OPERATING SYSTEM

Lecture Notes

Dr. Professor, J.M. Khalifeh

قسم المعلوماتية

الوحدة السابعة

النسخة العربية

Unit-7

Virtual Memory Management

ملخص

ناقشنا سابقا استراتيجيات إدارة الذاكرة المختلفة المستخدمة في أنظمة الحاسوب. جميع هذه الاستراتيجيات لها هدف واحد: الاحتفاظ بالعديد من العمليات في الذاكرة في وقت واحد للسماح بالبرمجة المتعددة. ومع ذلك، فإنها تميل إلى اشتراط وجود العملية بأكملها في الذاكرة قبل تنفيذها.

الذاكرة الافتراضية هي تقنية تسمح بتنفيذ العمليات التي ليست كاملة في الذاكرة. من أهم مزايا هذه التقنية أن البرامج يمكن أن تكون أكبر من الذاكرة الفعلية. علاوة على ذلك، تُلخص الذاكرة الافتراضية الذاكرة الرئيسية في مصفوفة تخزين كبيرة وموحدة للغاية، منفصلةً الذاكرة المنطقية كما يراها المبرمج عن الذاكرة الفعلية. تُحرر هذه التقنية المبرمجين من مخاوف قيود تخزين الذاكرة. كما تسمح الذاكرة الافتراضية للعمليات بمشاركة الملفات والمكتبات، وتطبيق الذاكرة المشتركة. بالإضافة إلى ذلك، توفر آلية فعالة لإنشاء العمليات. مع ذلك، ليس من السهل تطبيق الذاكرة الافتراضية، وقد تُخفض الأداء بشكل كبير إذا استُخدمت بشكل غير دقيق. في هذا الفصل، نقدم نظرة عامة مفصلة على الذاكرة الافتراضية، وندرس كيفية تطبيقها، ونستكشف تعقيداتها وفوائدها.

أهداف الوحدة

- تعريف الذاكرة الافتراضية واشرح فوائدها.
- توضيح كيفية تحميل الصفحات إلى الذاكرة باستخدام التصفيح حسب الطلب.
- تطبيق خوارزميات استبدال الصفحات FIFO، والأمثل، و.LRU

مدخل

تناولت الأقسام السابقة كيفية تجنب تجزئة الذاكرة عن طريق تقسيم متطلبات ذاكرة العملية إلى أجزاء أصغر (صفحات)، وتخزين هذه الصفحات بشكل غير متجاور في الذاكرة. مع ذلك، لا يزال يتعين تخزين العملية بأكملها في مكان ما في الذاكرة. عمليًا، لا تحتاج معظم العمليات الفعلية إلى جميع صفحاتها، أو على الأقل ليس جميعها دفعة واحدة، لعدة أسباب:

- لا حاجة إلى شيفرة معالجة الأخطاء Error handling code إلا في حال حدوث هذا الخطأ تحديدًا، وبعضها نادر جدًا.
- غالبًا ما تكون المصفوفات كبيرة الحجم في أسوأ الحالات، ولا يُستخدم منها عمليًا إلا جزء صغير.

- نادرًا ما تُستخدم بعض ميزات بعض البرامج، مثل روتين موازنة الميزانية الفيدرالية(-: .

للقدرة على تحميل أجزاء العمليات المطلوبة فعليًا فقط (وعند الحاجة إليها فقط) فوائد عديدة:

- يمكن كتابة البرامج لمساحة عنوان (مساحة ذاكرة افتراضية) أكبر بكثير من المساحة المتوفرة فعليًا على الحاسوب.
- لأن كل عملية تستخدم جزءًا صغيرًا فقط من مساحة عنوانها الإجمالية، يتبقى المزيد من الذاكرة للبرامج الأخرى، مما يُحسّن استخدام وحدة المعالجة المركزية (CPU) وإنتاجية النظام.
- يتطلب تبادل العمليات داخل وخارج ذاكرة الوصول العشوائي (RAM) كميات أقل من الإدخال/الإخراج، مما يُسرّع العمل.

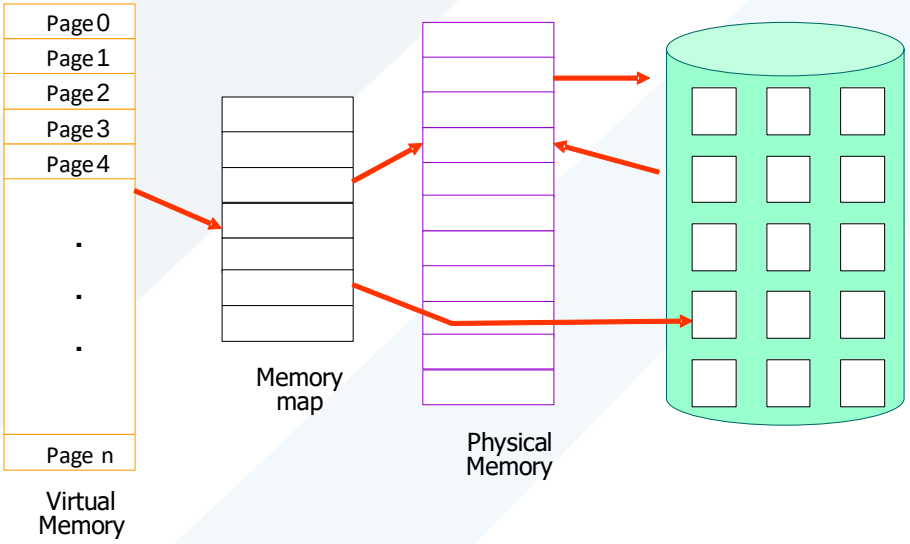

Figure 1 – Diagram showing virtual memory that is larger than physical memory

يوضح الشكل 1 التصميم العام للذاكرة الافتراضية، والتي قد تكون أكبر بكثير من الذاكرة الفعلية:

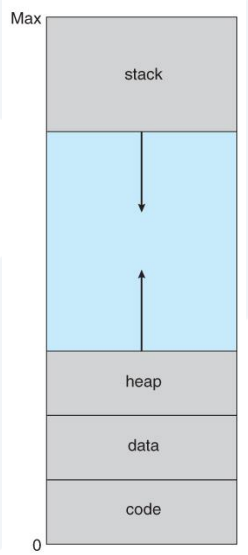

Figure 2 – Virtual address space

يوضح الشكل 2 مساحة العنوان الافتراضية, ***virtual address space***، وهي المنظور المنطقي للمبرمجين لتخزين ذاكرة العملية. يتم التحكم في التصميم الفعلي الفعلي من خلال جدول صفحات العملية لاحظ أن مساحة العنوان الموضحة في الشكل 2 متفرقة – لا تُستخدم أبدًا فجوة كبيرة في منتصف مساحة العنوان، إلا إذا نمت المكدس و/أو الكومة لملء هذه الفجوة. تتيح الذاكرة الافتراضية أيضًا مشاركة الملفات والذاكرة بواسطة عمليات متعددة، مع العديد من الفوائد:

- يمكن مشاركة مكتبات النظام عن طريق ربطها بمساحة العنوان الافتراضية لأكثر من عملية.
- يمكن للعمليات أيضًا مشاركة الذاكرة الافتراضية عن طريق ربط نفس كتلة الذاكرة بأكثر من عملية.
- يمكن مشاركة صفحات العملية أثناء استدعاء نظام( )fork ، مما يلغي الحاجة إلى نسخ جميع صفحات العملية الأصلية (الأصل).

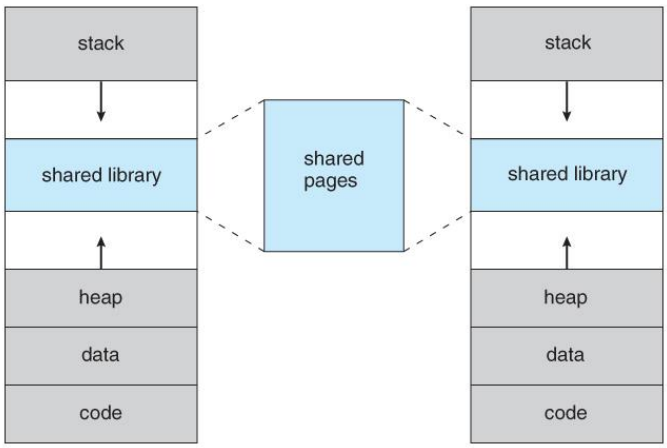

Figure 3 - Shared library using virtual memory

2 التصفيح عند الطلب Demand Paging

الفكرة الأساسية وراء ترحيل الطلب هي أنه عند تبديل عملية، لا تُستبدل صفحاتها دفعةً واحدة، بل تُستبدل فقط عند حاجتها إليها (عند الطلب). يُسمى هذا "مُبدّل كسول" ***lazy swapper***، مع أن مصطلح "متصفح" ***pager*** هو الأدق.

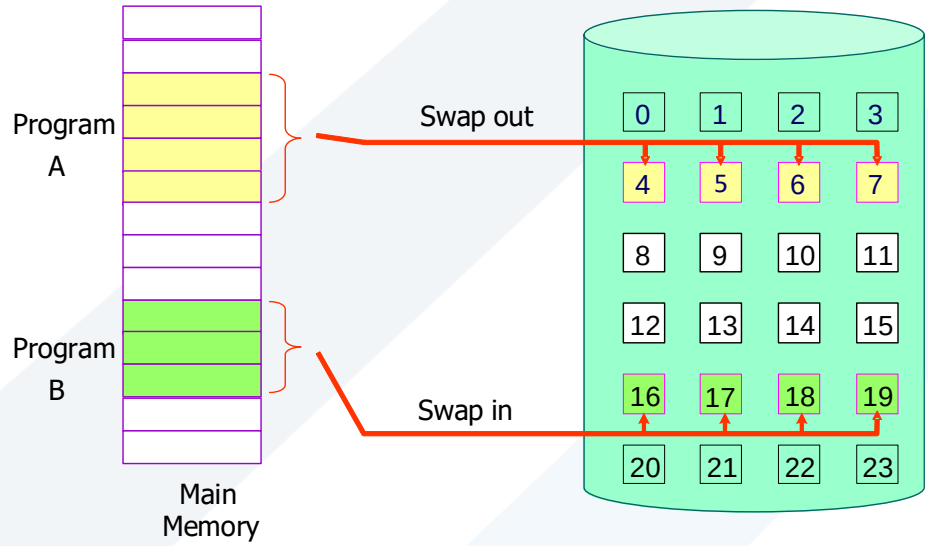

Figure 4 - Transfer of a paged memory to contiguous disk space

2.1 المفاهيم الأساسية

الفكرة الأساسية وراء ترحيل الطلب هي أنه عند تبديل عملية، يُحمّل مُبدّد الطلب إلى الذاكرة الصفحات التي يتوقع أن تحتاجها العملية فقط (فورًا).

تُعلّم الصفحات التي لم تُحمّل إلى الذاكرة على أنها غير صالحة في جدول الصفحات، باستخدام البت غير الصالح. (قد يكون باقي مُدخل جدول الصفحات إما فارغًا أو يحتوي على معلومات حول مكان الصفحة المُستبدلة على القرص الصلب).

إذا كانت العملية تصل فقط إلى الصفحات المُحمّلة في الذاكرة (الصفحات المُقيمة في الذاكرة) memory resident ، فستعمل العملية تمامًا كما لو أن جميع الصفحات قد حُمّلت إلى الذاكرة.

من ناحية أخرى، إذا كانت هناك حاجة لصفحة لم تُحمّل أصلاً، فسيتم إنشاء فخ خطأ صفحة page fault trap ، والذي يجب معالجته بسلسلة من الخطوات:

1. يتم أولاً التحقق من عنوان الذاكرة المطلوب، للتأكد من أنه طلب ذاكرة صالح.
2. إذا كان المرجع غير صالح، تُنهي العملية. وإلا، يجب ترحيل الصفحة.
3. يتم تحديد إطار حر، ربما من قائمة إطارات حرة.
4. تتم جدولة عملية قرص لإحضار الصفحة المطلوبة من القرص. (عادةً ما يؤدي هذا إلى حظر العملية في انتظار الإدخال/الإخراج، مما يسمح لعملية أخرى باستخدام وحدة المعالجة المركزية في هذه الأثناء).
5. عند اكتمال عملية الإدخال/الإخراج، يتم تحديث جدول صفحات العملية برقم الإطار الجديد، ويتم تغيير البت غير الصالح للإشارة إلى أن هذا أصبح الآن مرجع صفحة صالحًا.

6. يجب الآن إعادة تشغيل التعليمة التي تسببت في خطأ الصفحة من البداية (بمجرد تشغيل هذه العملية مرة أخرى على وحدة المعالجة المركزية).

في الحالات القصوى، لا تُستبدل أي صفحات بعملية ما حتى يتم طلبها بواسطة أخطاء الصفحة. يُعرف هذا باسم "الترحيل بناءً على الطلب" ***pure demand paging***. نظريًا، يمكن أن تُولد كل تعليمة أخطاء صفحات متعددة. عمليًا، هذا نادر جدًا، نظرًا لموقع المرجع ***locality of reference***.

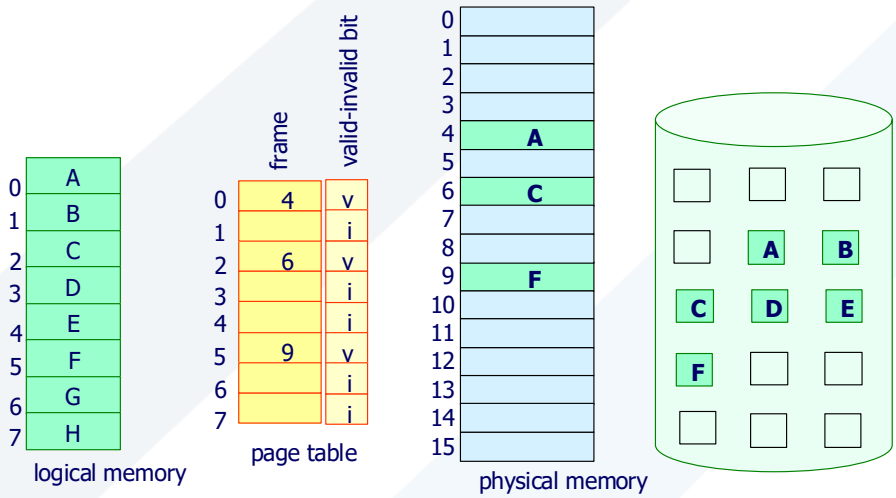

Figure 5 - Page table when some pages are not in main memory.

الأجهزة اللازمة لدعم الذاكرة الافتراضية هي نفسها المستخدمة في الترحيل والتبديل: جدول صفحات وذاكرة ثانوية. جزء أساسي من العملية هو إعادة تشغيل التعليمة من البداية بمجرد توفر الصفحة المطلوبة في الذاكرة. بالنسبة لمعظم التعليمات البسيطة، لا تُمثل هذه صعوبة كبيرة. ومع ذلك، هناك بعض البنى التي تسمح بتعليمة واحدة بتعديل كتلة بيانات كبيرة نسبيًا (قد تمتد عبر حدود الصفحة)، وإذا تم تعديل بعض البيانات قبل حدوث خطأ الصفحة، فقد يُسبب ذلك مشاكل. أحد الحلول هو الوصول إلى طرفي الكتلة قبل تنفيذ التعليمات، مما يضمن ترقيم الصفحات اللازمة قبل بدء التعليمات.

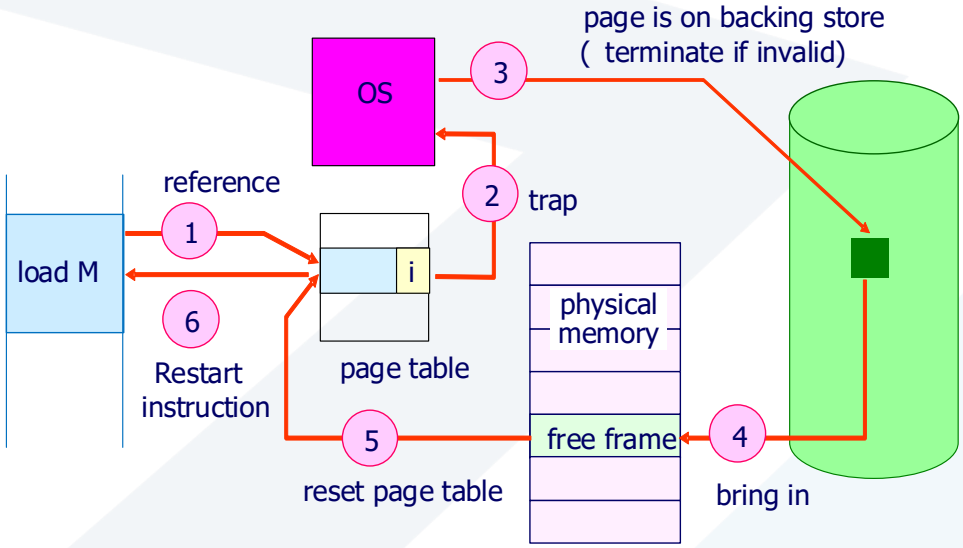

Figure 6 - Steps in handling a page fault

من الميزات الدقيقة أن مساحة التبديل أسرع في الوصول من نظام الملفات العادي، لأنها لا تحتاج إلى المرور عبر بنية الدليل بأكملها. لهذا السبب، تنقل بعض الأنظمة عمليةً كاملةً من نظام الملفات إلى مساحة المبادلة قبل بدء العملية، بحيث تتم جميع عمليات الترحيل اللاحقة من مساحة المبادلة الأسرع (نسبيًا). تستخدم بعض الأنظمة الترحيل عند الطلب مباشرةً من نظام الملفات للشيفرة الثنائية (التي لا تتغير أبدًا، وبالتالي لا يلزم تخزينها في عملية صفحة)، ولحجز مساحة المبادلة لأجزاء البيانات التي يجب تخزينها. يستخدم كلٌّ من Solaris وBSD Unix هذا النهج.

4 استبدال الصفحات

- لتحقيق أقصى استفادة من الذاكرة الافتراضية، نُحمّل عدة عمليات في الذاكرة في الوقت نفسه. ونظرًا لأننا نُحمّل فقط الصفحات التي تحتاجها كل عملية في أي وقت، فهناك مساحة كافية لتحميل عمليات أكثر بكثير مما لو اضطررنا لتحميل العملية بأكملها.
- مع ذلك، هناك حاجة للذاكرة أيضًا لأغراض أخرى (مثل تخزين الإدخال/الإخراج المؤقت)، فماذا يحدث إذا قررت إحدى العمليات فجأةً حاجتها إلى المزيد من الصفحات ولم تتوفر أي إطارات فارغة؟ هناك عدة حلول ممكنة للنظر فيها:
-
- تعديل الذاكرة المستخدمة في تخزين الإدخال/الإخراج المؤقت، وما إلى ذلك، لتحرير بعض الإطارات لعمليات المستخدم. يُعدّ قرار تخصيص الذاكرة لعمليات الإدخال/الإخراج مقابل عمليات المستخدم قرارًا معقدًا، مما ينتج عنه سياسات مختلفة على أنظمة مختلفة. (بعضها يُخصص مقدارًا ثابتًا لعمليات الإدخال/الإخراج، بينما يسمح البعض الآخر لنظام الإدخال/الإخراج بالتنافس على الذاكرة مع كل شيء آخر).
- وضع العملية التي تطلب المزيد من الصفحات في قائمة انتظار حتى تتوفر بعض الإطارات الفراغية.
- استبدل عملية ما من الذاكرة بالكامل، مما يُحرر إطارات صفحاتها.

- ابحث عن صفحة في الذاكرة غير مستخدمة حاليًا، واستبدلها بالقرص فقط، مما يُحرر إطارًا يُمكن تخصيصه للعملية التي تطلبها. يُعرف هذا باستبدال الصفحة، وهو الحل الأكثر شيوعًا. هناك العديد من الخوارزميات المختلفة لاستبدال الصفحات، وهو موضوع ما تبقى من هذا القسم.

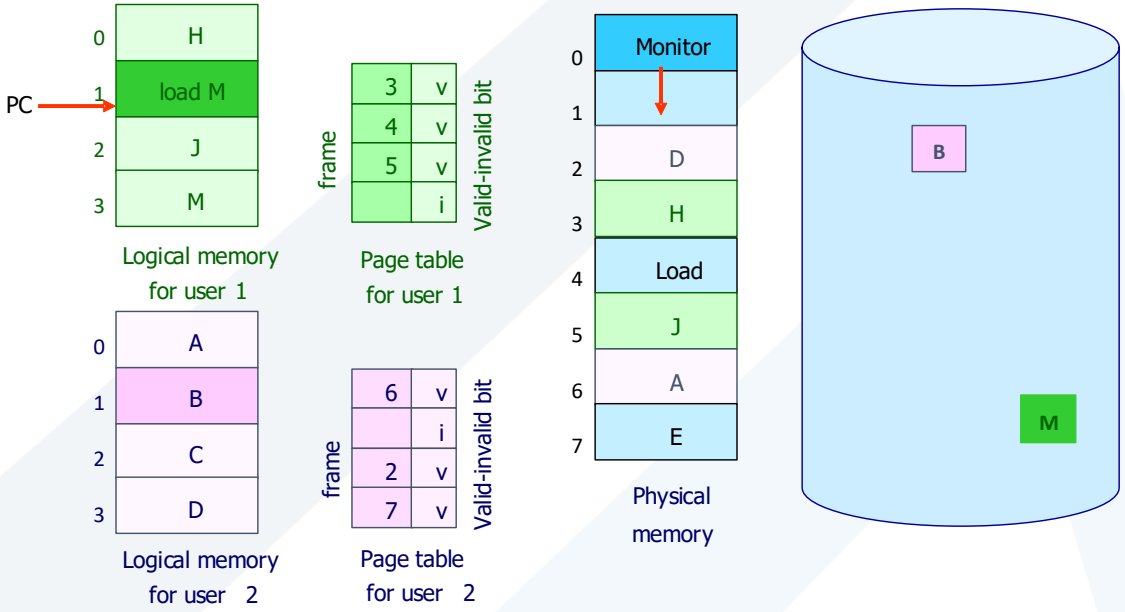

Figure 7- Need for page replacement.

4.1 استبدال الصفحات الأساسي

افترضت معالجة أخطاء الصفحات التي ناقشناها سابقًا وجود إطارات فارغة متاحة في قائمة الإطارات الفارغة. الآن، يجب تعديل معالجة أخطاء الصفحات لتحرير إطار عند الحاجة، كما يلي:

- ابحث عن موقع الصفحة المطلوبة على القرص، إما في مساحة المبادلة أو في نظام الملفات.
- ابحث عن إطار فارغ:
- إذا كان هناك إطار فارغ، فاستخدمه.
- إذا لم يكن هناك إطار فارغ، فاستخدم خوارزمية استبدال الصفحات لتحديد إطار موجود لاستبداله، يُعرف باسم الإطار المتضرر.
- اكتب الإطار المتضرر على القرص. غيّر جميع جداول الصفحات ذات الصلة للإشارة إلى أن هذه الصفحة لم تعد موجودة في الذاكرة.
- اقرأ الصفحة المطلوبة وخزّنها في الإطار. اضبط جميع جداول الصفحات والإطارات ذات الصلة للإشارة إلى التغيير.
- أعد تشغيل العملية التي كانت تنتظر هذه الصفحة.

لاحظ أن الخطوة 3ج تضيف كتابة إضافية على القرص إلى معالجة أخطاء الصفحات، مما يضاعف الوقت اللازم لمعالجة خطأ الصفحة. يمكن التخفيف من ذلك إلى حد ما عن طريق تعيين بت تعديل، أو بت متسخ لكل صفحة،

للإشارة إلى ما إذا كانت قد تغيرت منذ آخر تحميل لها من القرص. إذا لم يتم تعيين بت المتسخ، فإن الصفحة لم تتغير، ولا تحتاج إلى كتابتها على القرص. وإلا، فإن كتابة الصفحة مطلوبة. لا ينبغي أن يكون مفاجئًا أن العديد من استراتيجيات استبدال الصفحات تبحث تحديدًا عن الصفحات التي لم يتم تعيين بت المتسخ الخاص بها، وتفضل اختيار الصفحات النظيفة كصفحات ضحية. يجب أن يكون واضحًا أيضًا أن صفحات التعليمات البرمجية غير القابلة للتعديل لا يتم تعيين بت المتسخ الخاص بها أبدًا.

هناك متطلبان رئيسيان لتنفيذ نظام ترحيل طلب ناجح. يجب علينا تطوير خوارزمية تخصيص الإطارات frame–allocation algorithm وخوارزمية استبدال الصفحة page–replacement algorithm. يتمحور الأول حول عدد الإطارات المخصصة لكل عملية (وللاحتياجات الأخرى)، ويتعامل الأخير مع كيفية اختيار صفحة للاستبدال عندما لا تتوفر إطارات مجانية.

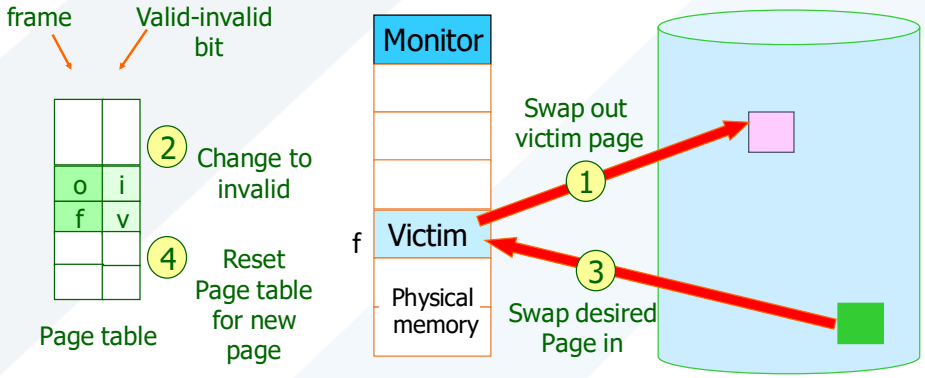

Figure 8 - Page replacement.

الهدف العام من اختيار هذه الخوارزميات وضبطها هو توليد أقل عدد من أخطاء الصفحات الإجمالية. نظرًا لبطء الوصول إلى القرص مقارنةً بالوصول إلى الذاكرة، فإن أي تحسينات طفيفة على هذه الخوارزميات يمكن أن تُحقق تحسينات كبيرة في الأداء العام للنظام.

تُقيّم الخوارزميات باستخدام سلسلة مُحددة من عمليات الوصول إلى الذاكرة تُعرف باسم السلسلة المرجعية **reference string** ، والتي يُمكن توليدها بإحدى ثلاث طرق شائعة (على الأقل):

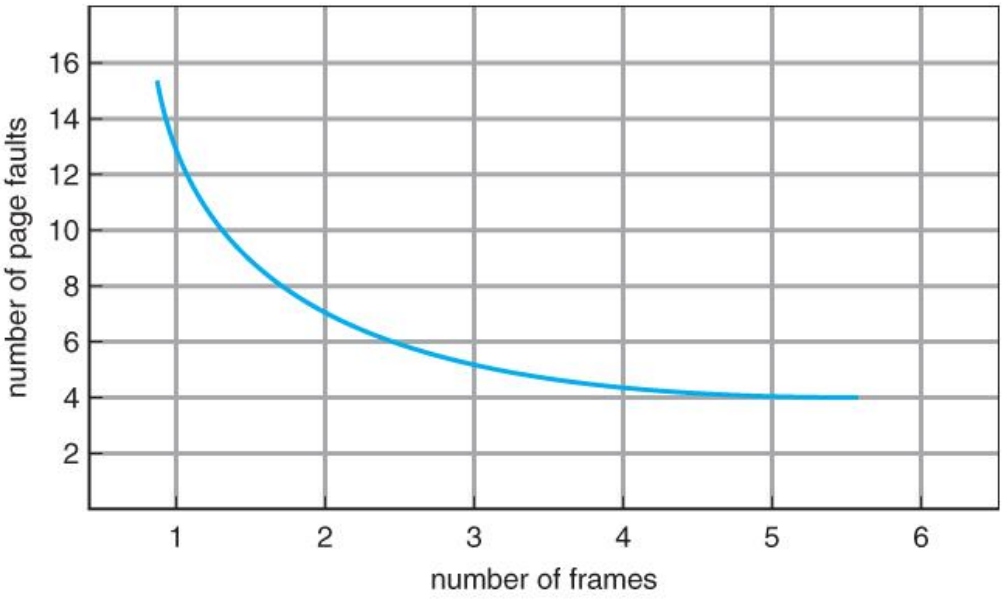

Figure 9 - Graph of page faults versus number of frames.

- توليد عشوائي، إما بتوزيع متساوٍ أو بمنحنى توزيع معين بناءً على سلوك النظام المُلاحظ. يُعد هذا النهج الأسرع والأسهل، ولكنه قد لا يعكس الأداء الفعلي بشكل جيد، لأنه يتجاهل موقع المرجع.

4.2 استبدال الصفحات باستخدام أسلوب FIFO

تُعدّ استراتيجية استبدال الصفحات البسيطة والواضحة هي FIFO، أي "الأول في الدخول أولًا في الخروج".

عند إدخال صفحات جديدة، تُضاف إلى ذيل قائمة الانتظار، وتكون الصفحة التي في أعلى قائمة الانتظار هي الضحية التالية. في المثال التالي، ينتج عن 20 طلبًا للصفحات 15 خطأً في الصفحة:

على الرغم من بساطة أسلوب FIFO وسهولة تطبيقه، إلا أنه ليس دائمًا الأمثل، أو حتى الفعّال.

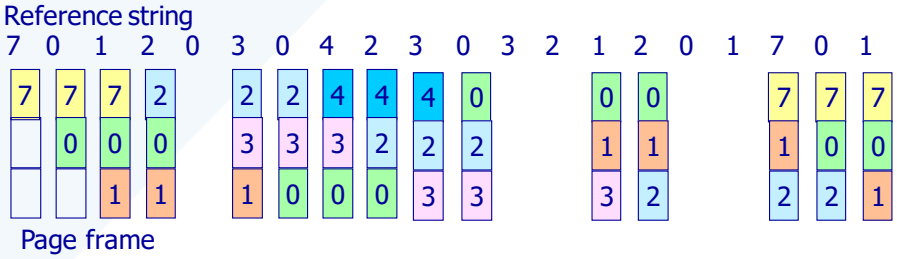

Figure 10 - FIFO page-replacement algorithm.

من التأثيرات المثيرة للاهتمام التي قد تحدث مع أسلوب FIFO شذوذ Belady، حيث أن زيادة عدد الإطارات المتاحة قد تؤدي في الواقع إلى زيادة عدد أخطاء الصفحات التي تحدث! على سبيل المثال، انظر إلى الرسم البياني التالي المستند إلى تسلسل الصفحات (1، 2، 3، 4، 1، 2، 5، 1، 2، 3، 4، 5) وعدد متفاوت من الإطارات المتاحة. من

الواضح أن الحد الأقصى لعدد الأخطاء هو ١٢ (كل طلب يُنتج خطأً)، والحد الأدنى هو ٥ (كل صفحة تُحمّل مرة واحدة فقط)، ولكن بين هذين الحدين، توجد بعض النتائج المثيرة للاهتمام:

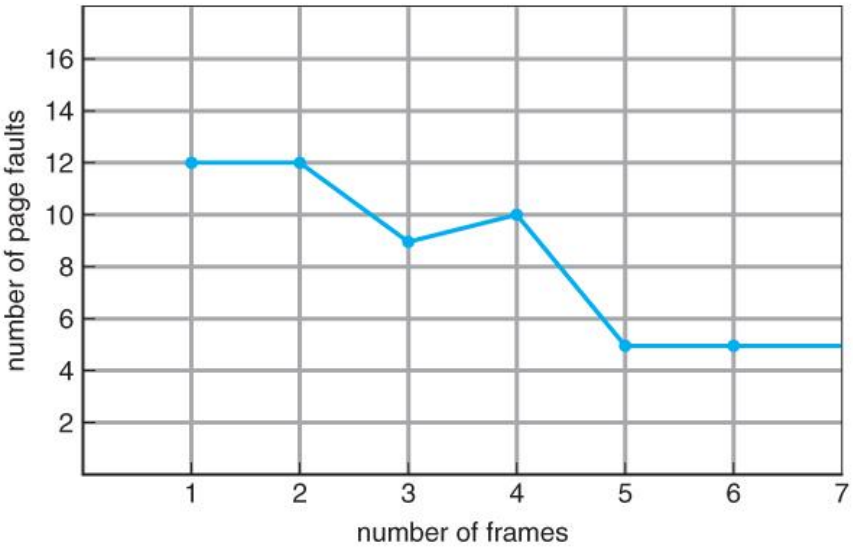

Figure 11 - Page-fault curve for FIFO replacement on a reference string.

٩,٤,٣ الاستبدال الأمثل للصفحات Optimal Page Replacement

أدى اكتشاف شذوذ بيلادي Belady's anomaly إلى البحث عن خوارزمية استبدال مثالية للصفحات، وهي ببساطة الخوارزمية التي تُنتج أقل عدد ممكن من أخطاء الصفحات، والتي لا تُعاني من شذوذ بيلادي.

توجد مثل هذه الخوارزمية، وتُسمى OPT أو MIN. هذه الخوارزمية ببساطة هي "استبدال الصفحة التي لن تُستخدم لأطول فترة مُستقبلية".

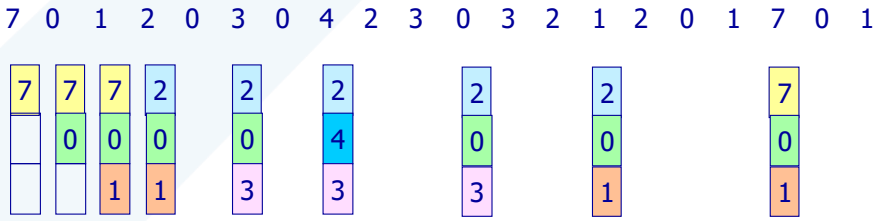

Figure 12 - Optimal page-replacement algorithm

على سبيل المثال، يوضح الشكل 12 أنه بتطبيق OPT على نفس سلسلة المرجع المستخدمة في مثال FIFO، فإن الحد الأدنى لعدد أخطاء الصفحة المحتملة هو ونظرًا لأن 6 من أخطاء الصفحة لا يمكن تجنبها (المرجع الأول لكل صفحة جديدة)، يمكن إظهار أن FIFO تتطلب 3 أضعاف عدد أخطاء الصفحة (الإضافية) مقارنة بالخوارزمية المثلى. (ملاحظة: يدعي الكتاب أن أخطاء الصفحة الثلاثة الأولى فقط مطلوبة من قبل جميع الخوارزميات، مما يشير إلى أن FIFO أسوأ بمرتين فقط من OPT.)

لسوء الحظ، لا يمكن تنفيذ OPT عمليًا، لأنه يتطلب التنبؤ بالمستقبل، ولكنه يمثل معيارًا جيدًا لمقارنة وتقييم الخوارزميات الجديدة المقترحة الفعلية.

عمليًا، تحاول معظم خوارزميات استبدال الصفحات تقريب OPT من خلال التنبؤ (التقدير) بطريقة أو بأخرى بالصفحة التي لن تُستخدم لأطول فترة زمنية. يعتمد أساس خوارزمية FIFO على التنبؤ بأن الصفحة التي تم جلبها منذ فترة طويلة هي الصفحة التي لن تكون هناك حاجة إليها مجددًا في المستقبل القريب. ولكن كما سنرى، هناك العديد من طرق التنبؤ الأخرى، وكلها تسعى جاهدة لمضاهاة أداء خوارزمية OPT.

4.4 استبدال الصفحة LRU

يعتمد التنبؤ وراء خوارزمية LRU، أو "الأقل استخدامًا مؤخرًا"، على أن الصفحة التي لم تُستخدم منذ فترة طويلة هي الصفحة التي لن تُستخدم مجددًا في المستقبل القريب. (لاحظ الفرق بين FIFO و LRU: فالأولى تنظر إلى أقدم وقت تحميل، والثانية تنظر إلى أقدم وقت استخدام).

يرى البعض أن LRU تُشبه OPT، إلا أنها تنظر إلى الماضي بدلًا من المستقبل. (تتميز OPT بخاصية مثيرة للاهتمام، وهي أنه بالنسبة لأي سلسلة مرجعية S وعكسها R، ستُولّد OPT نفس عدد أخطاء الصفحات لكل من S و R. وقد اتضح أن LRU تتمتع بنفس الخاصية.)

يوضح الشكل 15 LRU لسلسلة العينة لدينا، مما يُنتج 12 خطأً في الصفحات (مقارنةً بـ 15 خطأً في FIFO و9 أخطاءً في OPT).

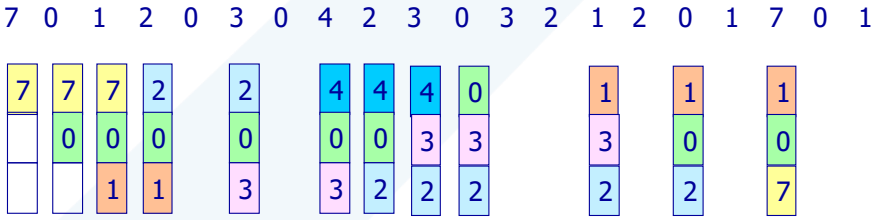

Figure 13 - LRU page-replacement algorithm.