جامعة المنارة

كلية: ....الهندسة.............

قسم: .... الهندسة المعلوماتية...........................

اسم المقرر: .... نظم تشغيل 2..........................

رقم الجلسة (...5...)

عنوان الجلسة

....طرق إدارة الذاكرة......

م.عمار مصطفى

جامعة

الفصل الدراسي

العام الدراسي 2024 /2025

جامعة المنارة
MANARA UNIVERSITY

جدول المحتويات

Contents

رقم الصفحة	العنوان
	البنية الهرمية للذاكرة
	الغاية من ادارة الذاكرة
	مساحة العنوان المنطقية مقابل المساحة الفيزيائية
	وحدة إدارة الذاكرة (MMU)
	الحماية Protection
	التقسيم
	التخصيص المتجاور Contiguous Allocation
	المبادلة Swapping
	الترحيل Paging

جامعة المنارة

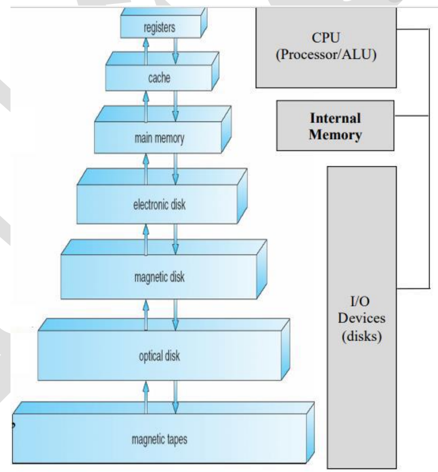
الغاية من الجلسة: شرح الطرق المختلفة لتنظيم الذاكرة الفيزيائية، شرح حالات إدارة الذاكرة المختلفة، شرح التقنيات المستخدمة في إدارة الذاكرة (التقسيم partitions - المبادلة swapping - الترحيل paging)

1- البنية الهرمية للذاكرة:

يمكن لوحدة المعالجة المركزية الوصول المباشر الى الذاكرة الرئيسية والمسجلات فقط، لذلك يجب إحضار البرامج والبيانات من القرص الى الذاكرة.

يتضمن التسلسل الهرمي للذاكرة:

- ذاكرة التخزين المؤقت cash
- الذاكرة الرئيسية: متوسطة السرعة ليست باهظة الزمن
- الأقراص الصلبة (الالكترونية والمغناطيسية و الضوئية ...)، حجم كبير، بطيء، رخيص، تخزين مستمر



2- الغاية من ادارة الذاكرة:

- الاحتفاظ بعمليات متعددة بالذاكرة لتحسين عمل وحدة المعالجة المركزية.
- إدارة وحماية الذاكرة الرئيسية أثناء مشاركتها في عمليات متعددة
- الحماية، لا يستطيع كل برنامج الوصول إلى ذاكرة الآخرين، بل يصل فقط الى المواقع التي خصصت لها.
- المشاركة: السماح لعدة عمليات بالوصول الى نفس الذاكرة المشتركة.

3- مساحة العنوان المنطقية مقابل المساحة الفيزيائية:

إن مفهوم مساحة العنوان المنطقية المرتبطة بمساحة عنوان فيزيائية منفصلة يشكل عنصرًا أساسيًا في إدارة الذاكرة بشكل صحيح

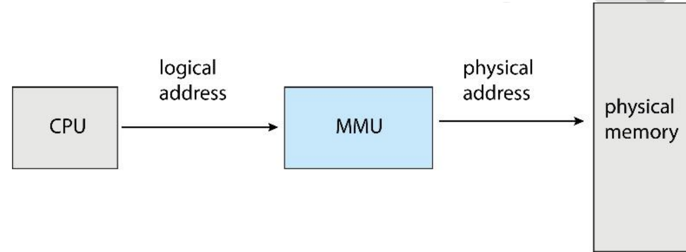
- العنوان المنطقي Logical address - الذي يتم إنشاؤه بواسطة وحدة المعالجة المركزية؛ ويشار إليه أيضًا باسم العنوان الافتراضي
- العنوان المادي Physical address - العنوان الذي تراه وحدة الذاكرة

العناوين المنطقية والمادية هي نفسها في مخططات ربط العناوين في وقت التجميع ووقت التحميل؛ وتختلف العناوين المنطقية (الافتراضية) والمادية في مخطط ربط العناوين في وقت التنفيذ

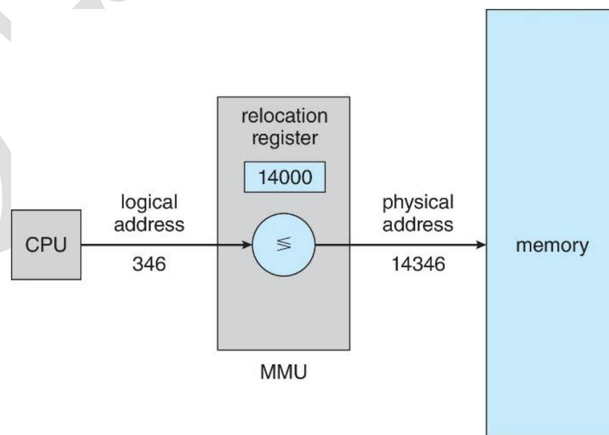
- مساحة العنوان المنطقية Logical address space هي مجموعة جميع العناوين المنطقية التي يتم إنشاؤها بواسطة برنامج
- مساحة العنوان المادي Physical address space هي مجموعة جميع العناوين المادية التي يتم إنشاؤها بواسطة برنامج

4- وحدة إدارة الذاكرة (MMU):

جهاز مادي يقوم في وقت التشغيل بربط العنوان الافتراضي بالعناوين المادية.

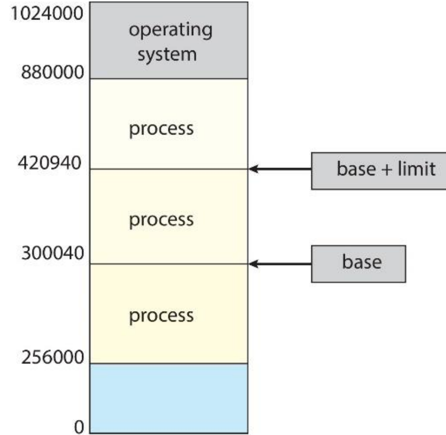


- السجل الأساسي base يسمى الآن سجل إعادة التوطين relocate able
- تضاف القيمة في سجل إعادة التوطين إلى كل عنوان يتم إنشاؤه بواسطة عملية مستخدم في وقت إرساله إلى الذاكرة
- يتعامل برنامج المستخدم مع العناوين المنطقية؛ ولا يرى العناوين المادية الحقيقية أبدًا
- يحدث الربط وقت التنفيذ عند الإشارة إلى الموقع في الذاكرة



5- الحماية Protection:

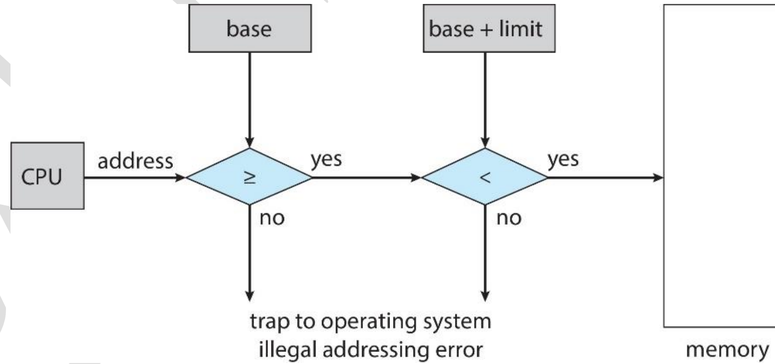
يجب التأكد من أن العملية يمكنها الوصول فقط إلى تلك العناوين الموجودة في مساحة العنوان الخاصة بها. يمكننا توفير هذه الحماية باستخدام زوج من السجلات الأساسية base وlimit registers لتحديد مساحة العنوان المنطقية للعملية



يجب على وحدة المعالجة المركزية التحقق من كل وصول إلى الذاكرة يتم إنشاؤه في وضع المستخدم للتأكد من أنه بين القاعدة والحد الأقصى لهذا المستخدم

5-1- حماية عنوان الأجهزة Hardware :

التعليمات الخاصة بتحميل السجلات الأساسية والحدشية هي تعليمات ذات امتياز



يمكن أن يحدث ربط عناوين التعليمات والبيانات بعناوين الذاكرة في ثلاث مراحل مختلفة

- ✓ وقت التجميع Compile time: إذا كان موقع الذاكرة معروفًا مسبقًا، فيمكن إنشاء كود مطلق؛ يجب إعادة تجميع الكود إذا تغير موقع البدء
- ✓ وقت التحميل Load time: يجب إنشاء كود قابل للنقل إذا لم يكن موقع الذاكرة معروفًا في وقت التجميع

- ✓ وقت التنفيذ Execution time: يتأخر الربط حتى وقت التشغيل إذا كان من الممكن نقل العملية أثناء تنفيذها من جزء ذاكرة إلى آخر

تحتاج إلى دعم الأجهزة لخرائط العناوين (على سبيل المثال، سجلات القاعدة والحد)

6- التقسيم:

يتم هنا تقسيم الذاكرة الى أقسام وتخصيص أقسام محددة من الذاكرة لعمليات مختلفة، يوجد عدة أنواع للتقسيم

6-1- التقسيم الثابت

- أقسام متساوية الحجم
- أي عملية حجمها أقل من أو يساوي حجم القسم يمكن تحميلها على قسم متاح

مشاكل التقسيم الثابتة

- قد لا يتناسب حجم البرنامج على القسم
- استخدام الذاكرة غير فعال
- أي برنامج مهما كان صغيرا، يشغل مساحة قسم كامل من الذاكرة
- يؤدي الى توليد الفراغات ضمن الأقسام

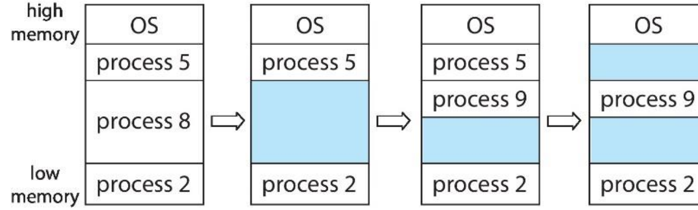
6-2- التقسيم الديناميكي أو المرن:

- تقسيم غير متساوي الحجم
- الحد من المشاكل السابقة دون حل كامل
- يمكن وضع البرامج الصغيرة في أماكن أصغر والحد من الفراغات الداخلية.

6-3- التقسيم المتغير Variable Partition:

- تخصيص أقسام متعددة
- درجة البرمجة المتعددة محدودة بعدد الأقسام
- أحجام أقسام متغيرة للكفاءة (بحجم يتناسب مع احتياجات عملية معينة)
- الثقب - كتلة من الذاكرة المتاحة؛ تنتشر الثقوب ذات الأحجام المختلفة في جميع أنحاء الذاكرة
- عندما تصل عملية، يتم تخصيص ذاكرة لها من ثقب كبير بما يكفي لاستيعابها
- تحرر العملية الخارجة قسمها، ويتم دمج الأقسام الحرة المجاورة
- يحتفظ نظام التشغيل بالمعلومات حول: أ) الأقسام المخصصة ب) الأقسام الحرة (الثقب)

جامعة المنارة



7- التخصيص المتجاور Contiguous Allocation:

- يجب أن تدعم الذاكرة الرئيسية كلًا من نظام التشغيل وعمليات المستخدم
- مورد محدود، يجب تخصيصه بكفاءة
- عادةً ما تنقسم الذاكرة الرئيسية إلى قسمين:
- نظام التشغيل المقيم، والذي يتم الاحتفاظ به عادةً في ذاكرة منخفضة مع متجه مقاطعة
- عمليات المستخدم يتم الاحتفاظ بها بعد ذلك في ذاكرة عالية
- كل عملية موجودة في قسم متجاور واحد من الذاكرة
- مسجلات النقل المستخدمة لحماية عمليات المستخدم من بعضها البعض، ومن تغيير كود نظام التشغيل والبيانات
- يحتوي المسجل الأساسي base على قيمة أصغر عنوان فعلي
- يحتوي المسجل الحدي limit على نطاق من العناوين المنطقية - يجب أن يكون كل عنوان منطقي أقل من سجل الحد
- تقوم وحدة إدارة الذاكرة بتعيين العنوان المنطقي ديناميكيًا يمكن بعد ذلك السماح بإجراءات مثل أن يكون كود النواة مؤقتًا وحجم النواة المتغير

هناك أربع طرق لإدارة الذاكرة في أنظمة التشغيل المختلفة :

1. التعيين المفرد المتجاور :Single contiguous allocatoin وهذه هي أبسط طريقة لتعيين الأقسام وتستخدم في نظام .MS-DOS وتكون الذاكرة بأكملها متاحة للعملية قيد التنفيذ باستثناء جزء من الذاكرة يكون محجوزًا من قبل النظام.
2. التعيين المجزّء :Partitioned allocation تقسّم الذاكرة في هذه الطريقة إلى كتل blocks أو أجزاء partitions مختلفة، وتحجز كل عملية ما تحتاج إليه من هذه الأجزاء.
3. إدارة الذاكرة بتقسيمها إلى صفحات :Paged memory management تقسّم الذاكرة في هذه الطريقة إلى وحدات ذات حجم ثابت تدعى بإطارات الصفحات page frames، وتستخدم هذه الطريقة في بيئات الذاكرة الافتراضية.virtual memory environment
4. إدارة الذاكرة المجزّأة :Segmented memory management تقسّم الذاكرة في هذه الطريقة إلى أجزاء segments مختلفة (الجزء هو تجمّع منطقي للبيانات أو الشيفرات الخاصة بالعملية)، ولا يشترط تجاور الذاكرة المحجوزة في هذه الطريقة.

8- مشكلة تخصيص التخزين الديناميكي Dynamic Storage-Allocation Problem:

كيف يتم تلبية طلب بحجم n من قائمة من الثقوب المجانية؟ يتم ذلك من خلال مجموعة من الخوارزميات

جامعة المَنارة
MANARA UNIVERSITY

1-8-خوارزمية الملاءمة الأولى First-fit:

تحجز هذه الطريقة أول كتلة ذات مساحة كافية من قمّة الذاكرة الرئيسية .

تتبع خوارزمية الملاءمة الأولى الخطوات التالية :

1. تُغذّى الخوارزمية بكتل الذاكرة المتاحة مع حجم كل كتلة إضافة إلى العمليات مع حجم كل عملية.
2. تهيّئ الخوارزمية بعدها جميع كتل الذاكرة لتكون شاغرة.
3. تبدأ الخوارزمية باختيار كل عملية والتحقق من إمكانية إسنادها إلى الكتلة الحالية.
4. إن كان حجم العملية أصغر من حجم الكتلة أو مساويًا له، تُسند العملية إلى تلك الكتلة وتنتقل الخوارزمية إلى العملية التالية.
5. تستمر الخوارزمية في التحقق من الكتل الباقية إن لم يتحقق الشرط في الخطوة السابقة.

يحاكي هذا الكود المكتوب بلغة C++ خوارزمية تخصيص الذاكرة حسب الأولوية. ويستخدم قائمة std::list لتمثيل كتل الذاكرة، مما يجعل عملية الإدخال والحذف فعالة. وهذا نموذج مبسط؛ أما النظام في العالم الحقيقي فسيكون أكثر تعقيدًا.

```
3  #include <iostream>
 4  #include <list>
 5
 6  struct MemoryBlock {
 7      int size;
 8      int address;
 9      bool allocated;
10
11      MemoryBlock(int s, int a) : size(s), address(a), allocated(false) {}
12  };
13
14  // Function to find and allocate a block using the first-fit algorithm
15  MemoryBlock* firstFit(std::list<MemoryBlock>& memory, int requestSize) {
16      for (auto it = memory.begin(); it != memory.end(); ++it) {
17          if (!it->allocated && it->size >= requestSize) {
18              // Found a suitable block
19              if (it->size == requestSize) {
20                  //Exact fit
21                  it->allocated = true;
22                  return &(*it);
23              } else {
24                  // Split the block
25                  int remainingSize = it->size - requestSize;
26                  MemoryBlock* allocatedBlock = &(*it);
27                  allocatedBlock->size = requestSize;
28                  allocatedBlock->allocated = true;
29                  memory.insert(++it, MemoryBlock(remainingSize, allocatedBlock->address +
requestSize));
30                  return allocatedBlock;
31              }
32          }
33      }
34      return nullptr; // No suitable block found
35  }
36
37
38  int main() {
39      // Initialize memory (Simulate available memory blocks)
40      std::list<MemoryBlock> memory;
41      memory.emplace_back(10, 0); //Block of size 10 starting at address 0
42      memory.emplace_back(15, 10); //Block of size 15 starting at address 10
43      memory.emplace_back(5, 25); //Block of size 5 starting at address 25
44      memory.emplace_back(20, 30); //Block of size 20 starting at address 30
```

```
45
46
47      int request1 = 7;
48      MemoryBlock* allocated1 = firstFit(memory, request1);
49      if (allocated1) {
50          std::cout << "Request 1 (" << request1 << ") allocated at address " << allocated1->address << ".\n";
51      } else {
52          std::cout << "Request 1 (" << request1 << ") failed.\n";
53      }
54
55      int request2 = 12;
56      MemoryBlock* allocated2 = firstFit(memory, request2);
57      if (allocated2) {
58          std::cout << "Request 2 (" << request2 << ") allocated at address " << allocated2->address << ".\n";
59      } else {
60          std::cout << "Request 2 (" << request2 << ") failed.\n";
61      }
62
63
64      // Print the memory status
65      std::cout << "\nMemory Status:\n";
66      for (const auto& block : memory) {
67          std::cout << "Address: " << block.address << ", Size: " << block.size << ", Allocated: " << (block.allocated ? "true" : "false") << "\n";
68      }
69
70      return 0;
71  }
```

1. هياكل البيانات:

بنية MemoryBlock: تمثل كتلة من الذاكرة. وهي تخزن الحجم وعنوان البداية وعلم flag مخصص للإشارة إلى ما إذا كانت الكتلة قيد الاستخدام حاليًا.

2. خوارزمية First-Fit (دالة FirstFit):

الإدخال: قائمة std:: من كائنات MemoryBlock تمثل الذاكرة المتاحة وحجم الطلب يمثل حجم كتلة الذاكرة المطلوبة.

المنطق:

يتكرر خلال قائمة كائنات MemoryBlock.

بالنسبة لكل كتلة، يتحقق مما إذا كانت الكتلة غير مخصصة (!it->allocated) وما إذا كان حجمها أكبر من أو يساوي الحجم المطلوب (it->size >= requestSize).

إذا تم العثور على كتلة مناسبة:

- الملاءمة الدقيقة: إذا كان حجم الكتلة مساويًا تمامًا للحجم المطلوب، يتم تعيين علم الكتلة المخصص على true، ويتم إرجاع مؤشر إلى الكتلة.
- التقسيم: إذا كان حجم الكتلة أكبر من الحجم المطلوب، يتم تقسيم الكتلة إلى قسمين:

- يتم تخصيص الجزء الأول للطلب، ويتم تحديث حجمه، ويتم تعيين علم التخصيص الخاص به على true.
- يتم إنشاء كتلة ذاكرة جديدة للجزء غير المخصص المتبقي وإدراجه في القائمة فورًا بعد الكتلة المخصصة. يتم حساب عنوان الكتلة الجديدة بناءً على عنوان الكتلة المخصصة وحجمها.
- يتم إرجاع مؤشر إلى الجزء المخصص.
- الإرجاع: إذا لم يتم العثور على كتلة مناسبة بعد التكرار عبر القائمة بالكامل، تعيد الدالة nullptr.

3. الدالة الرئيسية (main):

التهيئة: تنشئ قائمة std::list تسمى memory وتقوم بتهيئتها بأربعة كائنات MemoryBlock، ومحاكاة كتل الذاكرة المتاحة بأحجام وعناوين بداية مختلفة.

*(تنتهي مقتطفات التعليمات البرمجية المقدمة هنا. في البرنامج الكامل، ستكون الخطوات التالية هي استدعاء الدالة firstFit لطلب الذاكرة ثم التعامل مع المؤشر المسترجع.)

2-8-خوارزمية الملاءمة الفضلى Best-fit

تحجز هذه الطريقة أول كتلة ذات أصغر مساحة كافية من بين الكتل المتوفرة في الذاكرة الرئيسية.

خطوات الخوارزمية

1. تُغذّى الخوارزمية بكتل الذاكرة المتاحة مع حجم كل كتلة إضافة إلى العمليات مع حجم كل عملية.
2. تهيّئ الخوارزمية بعدها جميع كتل الذاكرة لتكون شاغرة.
3. تبدأ الخوارزمية باختيار كلّ عملية والبحث عن الكتلة ذات الحجم الأصغر والتي يمكن تعيينها إلى العملية الحالية، فإن وجدتها عيّنتها إلى العملية الحالية.
4. إن لم تعثر الخوارزمية على الكتلة المطلوبة تجاوزت تلك العملية وانتقلت إلى العملية التي تليها.

```
#include <iostream>
 2  #include <vector>
 3
 4  struct Block {
 5      int size; // Size of the block
 6      bool isFree; // Status of the block (free or allocated)
 7  };
 8
 9  class BestFitAllocator {
10  private:
11      std::vector<Block> blocks; // Vector to store memory blocks
12
13  public:
14      BestFitAllocator(const std::vector<int>& sizes) {
15          for (int size : sizes) {
16              blocks.push_back({size, true}); // Initialize blocks as free
17          }
18      }
19
20      void allocate(int requestSize) {
21          int bestIndex = -1;
22          for (int i = 0; i < blocks.size(); i++) {
23              if (blocks[i].isFree && blocks[i].size >= requestSize) {
24                  if (bestIndex == -1 || blocks[i].size < blocks[bestIndex].size) {
```

```
25                        bestIndex = i; // Find the best fit
26                    }
27                }
28            }
29
30            if (bestIndex != -1) {
31                blocks[bestIndex].isFree = false; // Allocate the block
32                std::cout << "Allocated " << requestSize << " units in block of size " <<
blocks[bestIndex].size << std::endl;
33            } else {
34                std::cout << "No suitable block found for " << requestSize << " units." <<
std::endl;
35            }
36        }
37
38        void deallocate(int blockIndex) {
39            if (blockIndex >= 0 && blockIndex < blocks.size() &&
!blocks[blockIndex].isFree) {
40                blocks[blockIndex].isFree = true; // Free the block
41                std::cout << "Deallocated block of size " << blocks[blockIndex].size <<
std::endl;
42            } else {
43                std::cout << "Invalid block index or block already free." << std::endl;
44            }
45        }
46
47        void printBlocks() {
48            for (int i = 0; i < blocks.size(); i++) {
49                std::cout << "Block " << i << ": Size = " << blocks[i].size << ", Status =
" << (blocks[i].isFree ? "Free" : "Allocated") << std::endl;
50            }
51        }
52    };
53
54    int main() {
55        BestFitAllocator allocator({100, 500, 200, 300, 600}); // Initialize with block
sizes
56        allocator.printBlocks();
57
58        allocator.allocate(212); // Request allocation
59        allocator.allocate(417); // Request allocation
60        allocator.allocate(112); // Request allocation
61
62        allocator.printBlocks(); // Print current block statuses
63
64        allocator.deallocate(1); // Deallocate a block
65        allocator.printBlocks(); // Print current block statuses after deallocation
66
67        return 0;
68    }
```

يحدد هذا الهيكل بنية بيانات بسيطة تسمى Block والتي تحتوي على عنصرين:

- الحجم size: عدد صحيح يمثل حجم كتلة الذاكرة.
- isFree: قيمة منطقية تشير إلى ما إذا كانت الكتلة حرة (true) أو مخصصة (false
- تدير فئة BestFitAllocator كتل الذاكرة باستخدام متجه خاص من هياكل الكتلة تسمى الكتل.
- تحتوي على طرق عامة لتخصيص الذاكرة وإلغاء التخصيص وعرض حالة كتل الذاكرة.

جامعة المنارة
MANARA UNIVERSITY

• يأخذ هذا المنشئ متجهًا من الأعداد الصحيحة (الأحجام) التي تمثل أحجام كتل الذاكرة المراد تخصيصها.

• يقوم بتهيئة كل كتلة على أنها حرة عن طريق دفع مثيل كتلة بالحجم المحدد وتعيين isFree على true.

• تحاول هذه الطريقة تخصيص كتلة ذاكرة بحجم requestSize.

• تقوم بتهيئة bestIndex إلى -1 لتتبع مؤشر الكتلة الأكثر ملاءمة.

• تتكرر الطريقة عبر جميع كتل الذاكرة:

• تتحقق مما إذا كانت الكتلة حرة ولها حجم أكبر من أو يساوي requestSize.

• إذا كان الأمر كذلك، فإنها تقارن الأحجام للعثور على أصغر كتلة مناسبة (أفضل ملاءمة).

• إذا تم العثور على كتلة مناسبة، فإنها تضع علامة عليها على أنها مخصصة عن طريق تعيين isFree على false وتطبع تفاصيل التخصيص. إذا لم يتم العثور على كتلة مناسبة، فإنها تُعلم المستخدم.

• تحرر هذه الطريقة كتلة تم تخصيصها مسبقًا عند blockIndex المعطى.

• تتحقق مما إذا كان الفهرس صالحًا وما إذا كانت الكتلة مخصصة حاليًا.

• إذا كان صالحًا، فإنها تضع علامة على الكتلة على أنها خالية وتطبع رسالة إلغاء التخصيص. إذا كان الفهرس غير صالح أو كانت الكتلة خالية بالفعل، فإنها تقدم رسالة خطأ.

• تتكرر هذه الطريقة عبر جميع الكتل وتطبع أحجامها وحالاتها (خالية أو مخصصة) على وحدة التحكم، مما يوفر نظرة عامة واضحة على حالة تخصيص الذاكرة.

• في الوظيفة الرئيسية:

• يتم إنشاء مثيل لـ BestFitAllocator بأحجام كتل محددة مسبقًا.

• تتم طباعة الحالة الحالية لكتل الذاكرة.

• يتم تقديم ثلاثة طلبات تخصيص لأحجام مختلفة.

• بعد محاولات التخصيص، تتم طباعة حالات كتلة الذاكرة مرة أخرى.

• يتم إلغاء تخصيص كتلة واحدة، ويتم طباعة الحالة المحدثة للكتل مرة أخرى.

3-8-خوارزمية الملاءمة الأسوأ Worst-fit:

تحجز هذه الطريقة الكتلة التي تمتلك أكبر مساحة كافية من بين الكتل المتوفرة في الذاكرة الرئيسية. وإن أتت عملية تتطلب مساحة كبيرة في مرحلة متأخرة فلن يكون هناك متّسع لهذه العملية في الذاكرة .

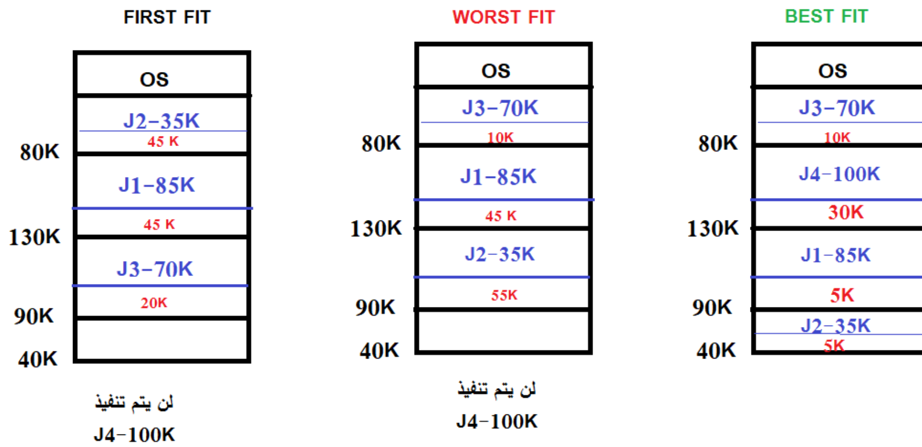
خطوات الخوارزمية

1. تُغذّى الخوارزمية بكتل الذاكرة المتاحة مع حجم كل كتلة إضافة إلى العمليات مع حجم كل عملية.
2. تهيّئ الخوارزمية بعدها جميع كتل الذاكرة لتكون شاغرة.
3. تبدأ الخوارزمية باختيار كلّ عملية والبحث عن الكتلة ذات الحجم الأكبر والتي يمكن تعيينها إلى العملية الحالية، فإن وجدتها عيّنتها إلى العملية الحالية.
4. إن لم تعثر الخوارزمية على الكتلة المطلوبة تجاوزت تلك العملية وانتقلت إلى العملية التي تليها.

مثال:

لنفرض لدينا رتل عمل يحوي المهام التالية: J1-85K, J2-35K, J3-70K, J4-100K

قائمة الثقوب : 80K,130K,90K,40K



9- التجزئة

- التجزئة الخارجية External Fragmentation –توجد مساحة ذاكرة إجمالية لتلبية طلب، ولكنها ليست متجاورة
- التجزئة الداخلية Internal Fragmentation – قد تكون الذاكرة المخصصة أكبر قليلاً من الذاكرة المطلوبة؛ هذا الاختلاف في الحجم هو ذاكرة داخلية للقسم، ولكنها غير مستخدمة

يكشف تحليل الملاءمة الأول أنه في حالة تخصيص N كتلة، يتم فقد 0.5 N كتلة بسبب التجزئة

قد يكون 1/3 غير قابل للاستخدام -> قاعدة 50 بالمائة

10- المبادلة Swapping :

يمكن مبادلة عملية مؤقتًا من الذاكرة إلى مخزن احتياطي، ثم إعادتها إلى الذاكرة لمواصلة التنفيذ، يمكن أن تتجاوز مساحة الذاكرة الفعلية الإجمالية للعمليات الذاكرة الفعلية

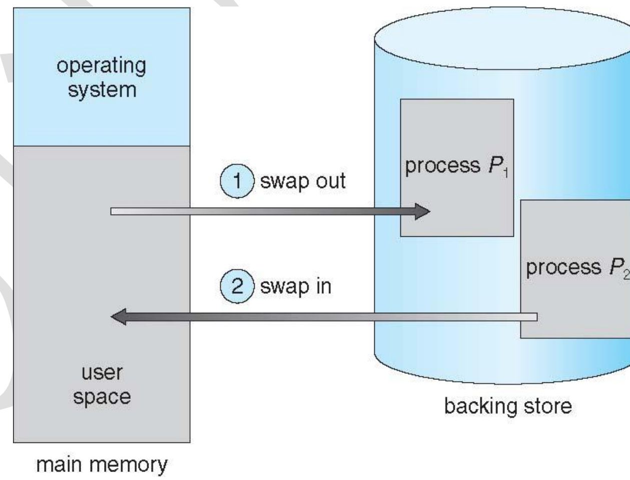
- مخزن النسخ الاحتياطي Backing store - قرص سريع كبير بما يكفي لاستيعاب نسخ من جميع صور الذاكرة لجميع المستخدمين؛ يجب توفير وصول مباشر إلى صور الذاكرة هذه
- اخراج ، ادخال Roll out, roll in- متغير المبادلة المستخدم لخوارزميات الجدولة القائمة على الأولوية؛ يتم مبادلة العملية ذات الأولوية المنخفضة حتى يمكن تحميل العملية ذات الأولوية الأعلى وتنفيذها

الجزء الأكبر من وقت المبادلة هو زمن النقل؛ زمن النقل الإجمالي يتناسب بشكل مباشر مع مقدار الذاكرة التي تم مبادلتها

يحافظ النظام على قائمة انتظار جاهزة للعمليات الجاهزة للتشغيل والتي تحتوي على صور ذاكرة على القرص

<u>هل تحتاج عملية الاستبدال إلى الاستبدال مرة أخرى بنفس العناوين المادية؟</u>

- يعتمد على طريقة ربط العنوان، بالإضافة إلى مراعاة عمليات الإدخال/الإخراج المعلقة من/إلى مساحة ذاكرة العملية
- توجد إصدارات معدلة من الاستبدال على العديد من الأنظمة (أي UNIX وLinux وWindows)
- يتم تعطيل الاستبدال عادةً
- يتم البدء في حالة تخصيص كمية أكبر من الحد الأقصى للذاكرة
- يتم تعطيله مرة أخرى بمجرد انخفاض الطلب على الذاكرة إلى ما دون الحد الأقصى



11- الترحيل Paging :

يمكن أن تكون مساحة العنوان المادية لعملية ما غير متجاورة؛ يتم تخصيص ذاكرة مادية للعملية كلما كانت الأخيرة متاحة

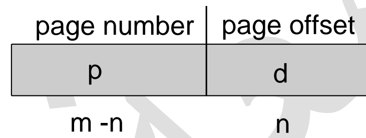
- يتجنب التجزئة الخارجية
- يتجنب مشكلة تباين أحجام أجزاء الذاكرة
- يقسم الذاكرة المادية إلى كتل ذات حجم ثابت تسمى الإطارات

- الحجم هو قوة 2، بين 512 بايت و16 ميجا بايت
- يقسم الذاكرة المنطقية إلى كتل بنفس الحجم تسمى الصفحات
- لتشغيل برنامج بحجم N صفحة، تحتاج إلى العثور على N إطار حر وتحميل البرنامج
- إعداد جدول صفحات لترجمة العناوين المنطقية إلى عناوين مادية
- يتم تقسيم مخزن النسخ الاحتياطي أيضًا إلى صفحات
- لا يزال هناك تجزئة داخلية

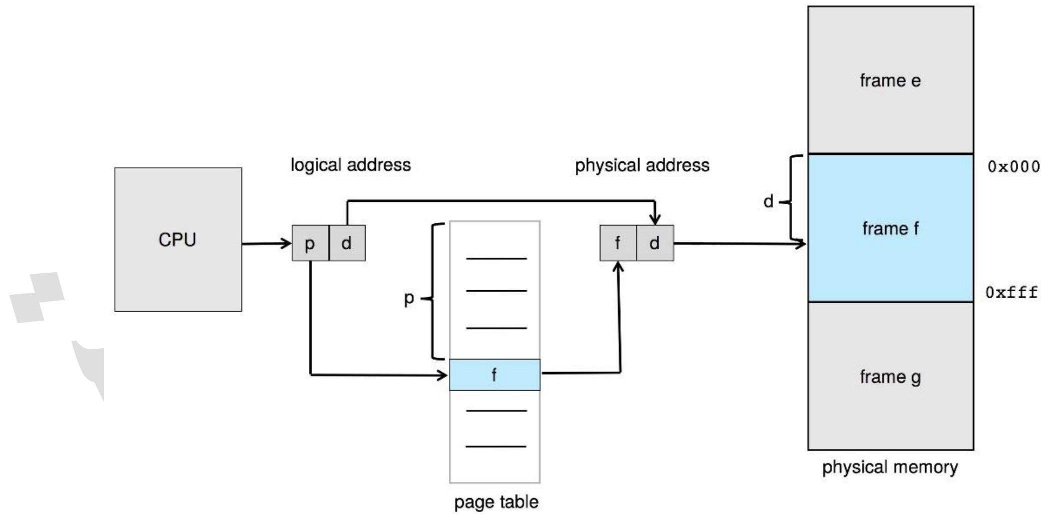
مخطط ترجمة العنوان Address Translation Scheme:

يتم تقسيم العنوان الذي يتم إنشاؤه بواسطة وحدة المعالجة المركزية إلى:

- رقم الصفحة (p) - يستخدم كمؤشر في جدول الصفحات الذي يحتوي على العنوان الأساسي لكل صفحة في الذاكرة الفعلية
- إزاحة الصفحة (d) - يتم دمجها مع العنوان الأساسي لتحديد عنوان الذاكرة الفعلية الذي يتم إرساله إلى وحدة الذاكرة

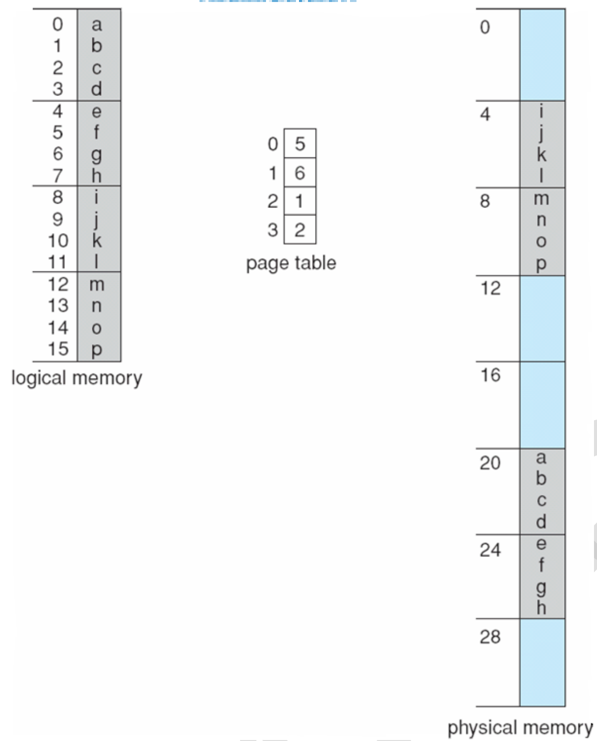


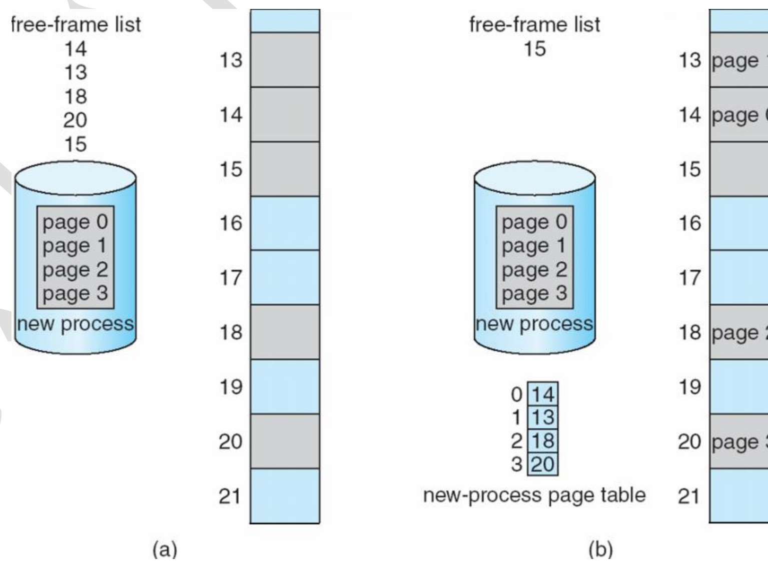
1-11-أجهزة الترحيل Paging Hardware:



نموذج الترحيل للذاكرة المنطقية والفيزيائية:

العنوان المنطقي: n = 2 و m = 4. باستخدام حجم صفحة يبلغ 4 بايت وذاكرة فعلية تبلغ 32 بايت (8 صفحات)





تنفيذ جدول الصفحات Implementation of Page Table

يتم الاحتفاظ بجدول الصفحات في الذاكرة الرئيسية

- ✓ يشير سجل قاعدة جدول الصفحات (PTBR) Page-table base register إلى جدول الصفحات
- ✓ يشير سجل طول جدول الصفحات (PTLR) Page-table length register إلى حجم جدول الصفحات

https://manara.edu.sy/

في هذا المخطط، يتطلب كل وصول للبيانات/التعليمات وصولين للذاكرة واحد لجدول الصفحات وواحد للبيانات/التعليمات

يمكن حل مشكلة الوصول للذاكرة المزدوجة باستخدام ذاكرة تخزين مؤقتة خاصة بالأجهزة للبحث السريع تسمى مخازن البحث عن الترجمة translation look-aside buffers (TLBs) (وتسمى أيضًا الذاكرة الترابطية).

فيما يلي مثال توضيحي بلغة C++ لنظام ترحيل بسيط، والذي ينشئ جدول صفحات أساسيًا لربط العناوين الافتراضية بالإطارات المادية. لا يتضمن هذا المثال كل التعقيدات المتعلقة بإدارة الذاكرة في نظام التشغيل الحقيقي، ولكنه يوفر فهمًا أساسيًا للمفهوم.

```
#include <iostream>
#include <vector>
#include <unordered_map>
4
const int PAGE_SIZE = 4; // Number of bytes per page
const int MEMORY_SIZE = 16; // Total physical memory size in bytes
const int NUM_PAGES = MEMORY_SIZE / PAGE_SIZE; // Number of pages in memory
8
9  // Structure to represent a page
10  struct Page {
11      int pageNumber; // Virtual page number
12      int frameNumber; // Physical frame number
13  };
14
15  // Class to simulate a paging system
16  class PagingSystem {
17  private:
18      std::unordered_map<int, Page> pageTable; // Maps virtual page numbers to physical
frames
19      std::vector<int> memory; // Physical memory represented as an array of frames
20
21  public:
22      PagingSystem() {
23          // Initialize physical memory
24          memory.resize(MEMORY_SIZE / PAGE_SIZE, -1); // -1 represents an empty frame
25      }
26
27      void loadPage(int virtualPageNumber) {
28          // Check if the page is already in memory
29          if (pageTable.find(virtualPageNumber) != pageTable.end()) {
30              std::cout << "Page " << virtualPageNumber << " is already in memory." <<
std::endl;
31              return;
32          }
33
34          // Find an empty frame to load the page
35          for (int frame = 0; frame < NUM_PAGES; frame++) {
36              if (memory[frame] == -1) { // If frame is empty
37                  memory[frame] = virtualPageNumber; // Load page into frame
38                  pageTable[virtualPageNumber] = {virtualPageNumber, frame}; // Update
page table
39                  std::cout << "Loaded page " << virtualPageNumber << " into frame " <<
frame << std::endl;
40                  return;
41              }
42          }
43
44          std::cout << "No free frames available to load page " << virtualPageNumber <<
std::endl;
45      }
46
47      void accessPage(int virtualPageNumber) {
48          if (pageTable.find(virtualPageNumber) != pageTable.end()) {
49              int frameNumber = pageTable[virtualPageNumber].frameNumber;
```

```
50              std::cout << "Accessing page " << virtualPageNumber << " in frame " <<
frameNumber << std::endl;
51          } else {
52              std::cout << "Page " << virtualPageNumber << " is not in memory. Loading
it now..." << std::endl;
53              loadPage(virtualPageNumber);
54          }
55      }
56
57      void printMemory() {
58          std::cout << "Physical Memory Status:" << std::endl;
59          for (int i = 0; i < NUM_PAGES; i++) {
60              std::cout << "Frame " << i << ": ";
61              if (memory[i] == -1) {
62                  std::cout << "Empty" << std::endl;
63              } else {
64                  std::cout << "Page " << memory[i] << std::endl;
65              }
66          }
67      }
68  };
70  int main() {
71      PagingSystem pagingSystem;
72
73      pagingSystem.loadPage(0); // Load page 0
74      pagingSystem.loadPage(1); // Load page 1
75      pagingSystem.loadPage(2); // Load page 2
76      pagingSystem.accessPage(1); // Access page 1
77      pagingSystem.accessPage(3); // Access page 3 (not loaded, should load it)
78      pagingSystem.loadPage(4); // Load page 4
80      pagingSystem.printMemory(); // Print current memory status
82      return 0;
```

يحاكي كود C++ هذا نظام ترقيم بسيط باستخدام خوارزمية استبدال الصفحة حسب أول مناسبة (أو، بشكل أكثر دقة، عدم استبدال الصفحة على الإطلاق - فهو ببساطة يقوم بتحميل الصفحات في أول إطار متاح). دعنا نقسم الكود قسمًا تلو الآخر:

1. الثوابت وهياكل البيانات:

- PAGE_SIZE: يحدد حجم الصفحة بالبايتات (4 بايت في هذه الحالة).
- MEMORY_SIZE: يحدد الحجم الإجمالي للذاكرة الفعلية بالبايتات (16 بايت).
- NUM_PAGES: يحسب عدد الصفحات التي يمكن أن تتسع لها الذاكرة الفعلية (MEMORY_SIZE / PAGE_SIZE).
- Page struct: يمثل صفحة في الذاكرة، ويخزن رقم الصفحة الافتراضية (pageNumber) ورقم الإطار الفعلي (frameNumber).

2. PagingSystem Class:

pageTable: خريطة غير مرتبة تعمل كجدول صفحات. يقوم بربط أرقام الصفحات الافتراضية (المفاتيح) بهياكل الصفحات (القيم)، مما يشير إلى مكان وجود كل صفحة افتراضية في الذاكرة الفعلية.

الذاكرة: متجه يمثل الذاكرة الفعلية. يتوافق كل عنصر من المتجه مع إطار، وتمثل القيمة الموجودة داخله رقم الصفحة الافتراضية المحملة في هذا الإطار. تشير القيمة -1 إلى إطار فارغ.

3. طرق PagingSystem:

- PagingSystem(): يقوم المنشئ بتهيئة متجه الذاكرة بـ -1 في كل إطار، مما يدل على أن جميع الإطارات فارغة في البداية.
- loadPage(int virtualPageNumber): تحاول هذه الوظيفة تحميل صفحة افتراضية في الذاكرة الفعلية.

- تتحقق أولاً مما إذا كانت الصفحة موجودة بالفعل في الذاكرة باستخدام pageTable.
- إذا لم يكن الأمر كذلك، فإنها تتكرر عبر متجه الذاكرة للعثور على أول إطار فارغ (-1).
- بمجرد العثور على إطار فارغ، تقوم بتحميل رقم الصفحة الافتراضية في هذا الإطار وتحديث pageTable وفقًا لذلك.
- إذا لم يتم العثور على أي إطارات فارغة، فهذا يشير إلى أن الذاكرة ممتلئة.
- accessPage(int virtualPageNumber): يحاكي هذا الوصول إلى صفحة افتراضية.
- يتحقق مما إذا كانت الصفحة موجودة بالفعل في جدول الصفحات.
- إذا كانت الصفحة موجودة، فإنها تطبع رسالة تشير إلى رقم الإطار.
- إذا لم تكن الصفحة موجودة، فإنها تستدعي loadPage() لتحميل الصفحة في الذاكرة.
- printMemory(): تطبع هذه الوظيفة الحالة الحالية للذاكرة الفعلية، مع إظهار الصفحات المحملة في أي إطارات.

4. وظيفة main():

توضح وظيفة main() استخدام فئة PagingSystem. فهي تقوم بتحميل عدة صفحات، والوصول إلى بعض الصفحات (بما في ذلك صفحة لم يتم تحميلها بعد)، ثم تطبع الحالة النهائية للذاكرة.

القيود والتحسينات:

- First-Fit Only: يستخدم هذا الكود استراتيجية First-Fit فقط لتخصيص الإطارات. سيستخدم النظام الأكثر واقعية خوارزميات استبدال الصفحات الأكثر تطورًا (مثل LRU أو FIFO أو الأمثل) للتعامل مع المواقف التي تكون فيها جميع الإطارات مشغولة.

- لا استبدال للصفحة: عندما تكون الذاكرة ممتلئة، يقوم البرنامج ببساطة بالإبلاغ عن خطأ. سيحتاج نظام الترقيم الحقيقي إلى سياسة استبدال الصفحات لطرد الصفحات الموجودة لإفساح المجال لصفحات جديدة.
- نموذج الذاكرة المبسّط: هذا نموذج مبسط للغاية. إدارة الذاكرة الحقيقية أكثر تعقيدًا إلى حد كبير، حيث تتضمن الذاكرة الافتراضية والتجزئة والمزيد.
- يعمل هذا الكود كنقطة بداية جيدة لفهم المفاهيم الأساسية للترقيم وجداول الصفحات. ومع ذلك، تذكر أن هذه محاكاة مبسطة، وأن أنظمة الترقيم في العالم الحقيقي أكثر تعقيدًا.