان عن خطأ من النوع Trap لنظام التشغيل، ويتم التعامل مع المحاولة على أنها خطأ فادح. يمنع هذا النظام برنامج المستخدم من تعديل الكود أو هياكل البيانات (عن طريق الخطأ أو عن عمد) إما لنظام التشغيل أو لمستخدمين آخرين.



Hardware address protection with base and limit registers

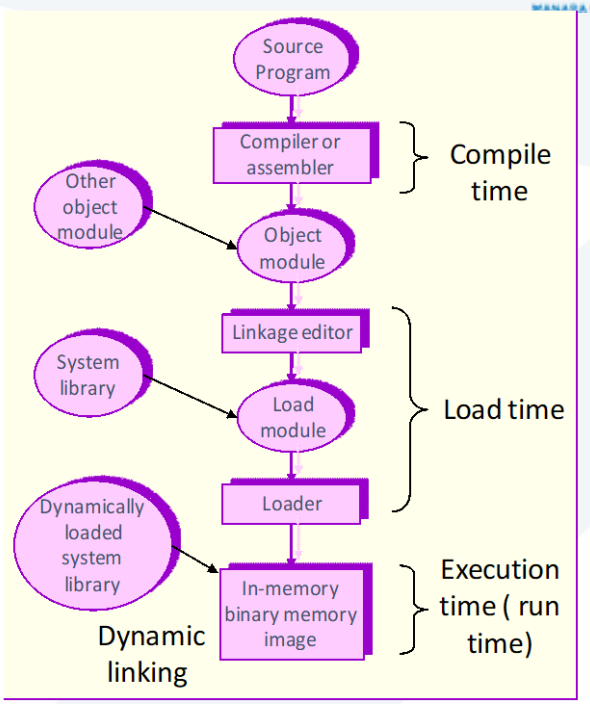
يمكن تحميل مسجلات القاعدة والحد فقط بواسطة نظام التشغيل، الذي يستخدم تعليمات خاصة ذات امتيازات. نظرًا لأنه لا يمكن تنفيذ التعليمات المميزة إلا في وضع kernel، وبما أن نظام التشغيل فقط هو الذي يتم تنفيذه في وضع kernel، فإن نظام التشغيل فقط يمكنه تحميل السجلات القاعدية والحدية. حيث يسمح هذا النظام لنظام التشغيل بتغيير قيمة السجلات ولكنه يمنع برامج المستخدم من تغيير محتويات السجلات.

يتم منح نظام التشغيل، الذي يتم تنفيذه في وضع kernel، وصولاً غير مقيد إلى ذاكرة نظام التشغيل وذاكرة المستخدمين. يسمح هذا الحكم لنظام التشغيل بتحميل برامج المستخدمين في ذاكرة المستخدمين، وإلغاء تلك البرامج في حالة حدوث أخطاء، والوصول إلى بارامترات استدعاءات النظام وتعديلها، وتنفيذ الإدخال / الإخراج من ذاكرة المستخدم وإليها، وتوفير العديد من خدمات أخرى. ينفذ نظام التشغيل لنظام المعالجة تبديل السياق، وتخزين حالة عملية واحدة من السجلات في الذاكرة الرئيسية قبل تحميل سياق العملية التالية من الذاكرة الرئيسية إلى السجلات.

ربط العناوين

يوجد البرنامج عادة، على القرص كملف ثنائي قابل للتنفيذ. حيث يجب إحضار البرنامج إلى الذاكرة ووضعه في سياق العملية ليصبح مؤهلاً للتنفيذ على وحدة المعالجة المركزية المتاحة. أثناء تنفيذ العملية، تصل إلى الإرشادات والبيانات من الذاكرة. في النهاية، تنتهي العملية، ويتم تحرير ذاكرتها لتستخدمها عمليات أخرى.

في معظم الحالات، يمر برنامج المستخدم بعدة خطوات – قد يكون بعضها اختياريًا – قبل تنفيذه. قد يتم تمثيل العناوين بطرق مختلفة خلال هذه الخطوات. إذ تكون العناوين في البرنامج المصدر رمزية بشكل عام (مثل المتغير count). يقوم المترجم عادةً بربط هذه العناوين الرمزية بالعناوين القابلة لإعادة التموضع (على سبيل المثال "14 بايت من بداية الوحدة"). يقوم الرابط Linker أو المحمل Loader بدوره بربط العناوين القابلة للنقل بالعناوين المطلقة (مثل 74014). كل ربط هو تحويل من مساحة عنوان إلى أخرى.



تقليديًا، يمكن ربط التعليمات والبيانات بعناوين الذاكرة في أي خطوة على طول الطريق إلى التنفيذ:

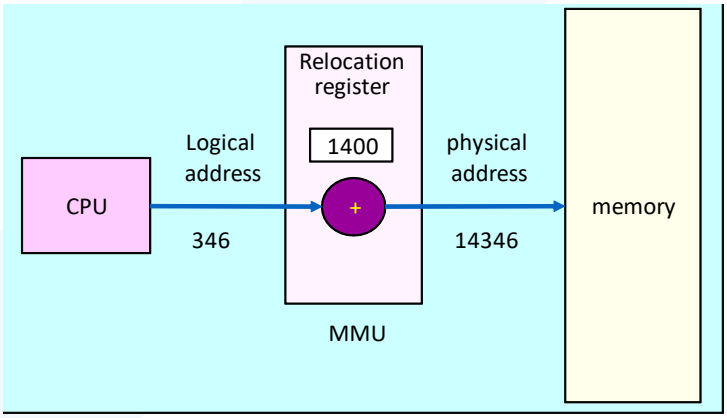
في وقت الترجمة Compiling: إذا كنت تعرف في وقت الترجمة مكان وجود العملية في الذاكرة، فيمكن عندئذٍ إنشاء رمز مطلق. على سبيل المثال، إذا كنت تعلم أن عملية المستخدم ستقيم بدءًا من الموقع R، فسيبدأ كود المترجم الذي تم إنشاؤه في هذا الموقع ويمتد من هناك. إذا تغير موقع البداية في وقت لاحق، فسيكون من الضروري إعادة ترجمة هذا الكود.

وقت التحميل: إذا لم يكن معروفًا في وقت الترجمة مكان العملية في الذاكرة، فيجب على المترجم إنشاء كود قابل لإعادة التموضع. في هذه الحالة، يتم تأخير الربط النهائي حتى وقت التحميل. إذا تغير عنوان البداية، فسنحتاج فقط إلى إعادة تحميل كود المستخدم لتعديل هذه القيمة المتغيرة.

وقت التنفيذ: إذا كان من الممكن نقل العملية أثناء تنفيذها من مقطع ذاكرة إلى آخر، فيجب تأجيل الربط حتى وقت التنفيذ. وتحتاج هذه الطريقة إلى توافر تجهيزات خاصة حتى يعمل هذا النظام، وتستخدمها معظم أنظمة التشغيل.

مساحة العنوان المنطقي مقابل مساحة العنوان الفيزيائي

يشار إلى العنوان الذي تم إنشاؤه بواسطة وحدة المعالجة المركزية بشكل عام على أنه عنوان منطقي.



يؤدي ربط العناوين في وقت التجميع أو التحميل إلى إنشاء عناوين منطقية ومادية متطابقة. بينما ينتج عن ربط العناوين في وقت التنفيذ عناوين منطقية ومادية مختلفة. في هذه الحالة، نشير عادةً إلى العنوان المنطقي كعنوان افتراضي.

مجموعة جميع العناوين المنطقية التي تم إنشاؤها بواسطة برنامج هي فضاء العناوين المنطقية. بينما مجموعة جميع العناوين الفيزيائية المقابلة لهذه العناوين المنطقية هي فضاء العناوين الفيزيائية. وبالتالي، فإنه في حالة ربط العنوان وقت التنفيذ، تختلف فضاءات العناوين المنطقية والفيزيائية.

يتم وقت التشغيل التبديل من العناوين المنطقية إلى العناوين الفيزيائية بواسطة جهاز يسمى وحدة إدارة الذاكرة (MMU). ويسمى المسجل القاعدي هنا مسجل إعادة التموضع. تتم إضافة القيمة الموجودة في مسجل إعادة التموضع

إلى كل عنوان تم إنشاؤه بواسطة عملية المستخدم في وقت إرسال العنوان إلى الذاكرة. على سبيل المثال، إذا كانت قيمة القاعدة عند 14000، فسيتم نقل عنوان المستخدم في الموقع 0 ديناميكيًا إلى الموقع 14000 ؛ وتم تحويل الوصول إلى الموقع 346 إلى الموقع 14346.

لا يصل برنامج المستخدم إلى العناوين الفيزيائية الحقيقية أبدًا. يمكن للبرنامج إنشاء مؤشر للموقع 346، وتخزينه في الذاكرة، ومعالجته، ومقارنته بالعناوين الأخرى كلها مثل الرقم 346. فقط عندما يتم استخدامه كعنوان ذاكرة (ربما في تحميل أو تخزين غير مباشر) تم نقله بالنسبة إلى المسجل القاعدي. يتعامل برنامج المستخدم مع العناوين المنطقية. تقوم أجهزة ربط الذاكرة بتحويل العناوين المنطقية إلى عناوين فيزيائية.

لدينا الآن نوعان مختلفان من العناوين: العناوين المنطقية (في النطاق من 0 إلى الحد الأقصى) والعناوين الفيزيائية (في النطاق 0+ R إلى max + R للقيمة الأساسية R). يقوم برنامج المستخدم بإنشاء عناوين منطقية فقط ويعتقد أن العملية تعمل في مواقع الذاكرة من 0 إلى الحد الأقصى. ومع ذلك، يجب تحويل هذه العناوين المنطقية إلى عناوين فيزيائية قبل استخدامها. يعد مفهوم مساحة العنوان المنطقية المرتبطة بمساحة عنوان فعلية منفصلة أمرًا أساسيًا لإدارة الذاكرة المناسبة.

التحميل الديناميكي

كان من الضروري في مناقشتنا حتى الآن، أن يكون البرنامج بأكمله وجميع بيانات العملية في الذاكرة الفيزيائية لتنفيذ العملية. وبالتالي اقتصر حجم العملية على حجم الذاكرة الفيزيائية. للحصول على استخدام أفضل لمساحة الذاكرة، يمكننا استخدام التحميل الديناميكي. مع التحميل الديناميكي، لا يتم تحميل الروتين حتى يتم استدعاؤه. يتم الاحتفاظ بجميع الإجراءات على القرص بتنسيق تحميل قابل لإعادة التموضع. يتم تحميل البرنامج الرئيسي في الذاكرة ويتم تنفيذه. عندما يحتاج روتين إلى استدعاء روتين آخر، يتحقق روتين الاستدعاء ما إذا كان قد تم تحميل الإجراء الآخر. إذا لم يحدث ذلك، يتم استدعاء أداة تحميل الارتباط لتحميل الروتين المطلوب في الذاكرة وتحديث جداول عناوين البرنامج لتعكس هذا التغيير. ثم يتم تمرير التحكم إلى روتين آخر تم تحميله حديثًا.

تتمثل ميزة التحميل الديناميكي في أنه يتم تحميل الروتين فقط عند الحاجة إليه. هذه الطريقة مفيدة بشكل خاص عند الحاجة إلى كميات كبيرة من التعليمات البرمجية للتعامل مع الحالات النادرة الحدوث، مثل إجراءات الخطأ. في مثل هذه الحالة، على الرغم من أن الحجم الإجمالي للبرنامج قد يكون كبيرًا، إلا أن الجزء المستخدم قد يكون أصغر بكثير.

لا يتطلب التحميل الديناميكي دعمًا خاصًا من نظام التشغيل. تقع على عاتق المستخدمين مسؤولية تصميم برامجهم للاستفادة من هذه الطريقة. قد تساعد أنظمة التشغيل المبرمج، مع ذلك، من خلال توفير إجراءات المكتبة لتنفيذ التحميل الديناميكي.

الارتباط الديناميكي والمكتبات المشتركة

المكتبات المرتبطة ديناميكيًا (DLLs) هي مكتبات نظام مرتبطة ببرامج المستخدم عند تشغيل البرامج. تدعم بعض أنظمة التشغيل الربط الستاتيكي فقط، حيث يتم التعامل مع مكتبات النظام مثل أي وحدة كائن أخرى ويتم دمجها بواسطة المُحمل في صورة البرنامج الثنائي. أما الربط الديناميكي، في المقابل، فهو يشبه التحميل الديناميكي.

هنا، بالرغم من ذلك، يتم تأجيل الربط، بدلاً من التحميل، حتى وقت التنفيذ. تُستخدم هذه الميزة عادةً مع مكتبات النظام، مثل مكتبة لغة C القياسية، وبدون هذه الميزة، يجب أن يتضمن كل برنامج في النظام نسخة من مكتبة اللغة الخاصة به (أو على الأقل الإجراءات التي يشير إليها البرنامج) في الصورة القابلة للتنفيذ. لا يؤدي هذا المطلب إلى زيادة حجم الصورة القابلة للتنفيذ فحسب، بل قد يؤدي أيضًا إلى إهدار الذاكرة الرئيسية. الميزة الثانية لـ DLL هي أنه يمكن مشاركة هذه المكتبات بين عمليات متعددة، بحيث يكون نسخة واحدة فقط من DLL في الذاكرة الرئيسية. لهذا السبب، تُعرف مكتبات DLL أيضًا بالمكتبات المشتركة، وتُستخدم على نطاق واسع في أنظمة Windows و Linux.

عندما يشير أحد البرامج إلى روتين موجود في مكتبة ديناميكية، يقوم المُحمل بتحديد موقع DLL، وتحميله في الذاكرة إذا لزم الأمر. ثم يقوم بضبط العناوين التي تشير إلى الوظائف في المكتبة الديناميكية إلى الموقع الموجود في الذاكرة حيث يتم تخزين DLL.

يجب في حال عدم تطبيق الارتباط الديناميكي، إعادة ربط جميع هذه البرامج للوصول إلى المكتبة الجديدة. حتى لا تنفذ البرامج عن طريق الخطأ إصدارات جديدة غير متوافقة من المكتبات، يتم تضمين معلومات الإصدار في كل من البرنامج والمكتبة. يمكن تحميل أكثر من إصدار من المكتبة في الذاكرة، ويستخدم كل برنامج معلومات الإصدار الخاصة به لتحديد نسخة المكتبة التي سيتم استخدامها. الإصدارات التي تحتوي على تغييرات طفيفة تحتفظ بنفس رقم الإصدار، بينما الإصدارات ذات التغييرات الرئيسية تزيد الرقم. وبالتالي، فإن البرامج التي تم تجميعها مع إصدار المكتبة الجديد فقط هي التي تتأثر بأي تغييرات غير متوافقة مدمجة فيها. ستستمر البرامج الأخرى المرتبطة قبل تثبيت المكتبة الجديدة في استخدام المكتبة القديمة.

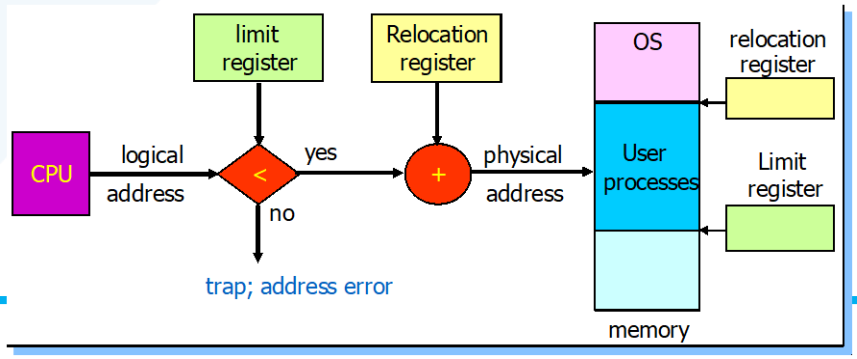
بخلاف التحميل الديناميكي، يتطلب الارتباط الديناميكي والمكتبات المشتركة بشكل عام المساعدة من نظام التشغيل. إذا كانت العمليات في الذاكرة محمية من بعضها البعض، فإن نظام التشغيل هو الكيان الوحيد الذي يمكنه التحقق لمعرفة ما إذا كان الروتين المطلوب موجودًا في مساحة ذاكرة عملية أخرى أو يمكنه السماح لعمليات متعددة بالوصول إلى عناوين الذاكرة نفسها.

التخصيص المتجاور للذاكرة

تنقسم الذاكرة عادةً إلى قسمين: أحدهما لنظام التشغيل والآخر لعمليات المستخدم. يمكننا وضع نظام التشغيل في عناوين ذات ذاكرة منخفضة أو عناوين ذاكرة عالية (كما في Linux و Windows). في تخصيص الذاكرة المتجاور، يتم احتواء كل عملية في قسم واحد من الذاكرة مجاور للقسم الذي يحتوي على العملية التالية. قبل مناقشة مخطط تخصيص الذاكرة هذا بشكل أكبر، لابد من الحديث عن مسألة حماية الذاكرة.

حماية الذاكرة

يمكننا منع عملية من الوصول إلى ذاكرة لا تمتلكها من خلال الجمع بين فكرتين تمت مناقشتهما مسبقًا هما المسجل

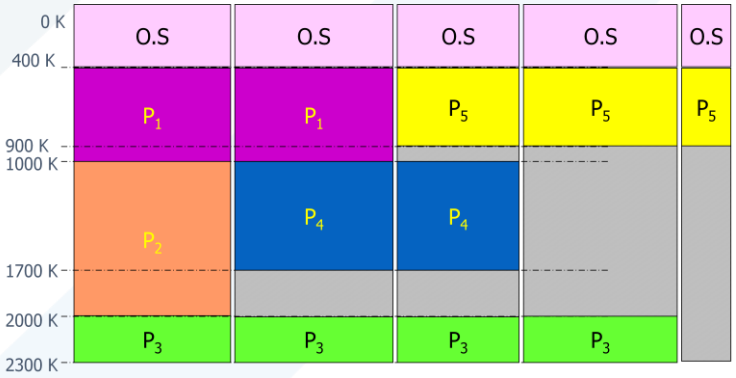


القاعدي، والمسجل الحدي. إذا كان لدينا نظام يمتلك مسجل قاعدي (مسجل اعادة التموضع) مع المسجل الحدي فإننا نحقق هدفنا. يحتوي مسجل إعادة التموضع على قيمة أصغر عنوان مادي ؛ يحتوي المسجل الحدي على نطاق من العناوين المنطقية (على سبيل المثال، إعادة التموضع = 100040 والحدي = 74600). يجب أن يقع كل عنوان منطقي ضمن النطاق المحدد بين المسجلين. تقوم MMU بتعيين العنوان المنطقي ديناميكيًا عن طريق إضافة القيمة في مسجل إعادة التموضع. يتم إرسال هذا العنوان المعين إلى الذاكرة.

عندما يحدد برنامج جدولة وحدة المعالجة المركزية عملية للتنفيذ، يقوم المرسل بتحميل مسجلات إعادة التموضع والحد بالقيم الصحيحة كجزء من تبديل السياق. نظرًا لأن كل عنوان يتم إنشاؤه بواسطة وحدة المعالجة المركزية يتم فحصه للتأكد من قيم هذه السجلات، يمكننا حماية كل من نظام التشغيل وبرامج المستخدمين الآخرين وبياناتهم من التعديل من خلال العملية قيد التشغيل.

تخصيص الذاكرة

تتمثل إحدى أبسط طرق تخصيص الذاكرة في تعيين العمليات لأقسام متغيرة الحجم في الذاكرة، حيث قد يحتوي كل قسم على عملية واحدة بالضبط. في نظام التقسيم المتغير هذا، يحتفظ نظام التشغيل بجدول يشير إلى أجزاء الذاكرة المتوفرة والأجزاء المشغولة. في البداية، تتوفر كل الذاكرة لعمليات المستخدم وتعتبر كتلة واحدة كبيرة من الذاكرة المتاحة، وهي عبارة عن فجوة. في النهاية، كما سترى، تحتوي الذاكرة على مجموعة من الثقوب بأحجام مختلفة.



في البداية، تكون الذاكرة بالكامل متاحة لعمليات المستخدم، مثلا تكون متاحة للعمليات 1 و 2 و 3. يتشكل بعد انجاز العملية الثانية الثانية مثلا، ثقب واحد بعناوين متجاورة. وبعد انجاز العملية الأولى في وقت لاحق تصل العملية 5 ويتم تخصيص الذاكرة لها. ويؤدي هذا إلى وجود فجوتين غير متجاورتين. مع تتالي انجاز العمليات ودخول عمليات جديدة إلى التنفيذ في النظام، يأخذ نظام التشغيل في الاعتبار متطلبات الذاكرة لكل عملية ومقدار مساحة الذاكرة المتاحة أثناء تحديد العمليات التي يتم تخصيص الذاكرة لها. عندما يتم تخصيص مساحة لعملية ما، يتم تحميلها في الذاكرة، حيث يمكنها بعد ذلك الدخول في المنافسة لكسب وقت وحدة المعالجة المركزية. عندما تنتهي عملية ما، فإنها تحرر ذاكرتها، والتي قد يوفرها نظام التشغيل بعد ذلك لعملية أخرى. والآن ماذا يحدث عندما لا توجد ذاكرة كافية لتلبية متطلبات عملية قادمة؟ أحد الخيارات هو ببساطة رفض العملية وتقديم رسالة خطأ مناسبة. ويتم وضع مثل هذه العمليات في قائمة الانتظار. عندما يتم تحرير الذاكرة لاحقًا، يتحقق نظام التشغيل من قائمة الانتظار لتحديد ما إذا كانت ستلبي متطلبات الذاكرة للعملية التي وضعت قيد الانتظار.

بشكل عام، كما ذكرنا، تتكون كتل الذاكرة المتاحة من مجموعة من الفجوات ذات الأحجام المختلفة المنتشرة في جميع أنحاء الذاكرة. عندما تصل عملية ما وتحتاج إلى ذاكرة، يبحث النظام في المجموعة عن فجوة كبيرة بما يكفي لهذه العملية. إذا كانت الفجوة كبيرًا جدًا، يتم تقسيمها إلى قسمين. يخصص واحد للعملية المزمع تنفيذها ؛ ويتم إرجاع الآخر إلى مجموعة الفجوات. وعندما تنتهي العملية، فإنها تحرر كتلة الذاكرة الخاصة بها، والتي يتم وضعها مرة أخرى في مجموعة الفجوات. إذا كانت الفجوة الجديدة مجاورة لفجوات أخرى، يتم دمج هذه الفجوات المتجاورة لتشكيل فجوة واحدة أكبر.

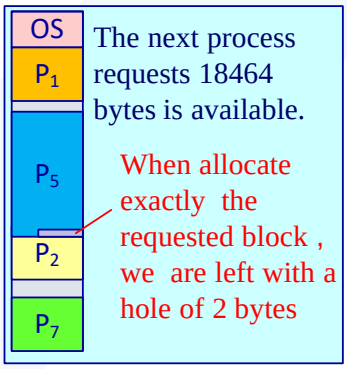
هذا الإجراء هو مثال خاص لمشكلة تخصيص التخزين الديناميكي العامة، والتي تتعلق بكيفية تلبية طلب الحجم n من قائمة الفجوات المتوفرة. هناك العديد من الحلول لهذه المشكلة. حيث تستخدم استراتيجيات مثل الملائمة الأولى والملائمة الأفضل والملائمة الأسوأ الأكثر استخدامًا لتحديد فجوة حرة من مجموعة الفجوات المتاحة.

- **الملاءمة الأولى:** نقوم بتخصيص الفجوة الأولى التي نعثر عليها ويكون حجمها بما يكفي لاستيعاب العملية المطلوبة. يمكن أن يبدأ البحث إما في بداية مجموعة الفجوات أو في الموقع الذي انتهى فيه البحث السابق، ويمكننا التوقف عن البحث بمجرد العثور على فجوة حرة كبيرة بما يكفي.
- **الملائمة الأفضل:** خصص أصغر فجوة يكون حجمها بما يكفي لاستيعاب العملية المطلوبة. هنا يجب علينا البحث في القائمة بأكملها، ما لم تكن القائمة مرتبة حسب الحجم. ينتج بتطبيق هذه الاستراتيجية الحصول على أصغر فجوة متبقية في نهاية التخصيص دائما.
- **الملاءمة الأسوأ:** نقوم بتخصيص أكبر فجوة. وهنا علينا مرة أخرى، البحث في القائمة بأكملها، ما لم يتم فرزها حسب الحجم. ينتج بتطبيق هذه الاستراتيجية الحصول على أكبر فجوة متبقية، والتي قد تكون أكثر فائدة من الفجوة الأصغر المتبقية بتطبيق الملائمة الأفضل.

أظهرت عمليات المحاكاة أن كلاً من الملاءمة الأولى والملائمة الأفضل أفضل من الملائمة الأسوأ من حيث تقليل الوقت واستخدام التخزين.و من الواضح أنه لا يمكن الجزم بأفضلية الملاءمة الأولى ولا الملائمة الأفضل من حيث استخدام التخزين، ولكن الملاءمة الأولى تكون أسرع عمومًا.

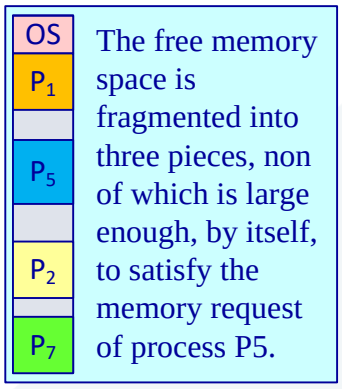
التجزئة

يتم تعريف التجزئة على أنها تشكل فجوات متباينة غير قابلة للاستخدام عند تحميل العملية وإزالتها بعد التنفيذ من الذاكرة. لا يمكن تخصيص هذه الفجوات لعمليات جديدة بسبب عدم دمجها أو كونها لا تفي بمتطلبات الذاكرة للعملية. لتحقيق درجة من البرمجة المتعددة، يجب أن نقلل من إهدار الذاكرة الناجمة عن هذه التجزئة. هناك نوعان من التجزئة في أنظمة التشغيل:



- **التجزئة الداخلية:** تحدث التجزئة الداخلية عندما يتم تخصيص كتل الذاكرة للعملية أكثر من الحجم المطلوب. نتيجة لذلك، يتم ترك بعض المساحة غير المستخدمة، والتي لا يسمح نظام التشغيل أو بنية النظام الحاسوبي باستخدامها لعملية أخرى، وهذا يؤدي إلى مشكلة تجزئة داخلية.

- **التجزئة الخارجية:** في التجزئة الخارجية، لدينا كتلة ذاكرة خالية، لكن لا يمكننا تخصيصها لعملية لأن الكتل ليست متجاورة. مثال: بفرض أن ثلاث عمليات p1 و p2 و p3 تأتي بأحجام 2 ميجابايت و 4 ميجابايت و 7 ميجابايت على التوالي. الآن يحصلون على كتل ذاكرة بحجم 3 ميجابايت و 6 ميجابايت و 7 ميجابايت مخصصة على التوالي. بعد تخصيص العملية p1 تركت مساحة 1 ميغا بايت غير مستخدمة. وكذلك بعد تخصيص العملية 2 تركت 2 ميغابايت غير مستخدمة. بفرض أن عملية جديدة p4 أتت وتتطلب كتلة 3 ميغا بايت من الذاكرة، وهي متوفرة، لكن لا يمكننا تخصيصها لأن مساحة الذاكرة الخالية ليست متجاورة. هذا يسمى التجزئة الخارجية.



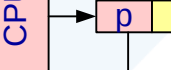
تتأثر كلا من الملائمة الأولى والملائمة الأفضل لتخصيص الذاكرة بالتجزئة الخارجية. للتغلب على مشكلة التجزئة الخارجية يتم استخدام الضغط Compaction . يتم في تقنية الضغط تجميع فجوات الذاكرة وتشكيل فجوة واحدة كبيرة مما يتيح استخدام هذه الفجوة من قبل العمليات أخرى بشكل فعال.

هناك حل آخر محتمل للتجزئة الخارجية وهو السماح لمساحة العنوان المنطقي للعمليات بأن تكون غير متجاورة، وبالتالي السماح للعملية بتخصيص ذاكرة فعلية أينما كان هذا الأخير متاحًا، وهذا ما يتم من خلال استخدام تقنيات مثل التصفيح Paging أو التجزئة Segmentation.

التصفيح

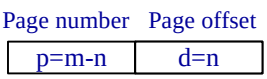
الطريقة الأساسية

تتضمن الطريقة الأساسية لتنفيذ التصفيح تقسيم الذاكرة الفعلية إلى كتل ثابتة الحجم تسمى الإطارات وتقسيم الذاكرة المنطقية إلى كتل من نفس الحجم تسمى الصفحات. عندما يتم تنفيذ عملية ما، يتم تحميل صفحاتها في أي إطارات ذاكرة متوفرة من مصدرها. يتم تقسيم مساحة التخزين للتطبيق إلى كتل ثابتة الحجم بنفس حجم إطارات الذاكرة أو مجموعات من إطارات متعددة. هنا تصبح مساحة العنوان المنطقي منفصلة تمامًا عن مساحة العنوان الفعلية، لذلك يمكن أن تحتوي العملية على مساحة عنوان منطقية 64 بت على الرغم من أن النظام يحتوي على أقل من 2^{64} بايت من الذاكرة الفعلية.



ينقسم كل عنوان تم إنشاؤه بواسطة وحدة المعالجة المركزية إلى جزأين: رقم الصفحة (p) وإزاحة الصفحة (d):

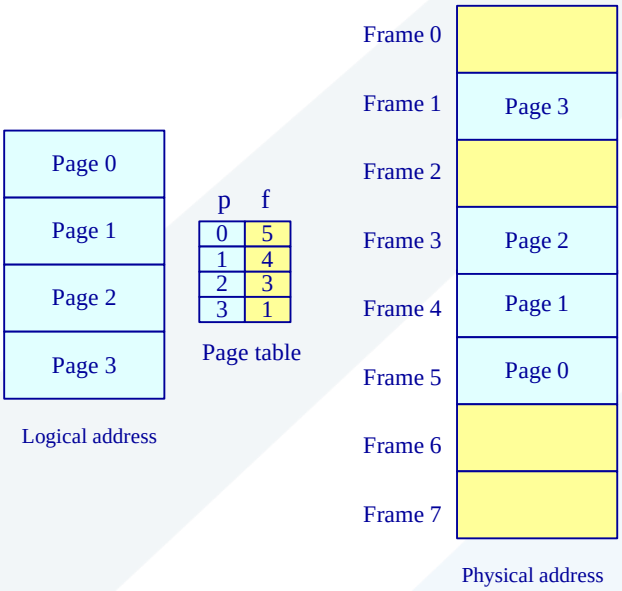
يتم استخدام رقم الصفحة كفهرس في جدول الصفحات لكل عملية. يحتوي جدول الصفحات على



العنوان الأساسي لكل إطار في الذاكرة الفعلية، والإزاحة هي موقع الإطار الذي تتم الإشارة إليه. وبالتالي، يتم دمج العنوان الأساسي للإطار مع إزاحة الصفحة لتحديد عنوان الذاكرة الفعلي.

يوضح ما يلي الخطوات التي تتخذها MMU لترجمة العنوان المنطقي الذي تم إنشاؤه بواسطة وحدة المعالجة المركزية إلى عنوان فعلي:

1. استخرج رقم الصفحة p واستخدمه كفهرس في جدول الصفحات.
2. استخرج رقم الإطار المقابل f من جدول الصفحة.
3. استبدل رقم الصفحة p في العنوان المنطقي برقم الإطار f.



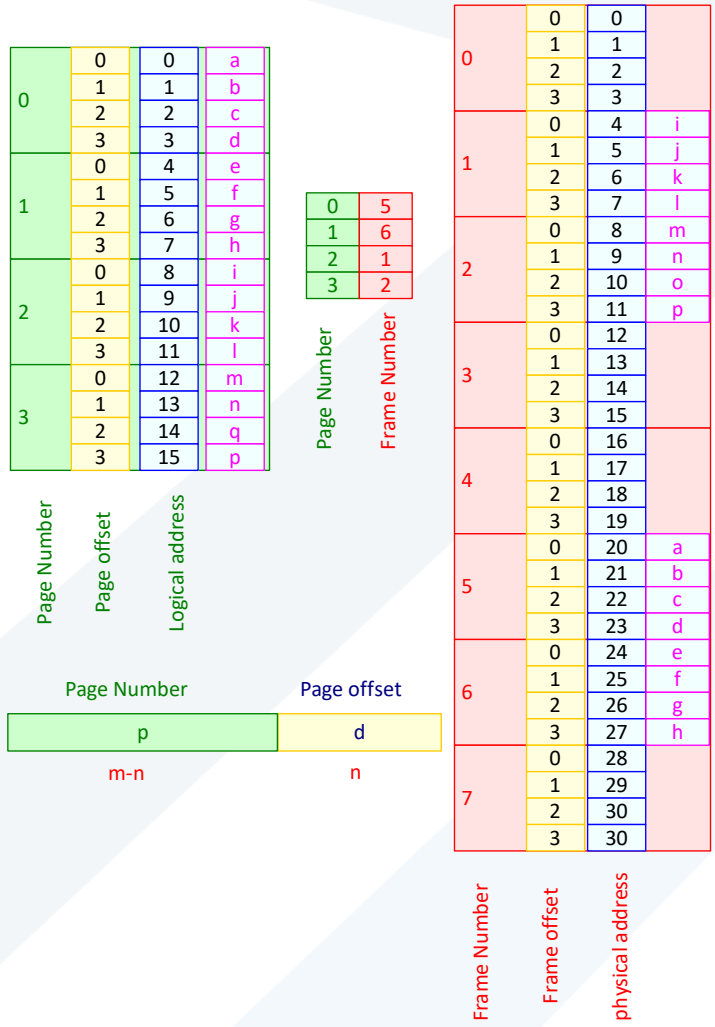
نظرًا لأن الإزاحة d لا تتغير، فلا يتم استبدالها، ويشتمل رقم الإطار والإزاحة الآن على العنوان الفعلي.

يتم تحديد حجم الصفحة (مثل حجم الإطار) بالاعتماد على البنية الفيزيائية. يكون حجم الصفحة عددا من مرتبة n^2، ويتراوح عادةً بين 4 كيلوبايت و 1 غيغابايت لكل صفحة، اعتمادًا على بنية الكمبيوتر. إن اختيار قوة 2 كحجم صفحة يجعل ترجمة العنوان المنطقي إلى رقم صفحة وإزاحة الصفحة أمرًا سهلاً. إذا كان حجم مساحة العنوان المنطقي هو 2^m، وكان حجم الصفحة 2^n بايت، فإن البتات الأكثر أهمية لعنوان منطقي تحدد رقم الصفحة، وتعين البتات ذات الترتيب المنخفض n إزاحة الصفحة. وبالتالي، يكون العنوان المنطقي كما يلي:

p هو فهرس في جدول الصفحة و d هو الإزاحة داخل الصفحة.

ضع في اعتبارك الذاكرة في الشكل. هنا، في العنوان المنطقي، n = 2 و m = 4. باستخدام حجم صفحة يبلغ 4 بايت وذاكرة فعلية تبلغ 32 بايت (8 صفحات)، نوضح كيف يمكن تعيين الذاكرة المنطقية في الذاكرة الفعلية. العنوان المنطقي 0 هو الصفحة 0، الإزاحة 0. عند الفهرسة في جدول الصفحات، نجد أن الصفحة 0 موجودة في الإطار 5. وبالتالي، فإن العنوان المنطقي 0 يعين العنوان الفعلي 20 [= (5 × 4) +0]. العنوان المنطقي 3 (الصفحة 0، الإزاحة 3) يخطط للعنوان الفعلي 23 [= (5 × 4) +3]. العنوان المنطقي 4 هو الصفحة 1، الإزاحة 0 ؛ وفقًا لجدول الصفحة، يتم تعيين الصفحة 1 للإطار 6. وبالتالي، فإن العنوان المنطقي 4 يخطط للعنوان الفعلي 24 [= (6 × 4) +0]. العنوان المنطقي 13 خريطة للعنوان الفعلي 9.

ربما لاحظت أن التصفيح نفسه هو شكل من أشكال الربط الديناميكي. حيث يرتبط كل عنوان منطقي بجهاز التصفيح ببعض العناوين الفعلية. استخدام التصفيح مشابه لاستخدام جدول السجلات الأساسية (أو إعادة التموضع في الذاكرة)، واحد لكل إطار من إطارات الذاكرة.



عندما نستخدم نظام التصفيح، فليس لدينا تجزئة خارجية: يمكن تخصيص أي إطار حر لعملية تحتاج إليه. ومع ذلك، قد يكون لدينا بعض التجزئة الداخلية. لاحظ أنه تم تخصيص الإطارات كوحدات. إذا لم تتطابق متطلبات الذاكرة لعملية ما مع حدود الصفحة، فقد لا يكون الإطار الأخير المخصص ممتلئًا تمامًا. على سبيل المثال، إذا كان حجم الصفحة هو 2048 بايت، فستحتاج عملية 72766 بايت إلى 35 صفحة بالإضافة إلى 1086 بايت.

سيتم تخصيص 36 إطارًا، مما ينتج عنه تجزئة داخلية من 2048 − 1086 = 962 بايت. في أسوأ الحالات، قد تحتاج العملية إلى n pages و byte واحد. سيتم تخصيص إطارات n + 1، مما يؤدي إلى تجزئة داخلية بحجم إطار كامل تقريبًا.

إذا كان حجم العملية مستقلاً عن حجم الصفحة، فإننا نتوقع أن يصل متوسط التجزئة الداخلية إلى نصف صفحة لكل عملية. يشير هذا الاعتبار إلى أن أحجام الصفحات الصغيرة أمر مرغوب فيه. ومع ذلك، يتم تضمين الحمل الزائد في كل إدخال في جدول الصفحات، ويتم تقليل هذا الحمل مع زيادة حجم الصفحات. أيضًا، يكون إدخال / إخراج القرص أكثر كفاءة عندما تكون كمية البيانات التي يتم نقلها أكبر. بشكل عام، نمت أحجام الصفحات بمرور الوقت حيث أصبحت العمليات ومجموعات البيانات والذاكرة الرئيسية أكبر. اليوم، يبلغ حجم الصفحات عادةً 4 كيلوبايت أو 8 كيلوبايت، وتدعم بعض الأنظمة أحجام صفحات أكبر. تدعم بعض وحدات المعالجة المركزية وأنظمة التشغيل أحجام صفحات متعددة. على سبيل المثال، في أنظمة x86-64، يدعم Windows 10 أحجام الصفحات من 4 كيلوبايت و 2 ميجابايت. يدعم Linux أيضًا حجمين من أحجام الصفحات: حجم الصفحة الافتراضي (عادةً 4 كيلوبايت) وحجم الصفحة الأكبر الذي يعتمد على البنية والذي يسمى الصفحات الضخمة.

في كثير من الأحيان، في وحدة المعالجة المركزية 32 بت، يبلغ طول كل إدخال في جدول الصفحات 4 بايت، ولكن هذا الحجم يمكن أن يختلف أيضًا. يمكن أن يشير الإدخال 32 بت إلى واحد من 2^{32} إطارًا للصفحة. إذا كان حجم الإطار 4 كيلوبايت (2^{12})، فيمكن للنظام الذي يحتوي على إدخالات 4 بايت معالجة 2^{44} بايت (أو 16 تيرابايت) من الذاكرة الفعلية. يجب أن نلاحظ هنا أن حجم الذاكرة الفعلية في نظام ذاكرة مقسم إلى صفحات يختلف عادةً عن الحد الأقصى للحجم المنطقي للعملية.

عندما تصل عملية إلى النظام ليتم تنفيذها، يتم فحص حجمها، معبراً عنه بالصفحات. كل صفحة من العملية تحتاج إلى إطار واحد. وبالتالي، إذا كانت العملية تتطلب عدد n من الصفحات، فيجب توفير عدد n من الإطارات على الأقل في الذاكرة. إذا كانت n إطارات متوفرة، فسيتم تخصيصها لعملية الوصول هذه. يتم تحميل الصفحة الأولى من العملية في أحد الإطارات المخصصة، ويتم وضع رقم الإطار في جدول الصفحات لهذه العملية. يتم تحميل الصفحة التالية في إطار آخر، ويتم وضع رقم الإطار الخاص بها في جدول الصفحة، وهكذا.

أحد الجوانب المهمة للتصفيح هو الفصل الواضح بين وجهة نظر المبرمج للذاكرة والذاكرة الفعلية. يرى المبرمج الذاكرة على أنها مساحة واحدة، تحتوي فقط على هذا البرنامج الواحد. في الواقع، ولكن في الواقع فإن برنامج المستخدم مبعثر في جميع أنحاء الذاكرة الفعلية، والتي تحتوي أيضًا على برامج أخرى. يتم التوفيق بين عرض المبرمج للذاكرة والذاكرة الفعلية بواسطة جهاز ترجمة العنوان. يتم ترجمة العناوين المنطقية إلى عناوين فعلية. هذا الربط مخفي عن المبرمج ويتحكم فيه نظام التشغيل.

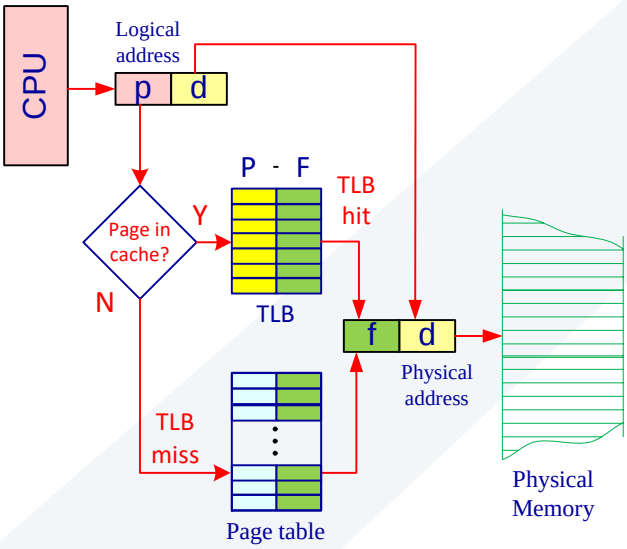
لاحظ أن عملية المستخدم بحكم التعريف غير قادرة على الوصول إلى الذاكرة التي لا تمتلكها. ليس لديها طريقة لمعالجة الذاكرة خارج جدول الصفحات الخاص بها، والجدول يتضمن فقط تلك الصفحات التي تمتلكها العملية.

نظرًا لأن نظام التشغيل يدير الذاكرة الفعلية، يجب أن يكون على دراية بتفاصيل تخصيص الذاكرة الفعلية – أي الإطارات يتم تخصيصها، والإطارات المتاحة، وعدد الإطارات الإجمالية الموجودة، وما إلى ذلك. يتم الاحتفاظ بهذه المعلومات بشكل عام في قاعدة بيانات واحدة على مستوى النظام تسمى جدول الأطر. يحتوي جدول الأطر على إدخال واحد لكل إطار، مما يشير إلى ما إذا كان هذا الأخير متاحا أم مخصصًا، وإذا تم تخصيصه، فصفحة أي عملية (أو عمليات) تحتله.

بالإضافة إلى ذلك، يجب أن يكون نظام التشغيل على دراية بأن عمليات المستخدم تعمل في مساحة المستخدم، ويجب تعيين كافة العناوين المنطقية لإنتاج عناوين فعلية. إذا أجرى المستخدم استدعاء نظام (لإجراء I / O، على سبيل المثال) وقدم عنوانًا كبارامتر (مسجل، على سبيل المثال)، يجب تعيين هذا العنوان لإنتاج العنوان الفعلي الصحيح. يحتفظ نظام التشغيل بنسخة من جدول الصفحات لكل عملية، تمامًا كما يحتفظ بنسخة من عداد التعليمات ويسجل المحتويات. تُستخدم هذه النسخة لترجمة العناوين المنطقية إلى عناوين فعلية عندما يتعين على نظام التشغيل تخصيص عنوان منطقي لعنوان فيزيائي يدويًا. يتم استخدامه أيضًا بواسطة مرحل وحدة المعالجة المركزية لتحديد جدول صفحة الأجهزة عندما يتم تخصيص وحدة المعالجة المركزية للعملية. لذلك يزيد التصفيح من وقت تبديل السياق.

دعم البنية الفيزيائية

يتم باستخدام تقنية التصفيح في نظام التشغيل، إنشاء جدول صفحات لكل عملية، والذي يحتوي على مدخلات مثل رقم الإطار أي عنوان الذاكرة الرئيسية التي نريد الإشارة إليها، وبعض البتات المفيدة الأخرى (على سبيل المثال، البت الذي يشير إلى صلاحية العنوان أو عدم الصلاحية صالح V/I).



السؤال الآن هو أين نضع جدول الصفحات، بحيث يكون وقت الوصول الإجمالي إلى محتوياته أقل ما يمكن. كانت المشكلة في البداية هي الوصول السريع إلى محتوى الذاكرة الرئيسي بناءً على العنوان الذي تم إنشاؤه بواسطة وحدة المعالجة المركزية (أي العنوان المنطقي). في البداية، فكر البعض في استخدام المسجلات لتخزين جداول الصفحات، بحيث يكون وقت الوصول أقل. أي وضع مدخلات جدول الصفحات في المسجلات، لكل طلب يتم إنشاؤه من وحدة المعالجة المركزية، حيث تتم مطابقته مع رقم الصفحة المناسب لجدول الصفحات، والذي سيحدد الآن المكان الموجود في الذاكرة الرئيسية لتلك الصفحة. ولكن المشكلة هي أن حجم المسجلات صغير (عمليًا، يمكن أن يستوعب بحد أقصى 0.5 كيلو إلى 1 كيلو من إدخالات جدول الصفحة). في الواقع قد يكون حجم العملية كبيرًا وهذا يؤدي إلى كون جدول الصفحات المطلوب كبيرًا أيضا، لذلك قد لا تحتوي السجلات على جميع المدخلات الخاصة بجدول الصفحة. و هذا ليس نهجًا عمليًا.

يتم الاحتفاظ بجدول الصفحات بالكامل في الذاكرة الرئيسية للتغلب على مشكلة الحجم هذه. لكن المشكلة هنا هي أن هناك مرجعين رئيسيين للذاكرة مطلوبان:

- للعثور على رقم الإطار
- ثم للذهاب إلى العنوان المحدد برقم الإطار

للتغلب على هذه المشكلة، تم إعداد ذاكرة تخزين مؤقت عالية السرعة لإدخالات جدول الصفحات تسمى Translation Lookaside Buffer (TLB). وهذه ذاكرة تخزين مؤقت خاصة تستخدم لتتبع المعاملات المستخدمة مؤخرًا. يحتوي TLB على إدخالات جدول الصفحات التي تم استخدامها مؤخرًا. نظرًا لعنوان افتراضي، يفحص المعالج TLB في حالة وجود إدخال جدول صفحة (TLB hit)، ويتم استرداد رقم الإطار وتشكيل العنوان الحقيقي. إذا لم يتم العثور على إدخال جدول الصفحات في (TLB miss)، فسيتم استخدام رقم الصفحة كدليل أثناء معالجة جدول الصفحات. يتحقق TLB أولاً مما إذا كانت الصفحة موجودة بالفعل في الذاكرة الرئيسية، وإذا لم تكن موجودة في الذاكرة الرئيسية، فسيتم إصدار خطأ في الصفحة، ثم يتم تحديث TLB لتضمين إدخال الصفحة الجديدة.

خطوات TLB hit

- تقوم وحدة المعالجة المركزية بإنشاء عنوان منطقي.
- تم التحقق من وجوده في TLB (إذا كان موجودا).

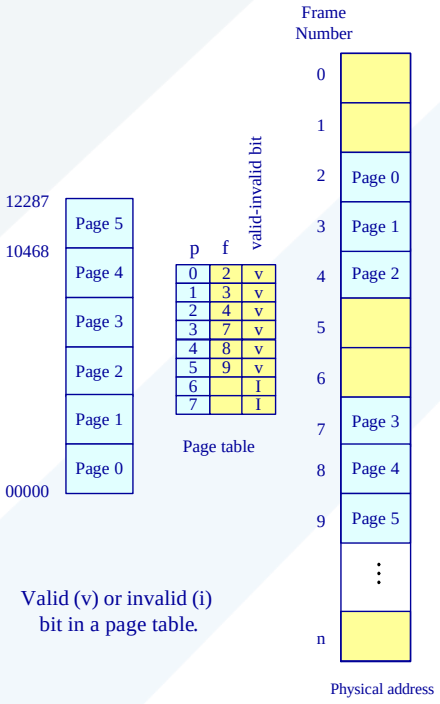
- يتم استرداد رقم الإطار المقابل، والذي يخبر الآن أين تكمن صفحة الذاكرة الرئيسية.

خطوات TLB miss

- تقوم وحدة المعالجة المركزية بإنشاء عنوان افتراضي (منطقي).
- يتم التحقق منه في TLB (إذا كان غير موجود).
- الآن تتم مطابقة رقم الصفحة مع جدول الصفحات الموجود في الذاكرة الرئيسية (بافتراض أن جدول الصفحة يحتوي على جميع PTE).
- يتم استرداد رقم الإطار المقابل، والذي يخبرنا الآن أين تكمن صفحة الذاكرة الرئيسية.
- يتم تحديث TLB باستخدام PTE الجديد (إذا لم تكن هناك مساحة، يتم استخدام إحدى تقنيات الاستبدال التي سنتحدث عنها لاحقا، مثل FIFO أو LRU أو MFU وما إلى ذلك).

الحماية

- يتم حماية الذاكرة في بيئة مقسمة بواسطة بتات حماية مرتبطة بكل إطار. عادة، يتم الاحتفاظ بهذه البتات في جدول الصفحات. بت واحد يمكن أن يحدد ما إذا كانت الصفحة للقراءة والكتابة أو للقراءة فقط.
- كل مرجع إلى الذاكرة يمر عبر جدول الصفحات للعثور على رقم الإطار الصحيح. في نفس الوقت الذي يتم فيه حساب العنوان الفعلي، يمكن التحقق من بتات الحماية للتحقق من عدم إجراء عمليات كتابة على صفحة للقراءة فقط.
- تؤدي محاولة الكتابة على صفحة للقراءة فقط إلى اعتراض الجهاز على نظام التشغيل (أي الاعلان عن انتهاك حماية الذاكرة).
- يتم إرفاق بت واحد إضافي بشكل عام بكل إدخال في جدول الصفحة: بت صالح – غير صالح V/I.
- عند تعيين هذا البت لتكون " صالح "، تكون الصفحة المرتبطة في مساحة العنوان المنطقي للعملية وبالتالي فهي صفحة صحيحة (أو صالحة).
- عند كون البت "غير صالح"، لا تكون الصفحة في مساحة العنوان المنطقي للعملية.

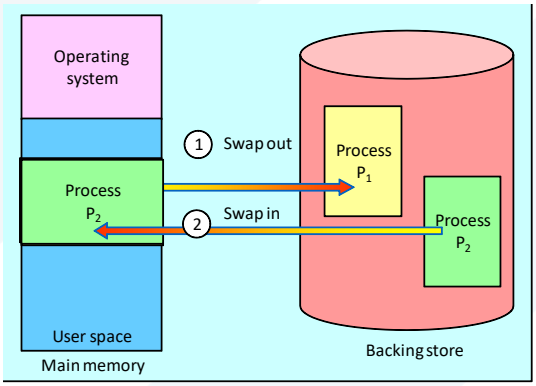


Valid (v) or invalid (i) bit in a page table.

- يتم حجز العناوين غير الصحيحة عن طريق استخدام بت صالح وغير صالح. يقوم نظام التشغيل بتعيين هذا البت لكل صفحة للسماح أو عدم السماح بالوصول إلى الصفحة.

- • افترض، على سبيل المثال، أنه في نظام بمساحة عنوان 14 بت (0 إلى 16383)، لدينا برنامج يجب أن يستخدم العناوين من 0 إلى 10468 فقط.
- حجم الصفحة 2 كيلو بايت (6 صفحات 2048 * 6 = 12288).
- يتم تعيين عناوين العناصر في الصفحات 0 و 1 و 2 و 3 و 4 و 5 بشكل طبيعي من خلال جدول الصفحات.
- أي محاولة لإنشاء عنوان الصفحات 6 أو 7، ستشير بت الصلاحية V/I قيمة غير صالح أي I، وسيقوم النظام بإعلان خطأ Trap (مرجع صفحة غير صالح).
- لاحظ أن المشكلة هنا تكمن في أن البرنامج يمتد إلى العنوان 10468 فقط، وأي مرجع يتجاوز هذا العنوان يعتبر غير صحيح.
- يتم تصنيف المرجع إلى الصفحة 5 على أن الصفحة صالحة، ويكون الوصول إلى عناوين تصل إلى 12287 صالحا. وتكون العناوين من 12288 إلى 16383 غير صالحة.

المبادلة Swapping



عند تنفيذ عملية ما، يجب أن تكون موجودة في الذاكرة. المبادلة هي عملية سحب عملية مؤقتا من الذاكرة الرئيسية إلى الذاكرة الثانوية، واستبدالها بعملية أخرى من الذاكرة الثانوية. تسمح المبادلة بتشغيل المزيد من العمليات. المهم في المبادلة هو وقت المبادلة والوقت الإجمالي والذي يتناسب طردًا مع مقدار الذاكرة التي يتم تبديلها. تفيد المبادلة في تنفيذ المقاطعات فإذا وصلت عملية ذات أولوية أعلى وتريد الخدمة، فيمكن لمدير الذاكرة سحب العملية ذات الأولوية الأقل وتحميل وتنفيذ العملية ذات الأولوية الأعلى. بعد الانتهاء من العملية ذات الأولوية الأعلى، يتم ادخال العملية ذات الأولوية المنخفضة مرة أخرى في الذاكرة ومواصلة تنفيذها.