الباب السادس
إدارة الذاكـرة الرئيسيـة (Memory Manager)

تنظيم وإدارة الذاكرة الرئيسية للحاسب لها تأثير كبير و أساسي على فعالية نظام التشغيل ، **تنظيم الحفظ أو التخزين يقصد به الكيفية التي ينظر من خلالها لوحدة التخزين** ، هل تستخدم من قبل **مستخدم واحد** أم **مجموعة مستخدمين** في نفس الوقت؟، وفى حالة وجود عدة برامج في الذاكرة الرئيسية في نفس الوقت هل كلها **تمنح مساحات متساوية أو تقسم الذاكرة الرئيسية لأجزاء مختلفة المساحة** كل منها يحتوى على برنامج . وهل تكون هذه **التقسيمات ثابتة غير قابلة للتغيير أو حرة يمكن تعديلها** حسب حاجة النظام, كذلك هل يجب أن تصمم **برامج المستخدم ليتم تنفيذها في قسم معين من الذاكرة** أو يمكنها العمل في أي قسم من الذاكرة و من ثم هل يجب **وضع البرنامج أو العملية في أجزاء متجاورة من الذاكرة أو متفرقة** حسب الأماكن الشاغرة في الذاكرة ، كل هذا مسؤول عنه **مدير الذاكرة (Memory manager).**

أهداف مدير الذاكرة : هنالك العديد من الأهداف من وراء تصميم مدير الذاكرة، مثل:

- حجز المواقع وتحريرها.
- التعامل مع هرمية الذاكرة.
- **العناوين المنطقية** :استخدام العناوين المنطقية للتعامل مع الذاكرة.
- **الحماية** :حماية البرامج عن بعضها البعض.
- **المشاركة** :توفير المشاركة بين البرامج دون التأثير على الحماية.
- **توسيع الذاكرة** :تمديد الذاكرة لتستوعب برامج أكبر من حجمها وذلك باستخدام جزء من القرص الصلب.

سنبدأ بتوضيح طريقة إدارة الذاكرة في النظم القديمة (الذي لم تعد مستخدمة حاليا ولكنها ستساعدنا في فهم نظم التشغيل الحديث للذاكرة).

حيث سنتحدث عن:

- **الذاكرة أحادية المهام (النظم القديمة).**
- **الذاكرة متعددة المهام (النظم الحديثة).**
- **التجزئة الثابتة (fixed partitions).**
- **التجزئة الديناميكية.**
- **تجميع الفراغات.**
- **الذاكرة بالصفحات.**
- **الذاكرة بالتقطيع.**

الشكل 5.1

نظام التشغيل أحادي المهام

قديما كانت الذاكرة صغيرة وتعمل **بالنظام الأحادي (single program)** حيث يسمح للمستخدم بتشغيل **برنامج واحد فقط بالإضافة إلى نظام التشغيل** وبالتالي فإن **الذاكرة الرئيسية** تكون مقسومة إلى **جزئيين**، جزء مخصص **لنظام التشغيل** وجزء مخصص **للمستخدم** لينفذ فيه برنامج واحد فقط كل مرة، هذا النوع من نظم التشغيل يسمى نظم التشغيل أحادية المهام (single task) الشكل 5.1.

عيوب النظام الأحادي

هنالك **برنامج واحد فقط بالذاكرة** حتى لو كان هذا البرنامج صغير جدا ولا يستغل إلا القليل من مساحة المستخدم، وهذا يتسبب في الآتي:

- إهدار مساحة الذاكرة.
- لن نستطيع تشغيل برامج حجمها أكبر من مساحة المستخدم.
- إهدار زمن المعالج.

كيف يدير نظام التشغيل الذاكرة (في هذه الحالة)

- يقوم **مدير الذاكرة بمعرفة حجم البرنامج** الذي طلبنا تنفيذه ، ثم يقارن هذا الحجم مع حجم مساحة المستخدم، **إذا كان حجم البرنامج أصغر أو يساوي مساحة المستخدم** سيتم تحميله في الذاكرة، وإلا فسيمنع تحميله وقد تظهر رسالة توضح ذلك **لا توجد ذاكرة كافية *(not enough memory)*.**

- أيضا يقوم **مدير الذاكرة بحماية منطقة نظام التشغيل** من برامج المستخدم بحيث يتم **وضع العنوان الفاصل بين نظام التشغيل ومساحة المستخدم** في مسجل يسمى **مسجل الحماية *(Protection register)***، عندما ينفذ برنامج المستخدم أي عملية وصول للذاكرة **سيقارن نظام التشغيل العنوان المستخدم في الأمر مع العنوان المخزن في مسجل الحماية** إذا كان **أعلى منه**، يمنع البرنامج من تنفيذ الأمر لأنه تعدي للحدود الإقليمية وتجاوز لمنطقة المستخدم للوصول إلى منطقة نظام التشغيل.

مثال على ذلك نظام التشغيل DOS

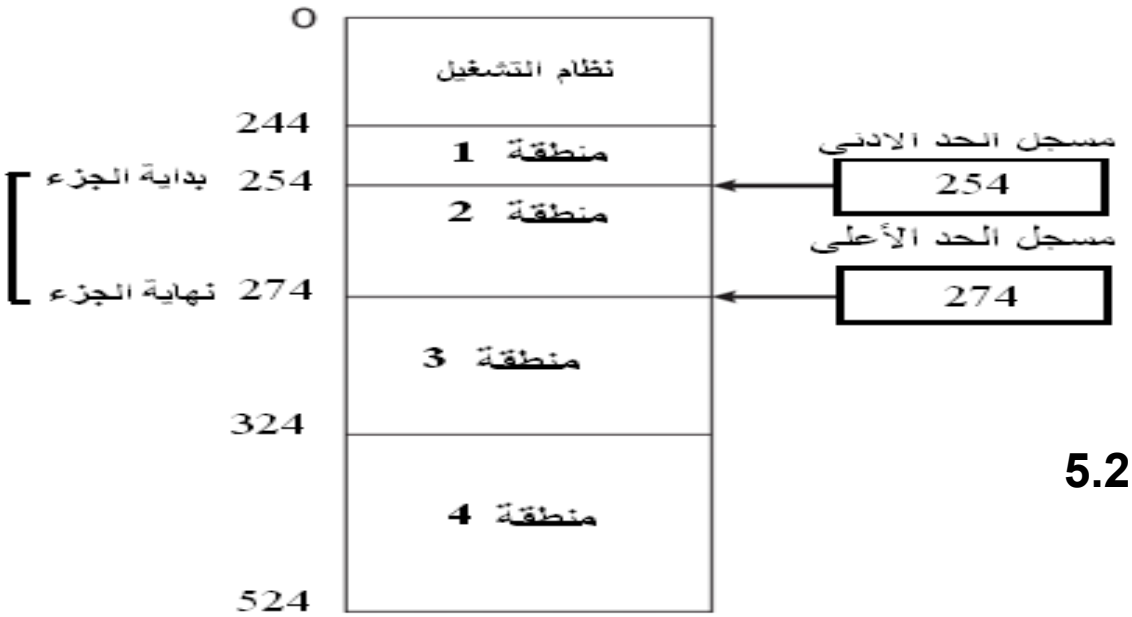
نظام التشغيل متعدد المهام

إذا سمح نظام التشغيل للمستخدم **بتشغيل أكثر من برنامج في وقت واحد**، فأنت تتعامل مع نظام تشغيل متعدد المهام (***multitasking***) أو متعدد البرامج (***multiprogramming***) حيث يمكن تحميل أكثر من برنامج بالذاكرة. (نظام التشغيل **ويندوز**).

وعلى ذلك فانه **يتم تقسيم الذاكرة الى عدة مناطق.**

التجزئة الثابتة

يقسم نظام التشغيل الذاكرة الى **مناطق ثابتة** وقد تكون متساوية او مختلفة في الأحجام وذلك لإتاحة الفرصة لتحميل برامج بأحجام مختلفة في هذه المناطق. **تستخدم إدارة الذاكرة مسجلي حدود لكل منطقة من مناطق الذاكرة** حيث يخزن في المسجل الأول **الحد الأعلى** للمنطقة بينما يخزن في المسجل الثاني **الحد الأسفل** للمنطقة.



الشكل 5.2

كيف يعمل مدير الذاكرة

يقوم مدير الذاكرة باستخدام جدول يسمى **جدول الحجز**، يحتوي هذا الجدول على **رقم كل منطقة وحجمها وأين توجد** بالذاكرة وهل هي مستخدمة أم فارغة.

رقم المنطقة	حجم المنطقة	بدايتها	استخدامها
1	10	244	فارغ
2	20	**254**	فارغ
3	50	**274**	مشغول
4	200	**324**	فارغ

الشكل 5.3

- **درجة تعدد المهام** هي **4** (يمكن تحميل 4 برامج في وقت واحد).
- **المنطقة 3** مشغولة (بها برنامج الآن، ولا يمكن تحميل برامج بها).
- **حجم أكبر برنامج** يمكن تحميله هنا هو **200 كيلوبايت** (منطقة رقم 4)

خوارزميات التسكين (الحجز)

إذا كان هنالك عدة مناطق فارغة ونريد تحميل برامج بها، فكيف سيتم تحميل هذه البرامج ؟ هنالك خوارزميات مختلفة لتحميل البرامج بالذاكرة مثل :

- **الأنسب (best-fit)** : يتم وضع البرنامج في **أقل فراغ يمكن أن يستوعبه**، حيث سيتم **البحث عن كل الفراغات المتاحة بالذاكرة واختيار أقل فراغ** يمكن أن يستوعب البرنامج (العملية) التي نريد تحميلها بالذاكرة.
- **الانسب-الأول (first-fit)** : يتم وضع البرنامج **في أول فراغ يمكن أن يستوعبه**، هذه الخوارزمية **لا تحتاج بحث عن كل الفراغات** الموجودة بالذاكرة، وإنما ستضع البرنامج المراد تحميله في أول فراغ تجده بالذاكرة يكون حجمه أكبر أو يساوي حجم البرنامج.
- **الانسب-الأسوأ (worst-fit)**: يتم وضع البرنامج في **أكبر فراغ يمكن أن يستوعب البرنامج**، حيث سيتم **البحث عن كل الفراغات الموجودة بالذاكرة واختيار الاكبر (الاسوأ)** أكبرها حجما لوضع البرنامج المراد تحميله فيه بحيث يكون الفراغ أكبر أو يساوي حجم البرنامج.

* **ملاحظة** : **يمكن استخدام الفراغات المتبقية لتسكين البرامج اذا كانت مناسبة في الحجم.**

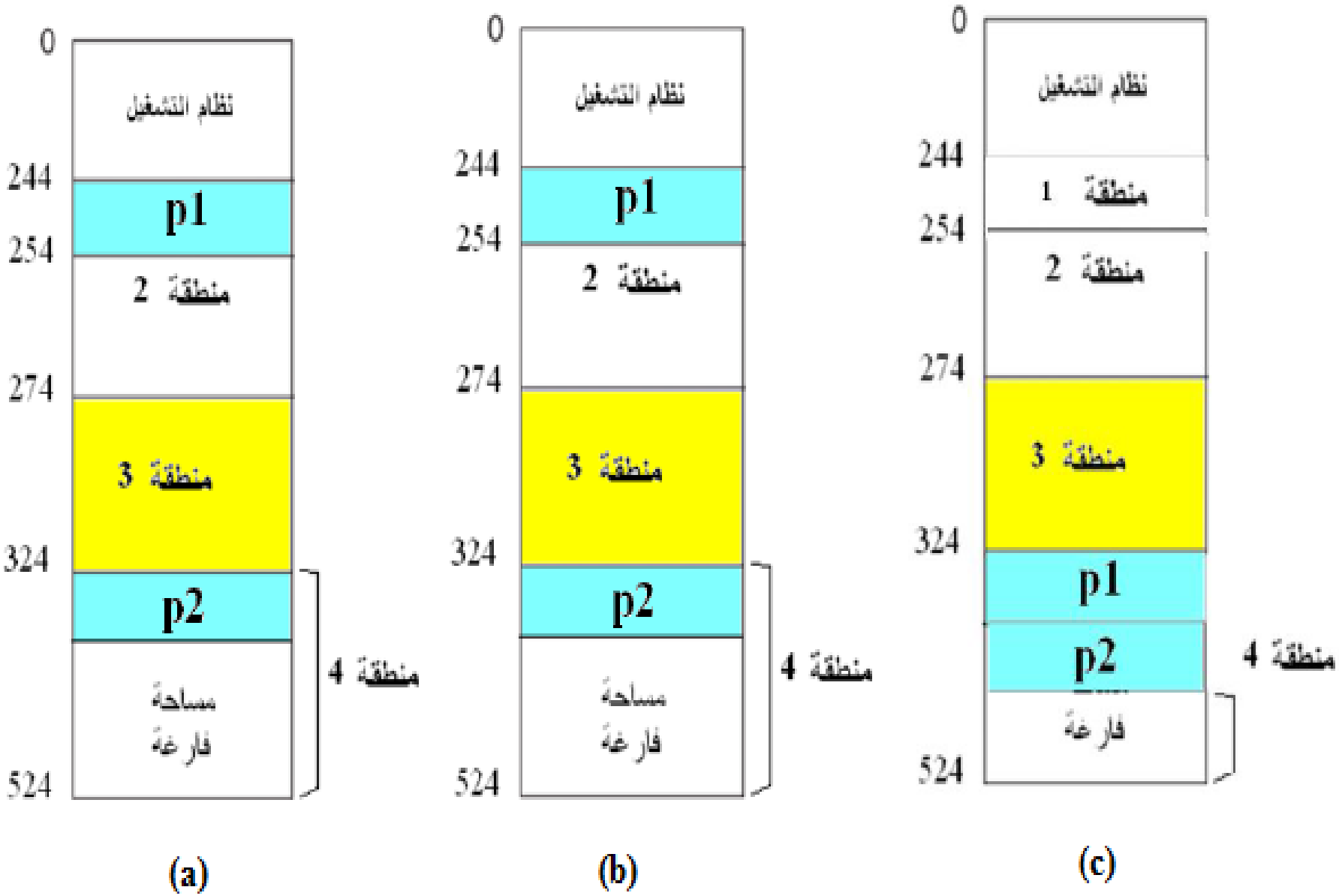
مثال

إذا كان لدينا ثلاث برامج بالأحجام P1=10 ، P2=50 ، P3=200 ، معطى بجدول الحجز الاتي، فكيف سيتم تحميل هذه البرامج باستخدام خوارزميات التسكين أعلاه ؟

الحل

- طريقة **الأنسب (best-fit)** سنضع P1 في **المنطقة 1**، و P2 في **المنطقة 4**، و P3 **ستنتظر** (لا يمكن تحميله لأنه لا يوجد فراغ كافي)، أنظر الشكل التالي (a).
- طريقة **الانسب-الأول (first-Fit)** سنضع P1 في **المنطقة 1**، و P2 في **المنطقة 4** و P3 **ستنتظر** (لا يمكن تحميلها لأنه لا يوجد فراغ كافي)، أنظر الشكل التالي (b).
- طريقة **الانسب-الأسوأ (worst-fit)** سنضع P1 في **المنطقة 4**، و P2 في المتبقي من (200-10=190) **المنطقة 4** و P3 لا يمكن تحميله بالذاكرة لأنه لا يوجد فراغ يكفي. أنظر الشكل التالي (c).

رقم المنطقة	حجم المنطقة	بدايتها	استخدامها
1	10	244	فارغ
2	20		فارغ
3	50		مشغول
4	200		فارغ


0
نظام التشغيل
244
p1
254
منطقة 2
274
منطقة 3
324
p2
مساحة
فارغة
524
4 منطقة
(a)
0
نظام التشغيل
244
p1
254
منطقة 2
274
منطقة 3
324
p2
مساحة
فارغة
524
4 منطقة
(b)
0
نظام التشغيل
244
منطقة 1
254
منطقة 2
274
منطقة 3
324
p1
p2
فارغة
524
4 منطقة
(c)

الشكل 5.4

مثال

إذا كانت لدينا ذاكرة تتكون من الأقسام التالية بالترتيب ومن اليسار لليمين:

100k, 500k, 200k, 300k, 600k

وضح كيف تضع كل من الخوارزميات، الأنسب (**BF**)، الأول (**FF**)، والأسوأ (**WF**)، العمليات التالية بالذاكرة:

P1=212k, **P2**=417k, **P3**=112k, **P4**=426k

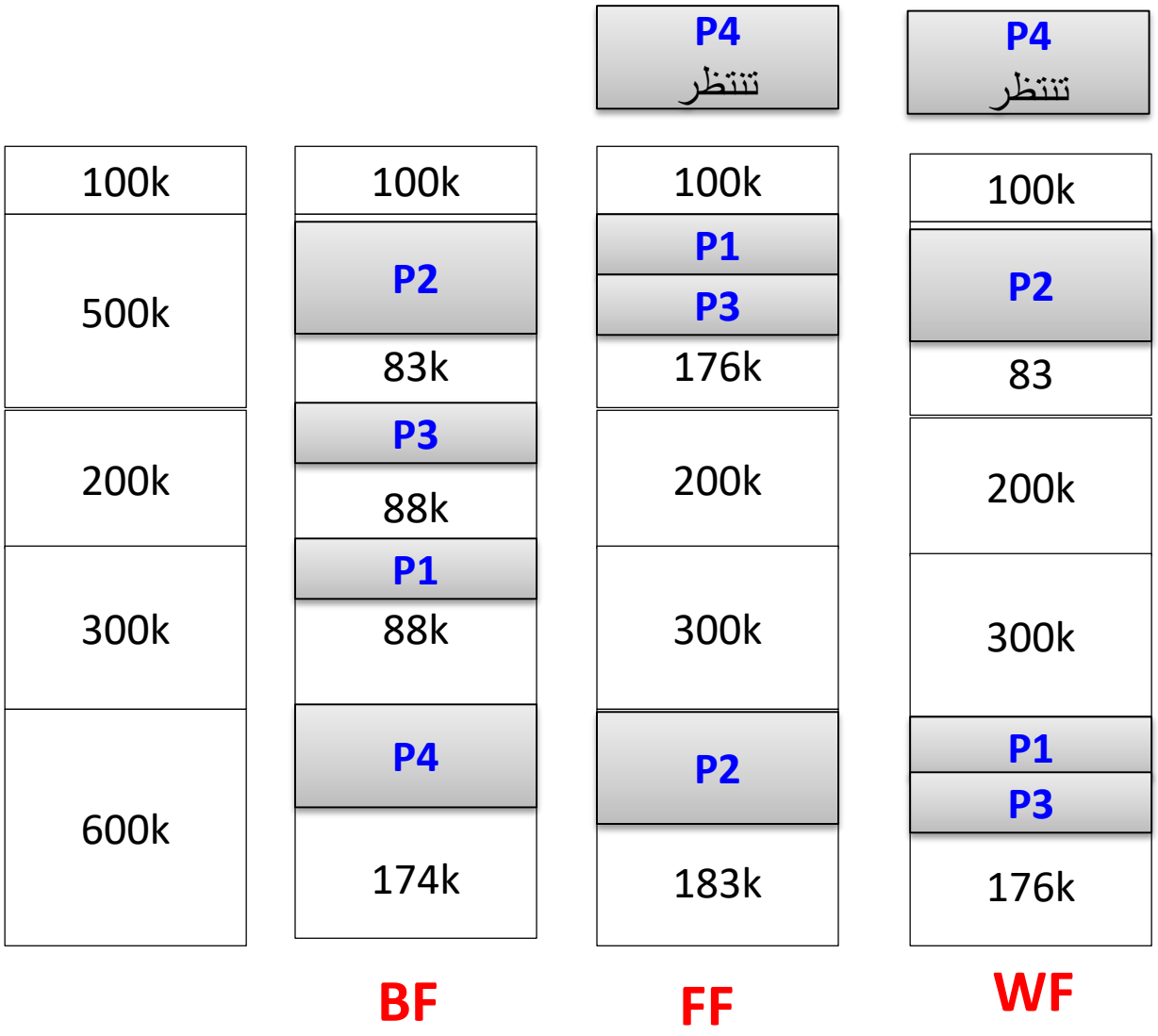
أي خوارزمية هي الأفضل في استخدام الذاكرة ؟

الحل

- **الأنسب (BF) :** نضع **212** في **300** ، **417** في **500** ، **112** في **200** ، **426** في **600**
- **الانسب-الاول (FF)** : ضع **212** في القسم **500** ، **417** في القسم **600** ، **112** في القسم **288** (قسم نتج من 500 - 212 = **288**) **426** تنتظر.
- **الأسوأ (WF) : 212** في **600** ، **417** في **500** ، **112** في **388** ، **426** تنتظر.

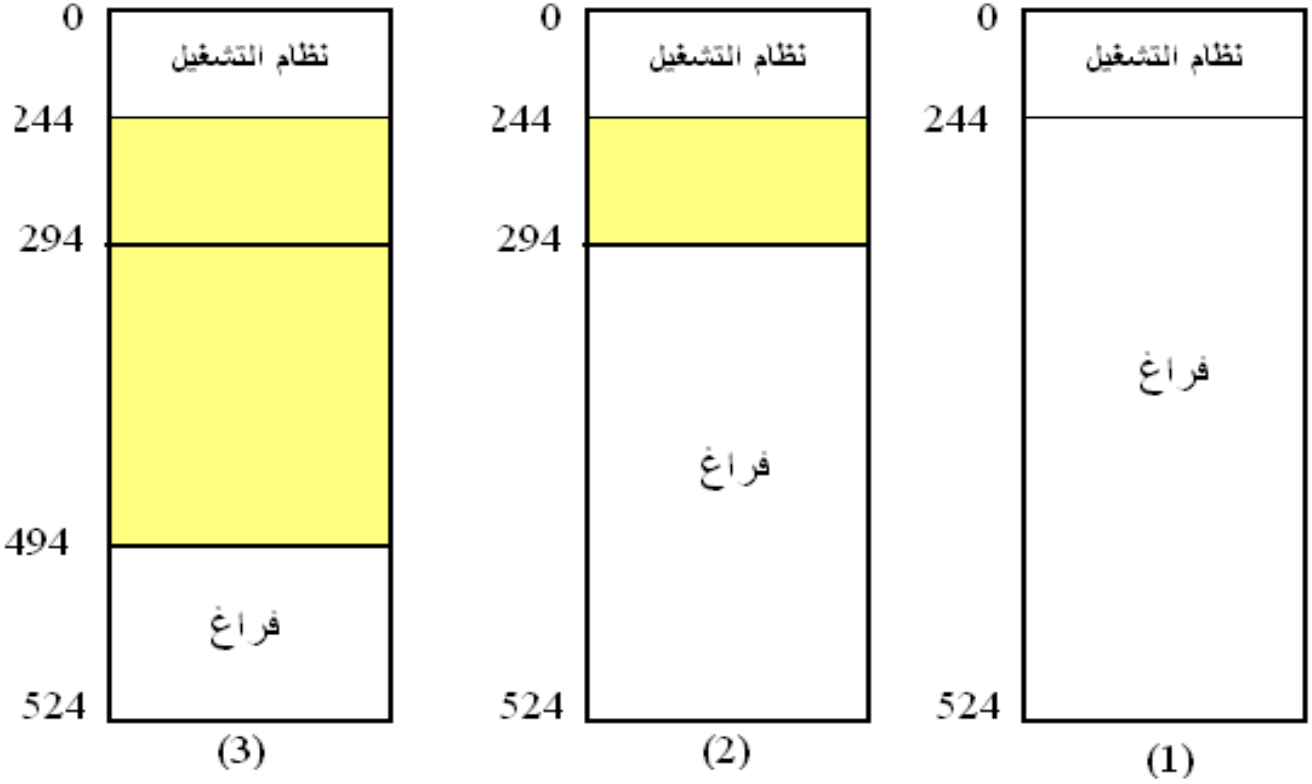
في هذا المثال نجد أن **الأنسب (BF)** هي أفضل خوارزمية.

P1 = 212k, **P2** = 417k, **P3** = 112k, **P4** = 426k



التجزئة الديناميكية

تكون **مساحة المستخدم بالذاكرة في البداية غير مقسمة**، ثم **ينشأ جزء كلما تم تحميل برنامج بالذاكرة** (يكون بحجم البرنامج) وتكون باقي المساحة فارغة، ثم إذا تم تحميل برنامج آخر، سيبنى جزء ثاني بحجم البرنامج الثاني، وهكذا، كما في الشكل الاتي.



الشكل 5.5

الفراغات (fragmentations)

- **الفراغات الخارجية External Fragmentation** : مساحة فارغة غير متلاصقة تنتج بسبب تحميل العمليات وإخراجها من الذاكرة.
- **الفراغات الداخلية Internal Fragmentation** : قد يتم حجز منطقة لعملية بحيث تكون العملية اصغر من المنطقة مما يولد فراغ داخلي لا يمكن استخدامه .

يمكننا تجميع **الفراغات الخارجية** مع بعضها **بالضغط compaction**، لتكون فراغ خارجي واحد كبير يمكن الاستفادة منه.

ملحوظة:

- الفراغات **الخارجية يمكن ضغطها** لتكوين فراغ كبير .
- الفراغات **الداخلية لا يمكن ضغطها** وتجميعها كفراغ واحد.
- الفراغات **الداخلية** تتكون في **التجزئة الثابتة**.
- الفراغات **الخارجية** تنتج في **التجزئة الديناميكية**.
- **الضغط** يمكن تطبيقيه في **التجزئة الديناميكية**.

مثال

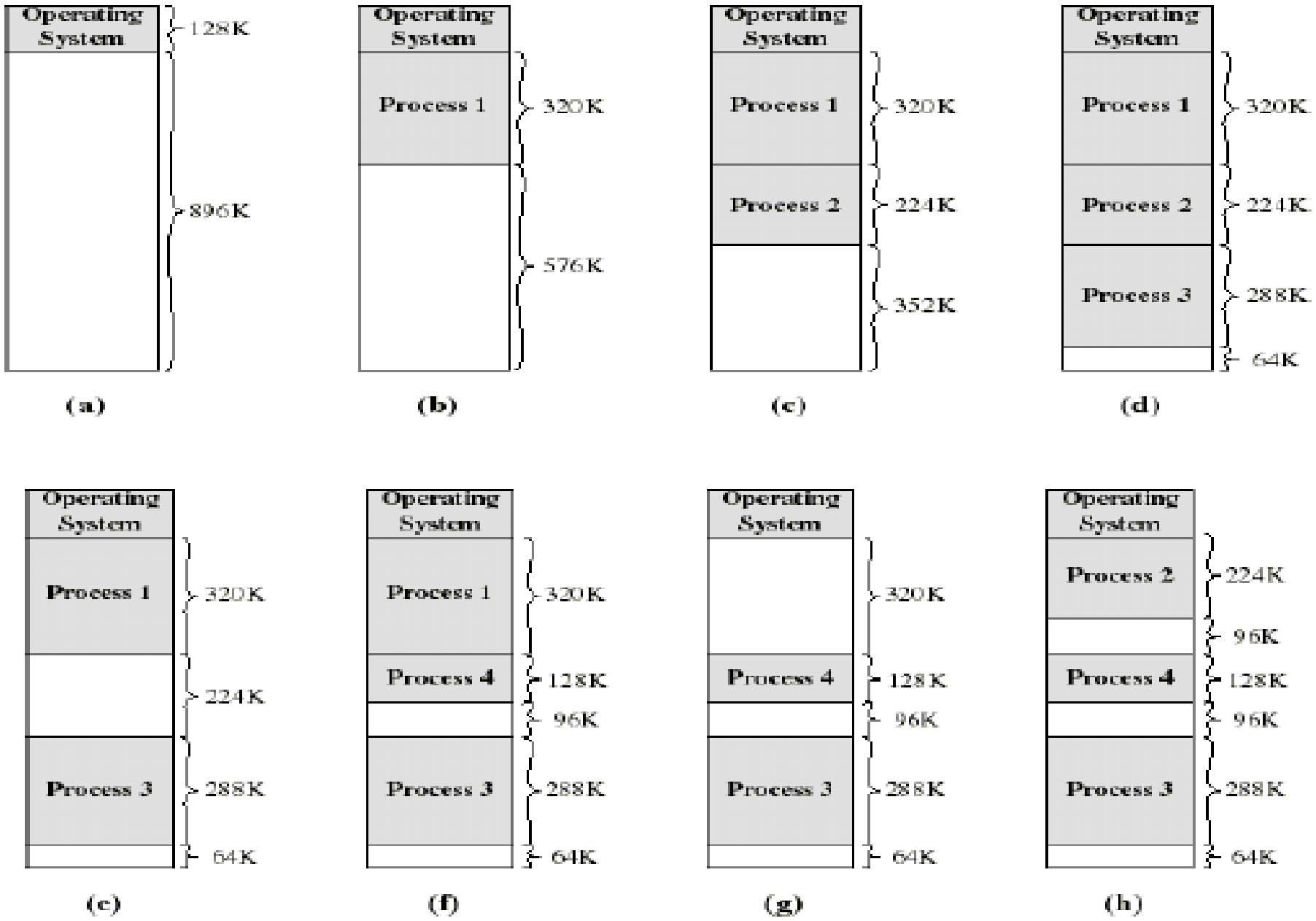
ذاكرة ديناميكية بمساحة حجمها **896 كيلوبايت**. بعد تحميل ثلاث عمليات فيها بالأحجام **320**، **224**, و**288**. سيكون **الجزء الحر** بها هو **64 كيلوبايت**، كما في الشكل التالي حيث يمكن حسابه كالاتي:

- **لحساب الجزء الحر**

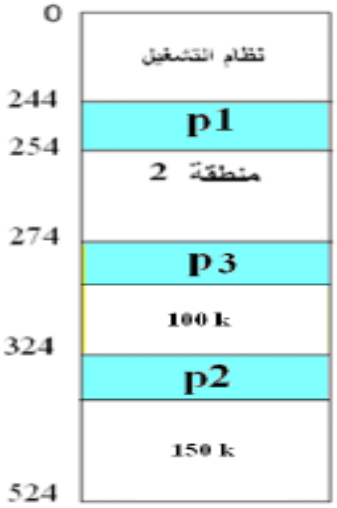
المساحة الحرة – مجموع حجم العمليات الثلاث المحملة = 896 - (288 + 224 + 320) = **64 كيلوبايت**

الآن إذا أردنا تحميل **عملية رابعة** بحجم **128كيلوبايت**، سيجيب نظام التشغيل بأنه لا توجد مساحة كافية بالذاكرة، ذلك لأن **64 < 128** لذلك علي العملية الرابعة الانتظار حتى تتوفر مساحة كافية لها.

إذا أخرج نظام التشغيل العملية **الثانية** (**e**)، يمكن للعملية **الرابعة** أن تحمل. لكن ستظهر **فراغات** (**fragmentation**) كما بالشكل (**f**). وإذا اخرج نظام التشغيل العملية **الأولى** (**g**)، ودخلت مكانها العملية **الثانية** مرة أخرى (**h**) سينشأ فراغ آخر كما في الشكل التالي (**h**).


Operating System
128K
896K
(a)
Operating System
Process 1
320K
576K
(b)
Operating System
Process 1
320K
Process 2
224K
352K
(c)
Operating System
Process 1
320K
Process 2
224K
Process 3
288K
64K
(d)
Operating System
Process 1
320K
224K
Process 3
288K
64K
(e)
Operating System
Process 1
320K
Process 4
128K
96K
Process 3
288K
64K
(f)
Operating System
320K
Process 4
128K
96K
Process 3
288K
64K
(g)
Operating System
Process 2
224K
96K
Process 4
128K
96K
Process 3
288K
64K
(h)

الشكل 5.6



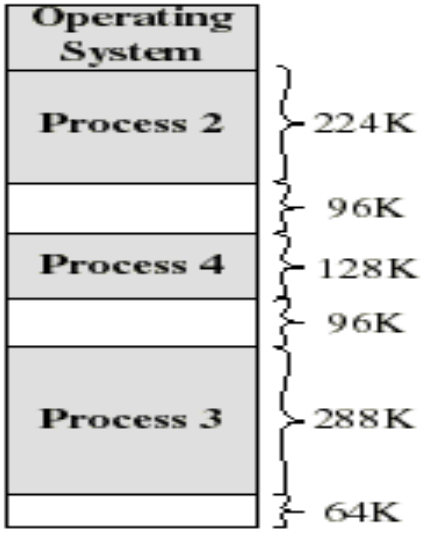
الشكل 5.7 : ذاكرة بالتجزئة الثابتة.

هل يمكن تحميل عملية حجمها **k250** في الشكل 5.7؟

- **لا** ، لأن الفراغات - في الشكل 5.7 **داخلية ولا يمكن ضغطها**.

تجميع الفراغات (defrag) أو الضغط (compaction)

إذا **لم تتوفر منطقة لتحميل برنامج معين** (لأنه أكبر من أي منطقة فارغة)، قد **تستخدم** بعض إدارات الذاكرة **عملية الضغط أو تجميع الفراغات** لتوفير مساحة أكبر وذلك بحساب مجموع الفراغات المتوفرة، فإذا كان حجمها أكبر أو يساوي حجم البرنامج تجمع هذه الفراغات لتكون فراغ واحد كبير يستطيع تحميل البرنامج. مثلا في **الشكل 5.8 (h)**، إذا استخدمنا الضغط سنجد أن الفراغ الذي سيجمع هو **96+96+64=256K**



(h)

الشكل 5.8

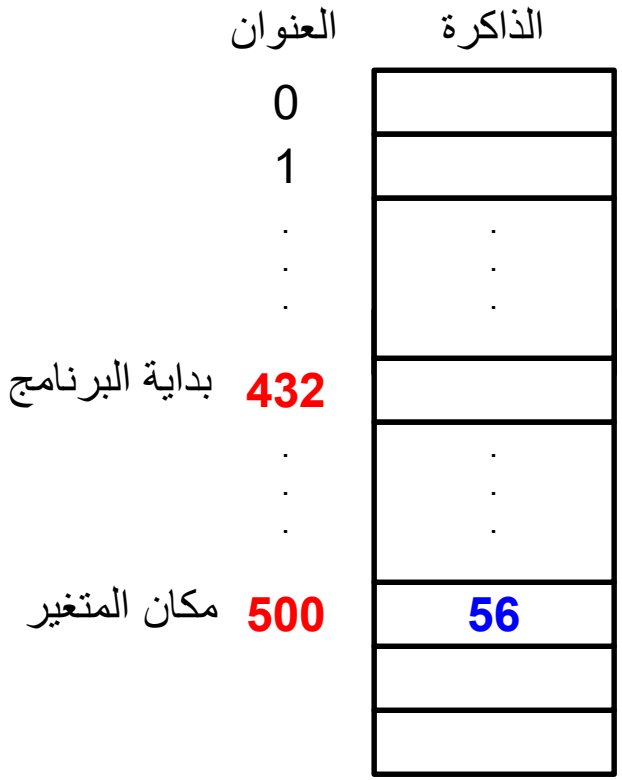
مشاكل تعدد المهام

الذاكرة متعددة المهام سواء كانت بالتجزئة الثابتة أو التجزئة الديناميكية أو غيرها، بها مشاكل ناتجة عن تعدد البرامج (أي وجود أكثر من برنامج بالذاكرة)، هذه المشاكل هي :

- **مشكلة تغيير الموقع Relocation**

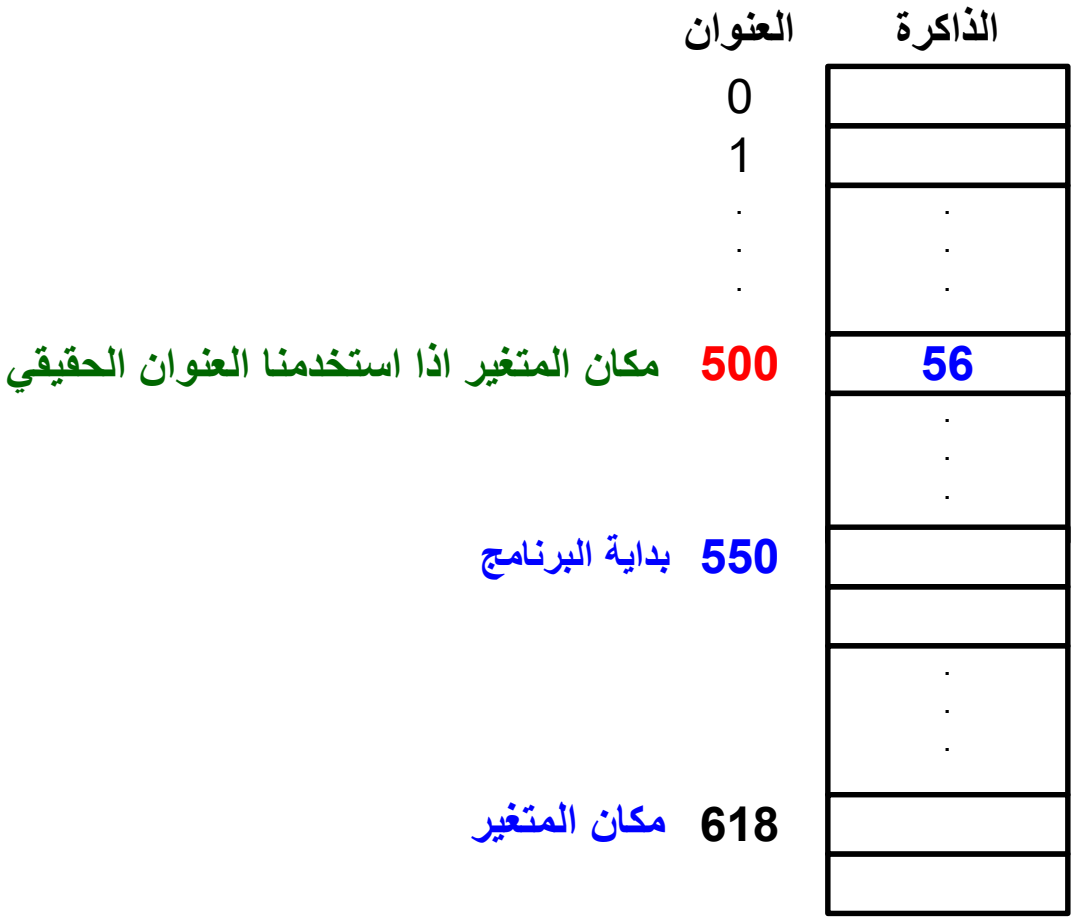
عند تحميل برنامج ما للتنفيذ، لا نعرف في أي منطقة بالذاكرة سيتم تحميله، أو إذا حدث **تبديل (swap)** أي تم إخراج البرنامج من الذاكرة لسبب ما، ثم تم اعادته مرة ثانية إلى الذاكرة، **قد يرجع البرنامج في منطقة أخرى** بالذاكرة غير التي كان فيها.

فمثلا إذا تم تحميل برنامج بموقع يبدأ بالعنوان **432**، وكان هنالك متغير داخل البرنامج بالموقع **500**، وأراد البرنامج تخزين متغير قيمته **56** كما بالشكل 5.9.



الشكل 5.9 : استخدام العنوان الحقيقي

الآن إذا تم تحميل البرنامج في موقع آخر يبدأ بالعنوان **550** مثلا، الشكل 5.10. فإن البرنامج سيستخدم عنوان الذاكرة **500** للوصول للمتغير، ولكن العنوان **500** الآن أصبح خارج منطقة البرنامج.



الشكل 5.10 : استخدام العنوان الحقيقي

أحد **الحلول** لهذه المشكلة هو **تعديل أوامر البرنامج قبل تحميله بالذاكرة**، حيث **نضيف رقم بداية عنوان المنطقة الذي سيحمل فيها هذا البرنامج إلى كل أوامر البرنامج.**

مثلا إذا حمل البرنامج بمنطقة تبدأ **بالعنوان 550**، فكل **أمر** سيضاف إليه الرقم **550** فإذا كان البرنامج يتعامل مع **متغير بالعنوان 68** كما في المثال السابق **فسيتغير العنوان** إلى **618=68+550**، حيث يتم استخدام **مسجل الاساس (Base)** لتخزين بداية المنطقة التي سيتم وضع البرنامج بها.

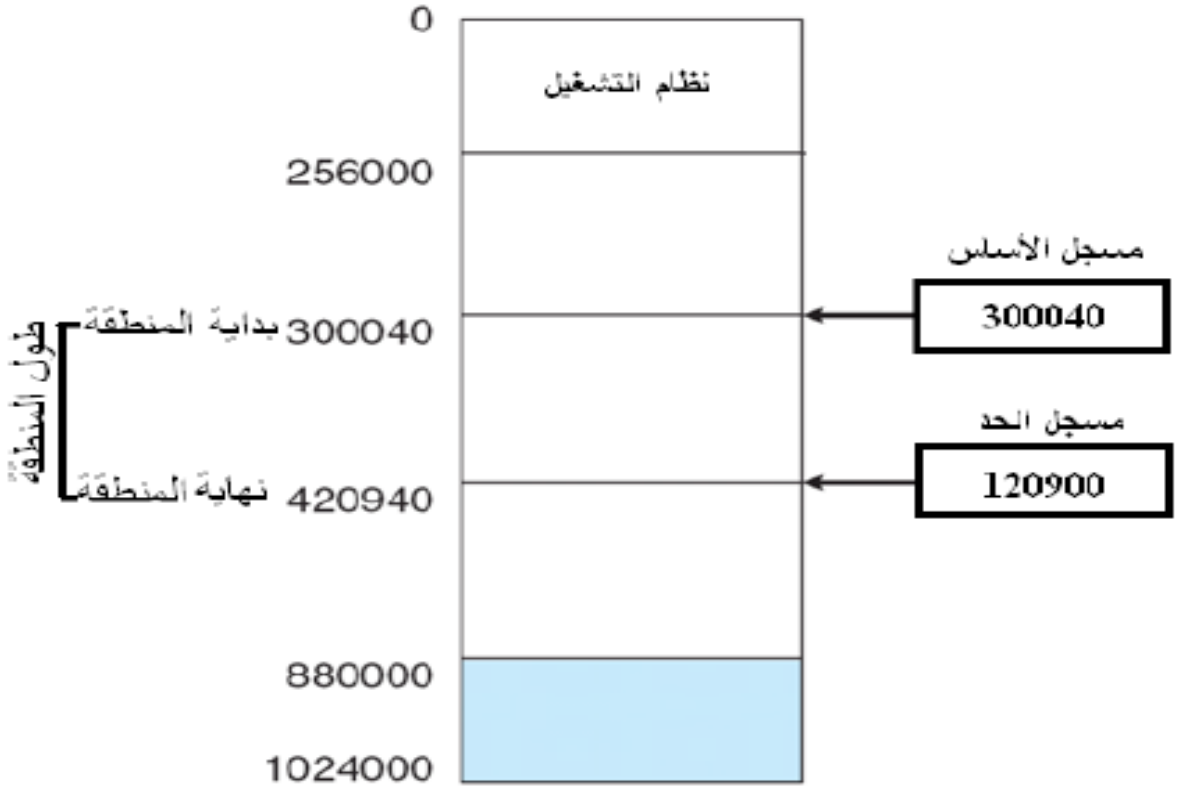
بهذه الطريقة تحل مشكلة تغيير الموقع.

- **مشكلة الحماية Protection**

وصول **برنامج إلى منطقة برنامج آخر يسبب مشكلة تداخل قد تؤدي إلى تنفيذ خاطئ لهذه البرامج**. فمثلا قد تستخدم **برامج تخريبيه (malicious program)** وذلك بإنشاء أوامر تمكنها من **القفز (jump)** إلى أي مكان بالذاكرة. وبما أننا نستخدم **عناوين حقيقية** ، فليس هنالك طريقة لتوقيف مثل هذه البرامج من الوصول إلى أي خلية والكتابة أو القراءة منها.

حل مشكلتي تغيير المواقع والحماية

هنالك حل يمكن استخدامه لمعالجة مشكلتي **إعادة تغيير المواقع والحماية** في آن واحد، ألا **وهو استخدام مسجلي الأساس (base) والطول (limit) عند تحميل أي برنامج** سيحتوي **مسجل الأساس** على عنوان **بداية المنطقة** التي سيوضع بها البرنامج، **ومسجل الطول** سيخزن به **طول هذا المنطقة** كما في **الشكل 5.11**



الشكل 5.11 : استخدام مسجلي الاساس والحد

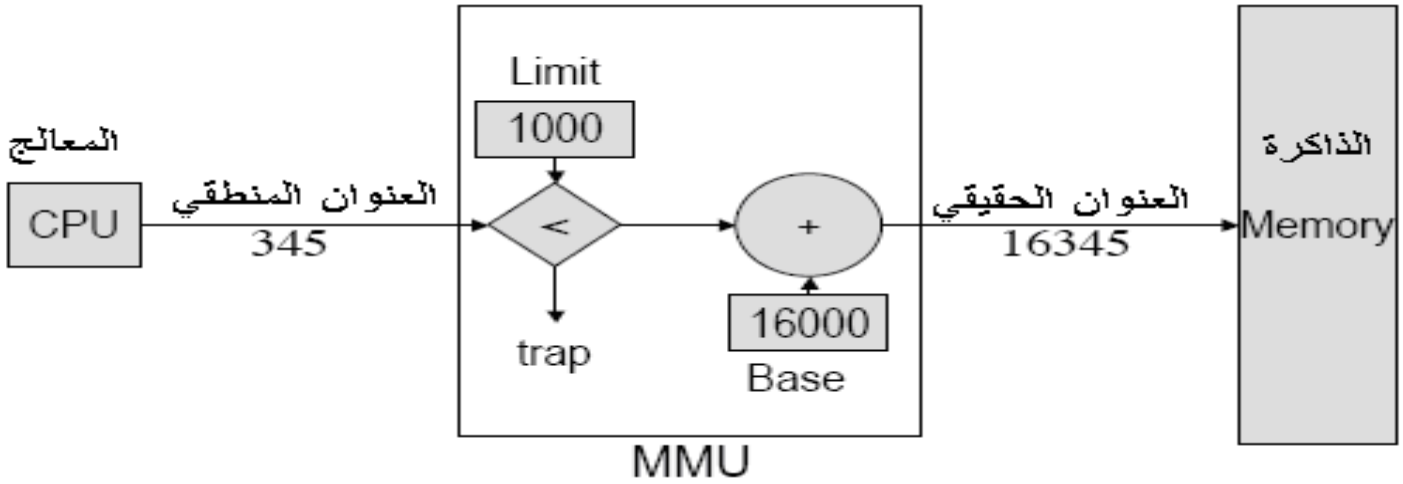
يستخدم المعالج **عناوين منطقية** (**ترقيم تسلسلي من 0 إلى طول الحد**) للتعامل مع البرنامج، وسيتم **إضافة** محتوى **مسجل الأساس** (**عنوان بداية المنطقة**) إلى **العنوان المنطقي** قبل إرساله للذاكرة. مثلا إذا أردنا تنفيذ أي **أمر** يتعامل مع الذاكرة، مثل **الأمر (Load 345)** ، فإنه لابد من إجراء الخطوات التالية قبل تنفيذ:

- هل الرقم **345** أكبر من **طول المنطقة** (**limit**)؟ (**للحماية**)
- إذا كانت الإجابة **نعم** سيرسل إشعار بأن **العنوان خطأ** (**trap**: address error.)
- إذا كانت الإجابة **لا** سنحول **العنوان المنطقي إلى عنوان حقيقي** وذلك **بإضافة** محتوى **مسجل الأساس للعنوان** الذي يستخدمه الأمر (انظر الشكل 5.12):

Load (345+16000)

ثم يرسل إلى الذاكرة، الشكل 5.12

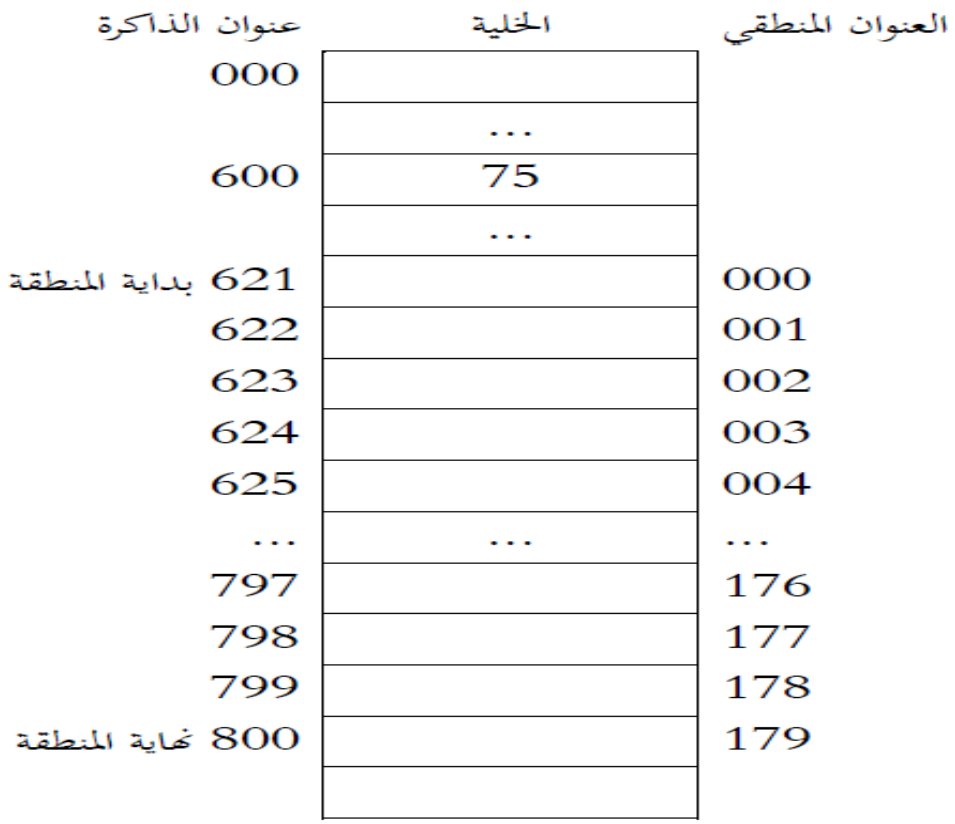
من **عيوب هذه الطريقة** أنه لابد من إجراء الخطوات أعلاه على كل أمر يتعامل مع الذاكرة مما يتسبب في **أبطاء التنفيذ**.



الشكل 5.12 : تحويل العنوان المنطقي الى الحقيقي

العناوين المنطقية (*Logical addresses*)

تتكون الذاكرة من خلايا مصطفة وراء بعضها البعض وتأخذ كل خلية **عنوان غير متكرر يسمى العنوان الحقيقي** (**تسمى أحيانا عناوين فيزيائية physical addresses**)، هذه **العناوين** هي التي **تستخدمها الذاكرة**. **أما المعالج والبرامج** فيستخدم **عناوين منطقية** (**تسمى أحيانا عناوين ظاهرية - virtual addresses**) هذه العناوين تبدأ من الصفر وتزداد تسلسليا إلى نهاية البرنامج أو نهاية المنطقة. الشكل 5.13.



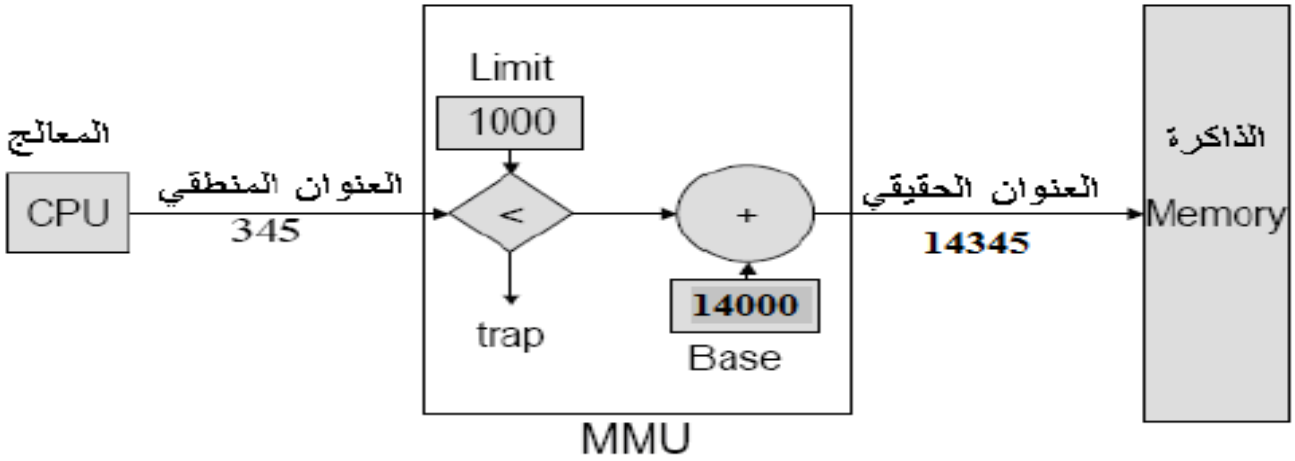
الشكل 5.13 :
العناوين الحقيقية والعناوين المنطقية

السؤال هنا من يقوم بعملية تحويل العنوان ؟

الإجابة: وحدة إدارة الذاكرة (Memory Management Unit -MMU).

ما هي وحدة إدارة الذاكرة (MMU)؟

هي عبارة عن **جهاز صغير** (**عتاد**) موجود بالمعالج **يقوم بتحويل العناوين المنطقية التي يستخدمها المعالج إلى عناوين حقيقية**، بحيث لا يهتم ولا يعرف المعالج كيف يتم ذلك، فالمعالج يقوم بإرسال عنوانه المنطقي إلى **MMU** (مثلا العنوان رقم **345**) ، فيقوم **MMU بتحويل العنوان المنطقي إلى حقيقي** ثم يرسله للذاكرة (**العنوان - 14345**) ، فتصل المعلومة للمعالج وهو لا يدري موقعها الأصلي بالذاكرة، **الشكل 5.14.**



الشكل 5.14 : تحويل العنوان المنطقي الى حقيقي

* تنبيه :

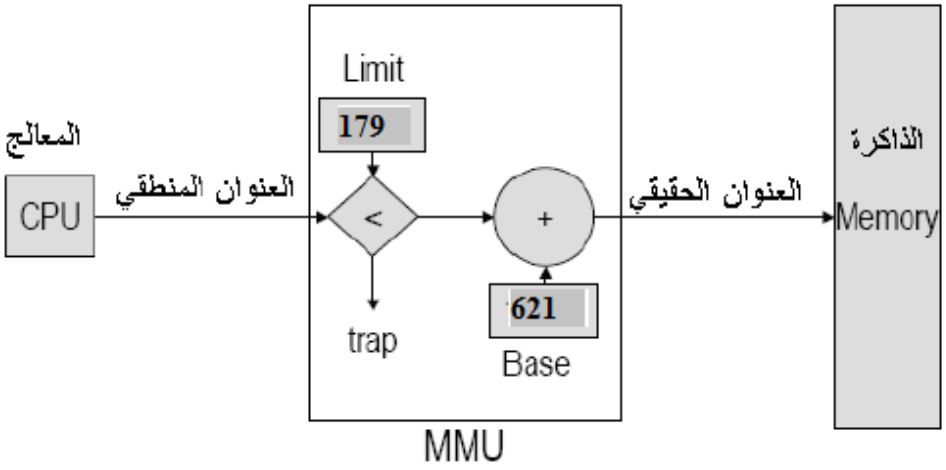
- **العنوان المنطقي** : يبدأ من **القيمة 0** إلى **قيمة مسجل الحد**.
- **العنوان الحقيقي** : يبدأ من مسجل الأساس إلى (**مسجل الأساس + قيمة مسجل الحد**).

- هل **العنوان المنطقي** الذي **تساوي** قيمته **قيمة مسجل الطول** يعتبر **عنوان صحيح** ؟
  لا، لأن أكبر **عنوان منطقي** يجب أن يكون أقل من **مسجل الحد بواحد**.

مثال

وضح كيف نحول العناوين **المنطقية** التالية إلى **حقيقية** : - بالرجوع **للشكل 5.15**.

- 4
- 177
- 200



الشكل 5.15

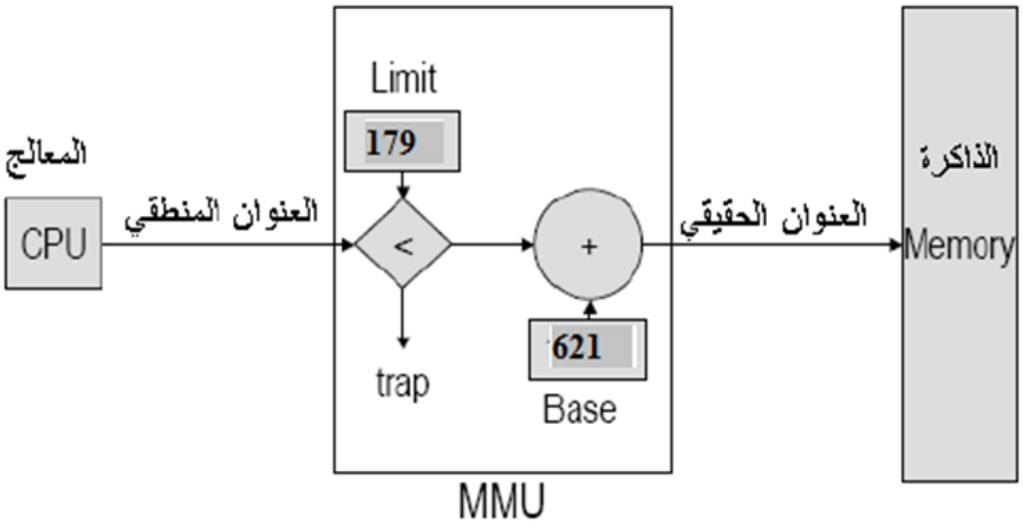
الحل

- 625 = 621 + 4
- 798 = 621 + 177
- **خطأ** لان **200 > 179** (أي ان العنوان المنطقي > من مسجل الحد)

مثال (معكوس)

حول العناوين **الحقيقية** التالية الى **منطقية**:

- 625
- 798
- 1000



الحل

- 625 - 621 = **4**
- 798 - 621 = **177**
- 1000-621 = **379** (**خطأ** لان العنوان الحقيقي خارج نطاق العملية.)

الذاكرة بالتصفح (Paging)

نلاحظ أنه إذا قسمنا البرنامج لجزأين كبيرين فإن:

- يتطلب كل جزء فراغ كبير لتحميله به.
- المبادلة ستكون بطيئة جدا وذلك لكبر حجم الأجزاء المتبادلة.
- إذا احتجنا إلى جزئية صغيرة من النصف الذي بالقرص فسنضطر إلى تبديله

التصفح (Paging)

تقسيم البرنامج إلى أجزاء صغيرة متساوية في الحجم تسمى صفحات (pages)، وتقسم الذاكرة إلى مناطق صغيرة متساوية في الحجم (ومساوية لحجم الصفحة) تسمى إطارات (frames). بحيث يكون كل إطار قادرا على تخزين صفحة.

لتنفيذ برنامج بعدد ن صفحة، فسنحتاج إلى ن إطار فارغ بالذاكرة، وإلا سنحتاج إلى ذاكرة ظاهرية.

قد تنشأ فراغات داخلية في Internal fragmentation في ذاكرة الصفحات.

مثلا إذا كان حجم الصفحة 4kb وكان لدينا برنامج حجمه 110kb ، فإننا سنقسم البرنامج كالتالي:

27 = 110/4 الباقي 2، إذن سأحتاج إلى 27 صفحة بالإضافة إلى الباقي من البرنامج وهو 2kb فأضطر إلى وضعها في صفحة (وبما أن الصفحة حجمها أربعة فإن الباقي وهو 2kb سنضعه في صفحة ويكون لدينا فراغ داخلي حجمه 2kb)، لذلك سنحتاج إلى 28 صفحة للبرنامج مع فراغ داخلي حجمه 2kb في الصفحة الأخيرة.

التعامل مع العناوين في الصفحات

يتكون العنوان المنطقي من شقين:

- عنوان الصفحة (رقم الصفحة).
- عنوان الخانة داخل الصفحة (الإزاحة offset).

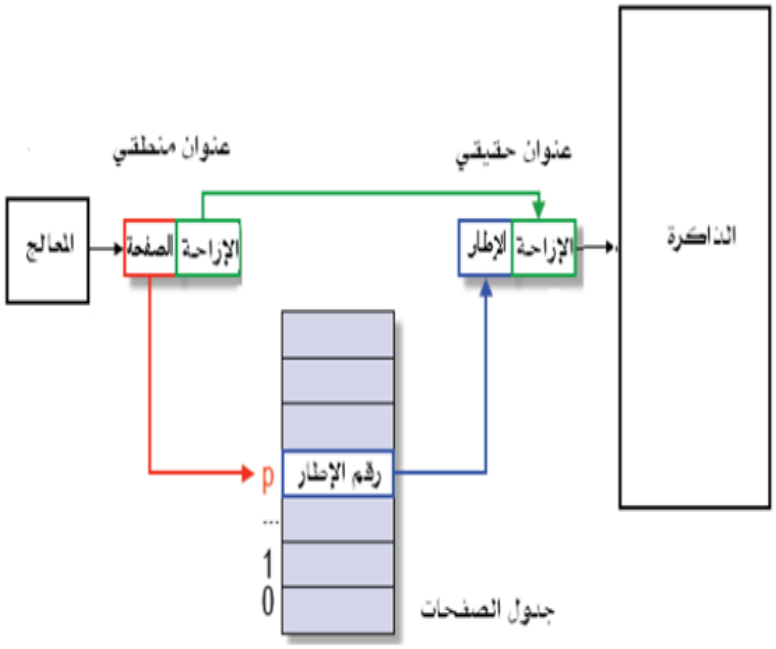
رقم الصفحة	الإزاحة
p	*d*

شكل رقم (5.16): العنوان المنطقي.

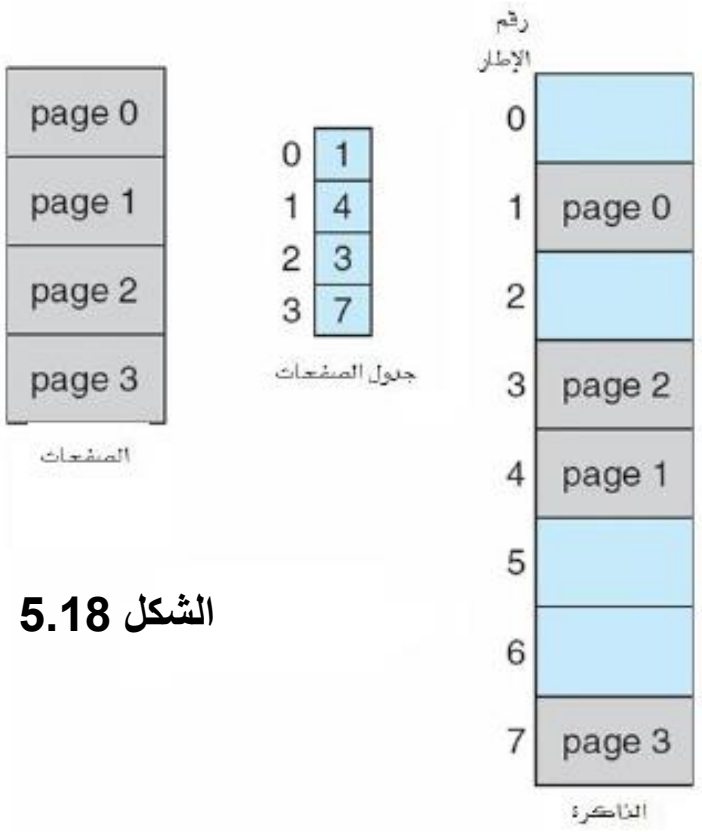
في **الشكل 5.17** يستخدم **المعالج عنوان الصفحة ومكان الخانة داخل الصفحة (الإزاحة)**، فيقوم الجهاز بالبحث عن **رقم الإطار** الذي وضعت به الصفحة بالذاكرة من **جدول الصفحات**، حيث يمثل **عنوان الصفحة فهرس** نصل به إلى **رقم الإطار** الذي تخزن به الصفحة في الذاكرة.

ندمج رقم الإطار مع الإزاحة فينتج العنوان الحقيقي.

يتم إنشاء **جدول صفحات** لكل عملية بحيث يحتوي هذا الجدول على **رقم الصفحات** التي تم تحميلها بالذاكرة **وأرقام الإطارات** التي وضعت فيها هذه الصفحات، كما في الشكل 5.18.



شكل رقم (5.17): جهاز تحويل العنوان منطقي إلى حقيقي.



الشكل 5.18

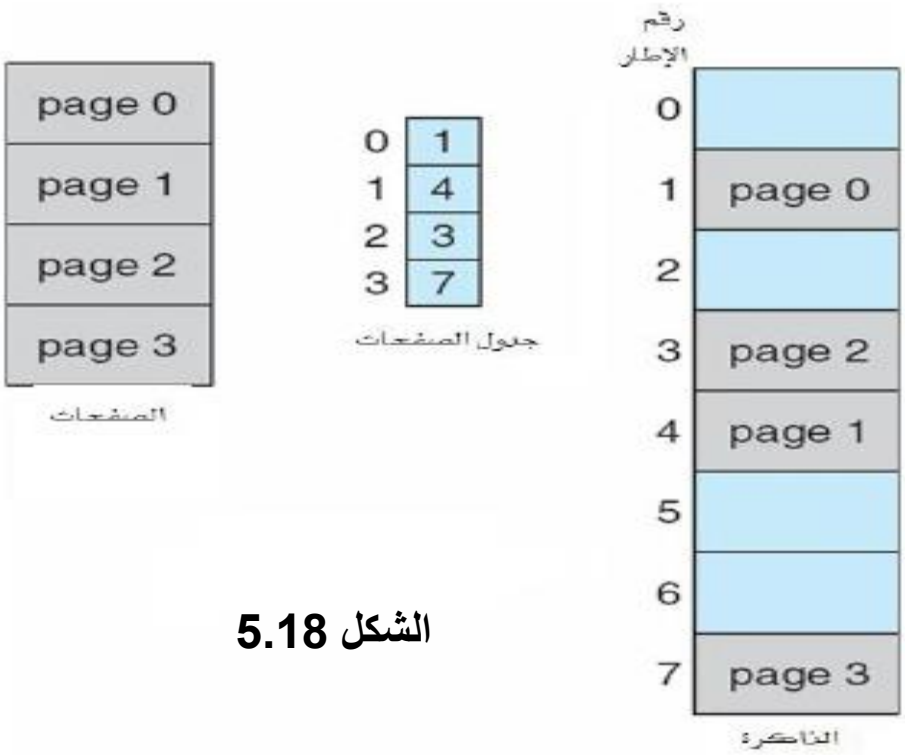
مثال حول العناوين المنطقية التالية إلى عناوين حقيقية (بالرجوع للشكل (5.18)):

2	10	1	0	3	3	0	7

الحل

سنبحث عن الإطار الذي توجد به الصفحة من جدول الصفحات ونستبدله برقم الصفحة (ونضع الغزاحة كما هي) فتكون العناوين الحقيقية كما يلي:

3	10	4	0	7	3	1	7



الشكل 5.18

مثال : في هذا المثال هناك معلومات بالصفحات و المطلوب معرفة مكان المعلومة بالذاكرة على سبيل المثال اين يتواجد الحرف a ؟ الشكل 5.19.

الحل

الحرف a يوجد بالصفحة الاولى (رقم صفر)، ويوجد في أول خانة داخل الصفحة (الإزاحة = صفر)، سنقوم بتحويل رقم الصفحة إلى رقم الإطار من جدول الصفحات حيث سنجد أن الصفحة رقم صفر توجد في الإطار رقم خمسة، لنصل للمعلومة مباشرة نضرب رقم الإطار في طول الصفحة (4) ونضيف إليه الإزاحة:

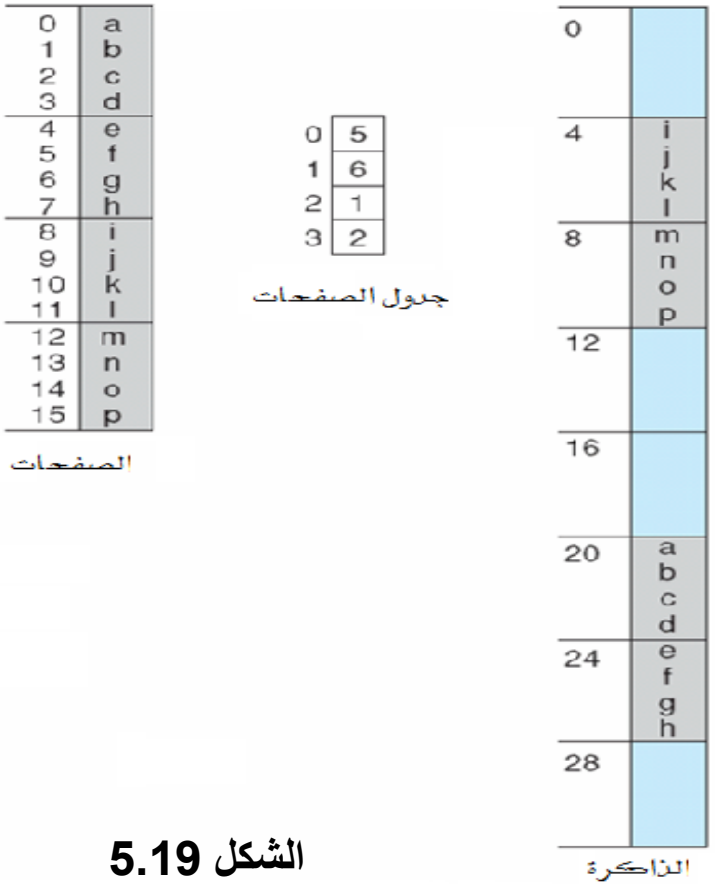
$$5 \times 4 + 0 = 20$$

فنجد أن العنوان رقم 20 بالذاكرة يحتوي فعلا على الحرف a

أيضا الحرف p مثلا في الصفحة رقم 3، في الخانة رقم 3 الصفحة 3 توجد بالإطار 2 (من جدول الصفحات)، لمعرفة مكان الحرف p بالذاكرة سنجري العملية التالية:

$$2 \times 4 + 3 = 11$$

ونجد 11 هي مكان الحرف p بالذاكرة.



الشكل 5.19

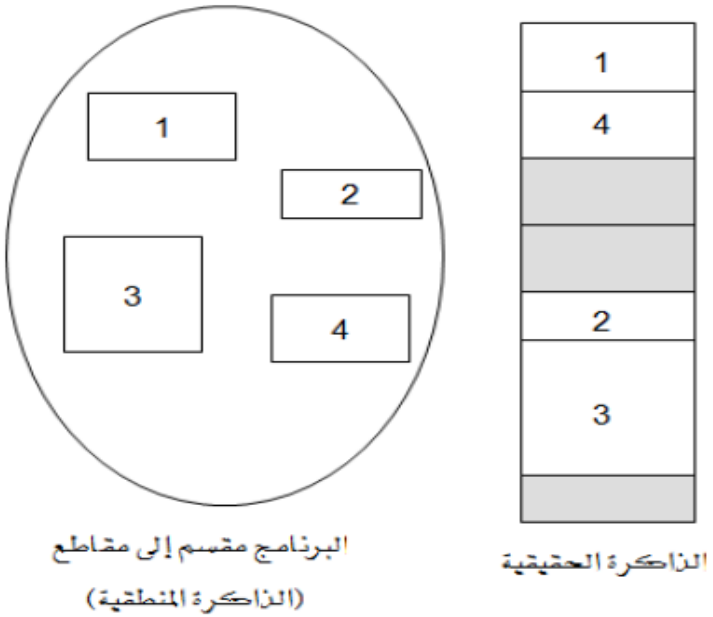
مكان المعلومة = رقم الاطار * طول الصفحة + الازاحة

التقطيع (segmentation)

هو طريقة لإدارة الذاكرة تدعم نظرة المستخدم للذاكرة، حيث نقوم **بتقسيم البرنامج إلى أجزاء منطقية**، **فالبرنامج الرئيسي** يكون في مقطع، **الدوال** في مقطع **والبيانات** في مقطع آخر، وهكذا.

يعتبر **المقطع** وحدة منطقية مثل البرنامج الرئيسي، دالة معينة، كائن، المتغيرات العامة والخاصة، المكدس (stack)، المصفوفات.

- بعد تقسيم البرنامج إلى مقاطع **نضع كل مقطع في خانة بالذاكرة** (الشكل 5.20).



الشكل 5.20

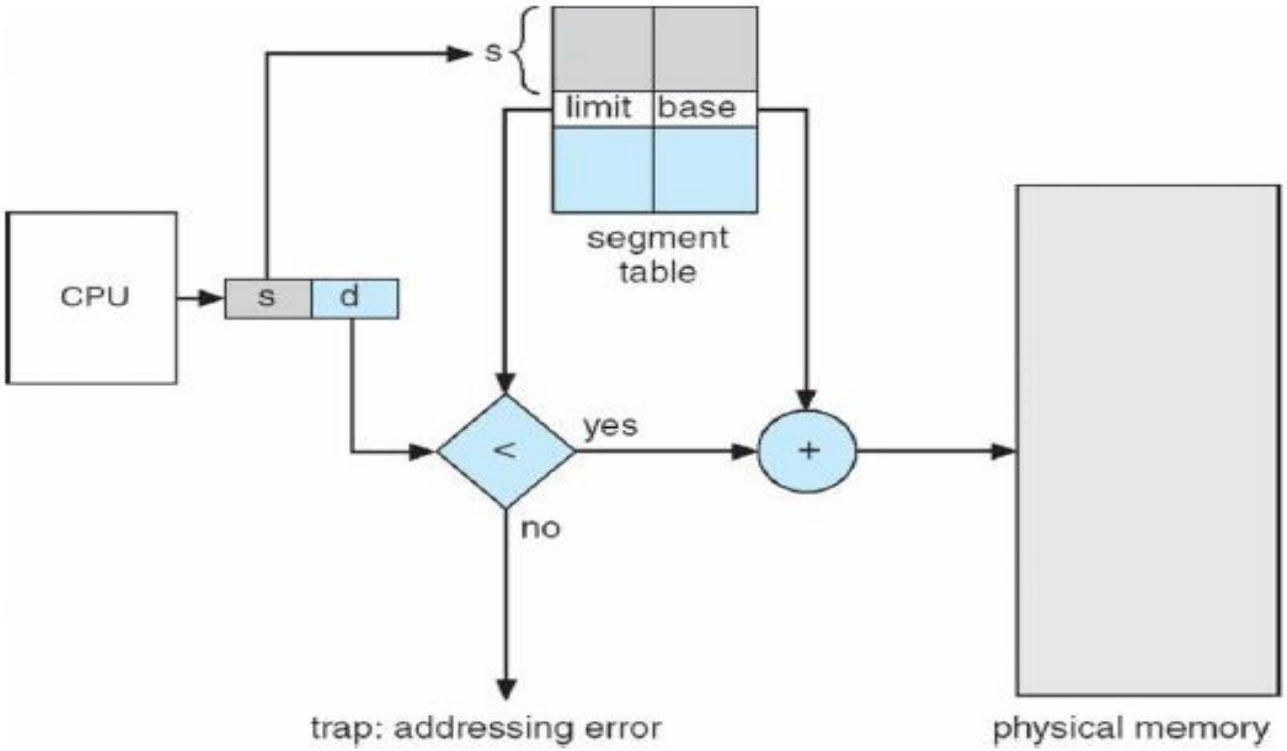
العناوين في التقطيع

يتكون **العنوان المنطقي** من : (**رقم المقطع**، **ورقم الإزاحة**)

segment-number، **offset** .

تتم **عملية التحويل بين العنوان المنطقي والعنوان الحقيقي باستخدام مسجلي أساس ومسجل الطول**، حيث يحتوي **مسجل الأساس (base) عنوان بداية المقطع** في الذاكرة **ومسجل الطول** يحتوي على **طول المقطع (limit)** توضع مسجلات الأساس والطول لكل المقاطع في جدول واحد يسمى **جدول المقاطع (segment table)**، حيث هنالك جدول مقاطع لكل عملية.

هنالك جهاز يقوم بعملية تحويل العناوين من منطقية إلى حقيقية يعمل كما في الشكل 5.21)



الشكل 5.21

مثال: الشكل 5.22 يوضح برنامج مكون من **5 مقاطع**، ويبين جدول المقاطع مكان كل مقطع بالذاكرة الرئيسية.

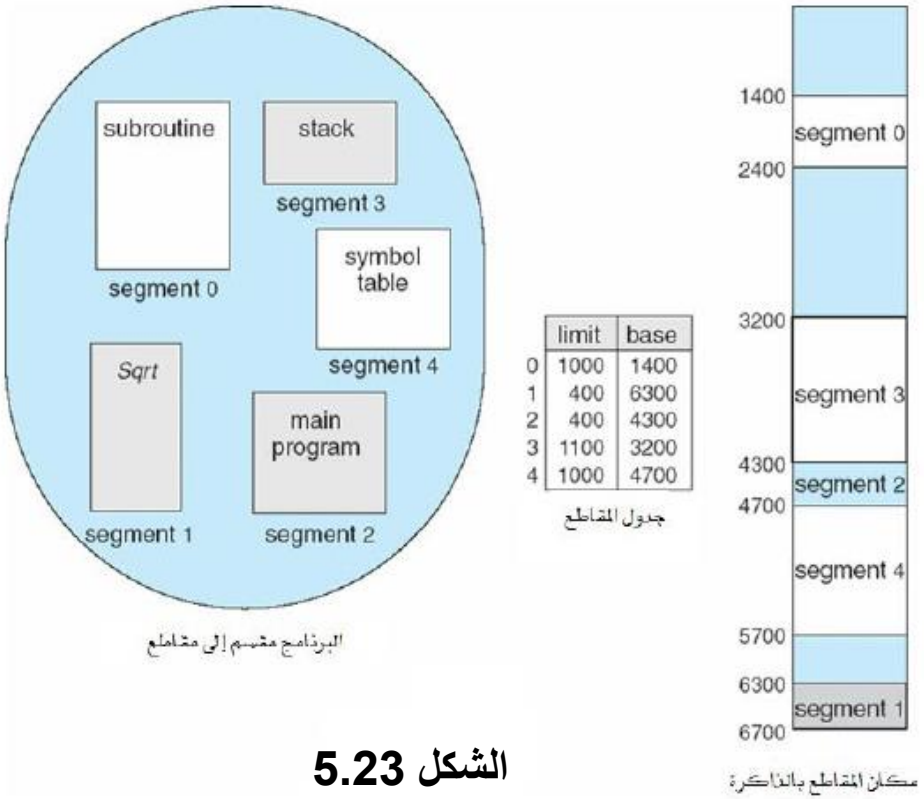
شكل 5.23 : لتوضيح كيف يتم التحويل من العنوان **المنطقي** إلى **الحقيقي**، لنحول العنوان المنطقي التالي إلى حقيقي:

0	20

هذا العنوان يعني أنني أريد معلومة من **المقطع رقم 0** **الإزاحة** رقم **20**.

الحل

- **أبحث** في جدول المقاطع عن قيمة مسجل الأساس **(base)** للمقطع **0** وهي **1400** .
- **قارن** قيمة **الإزاحة** مع مسجل الطول **(limit)** (أي هل **20** أصغر من **1000**) ؟ إذا كانت الإجابة **نعم** فسأقوم بالتحويل وإلا سيكون هنالك خطأ في الإزاحة.
- إذاً **العنوان الحقيقي** سيكون **محتوى مسجل الأساس + الإزاحة** أي
- **1420 = 1400 + 20**



الشكل 5.23

تمارين محلولة

1. إذا كان لدينا جدول المقاطع (segment table) التالي:

رقم المقطع	الأساس (base)	الحد/الطول (limit)
0	300	600
1	7000	20
2	0	300
3	1000	6000
4	900	100

حول العناوين المنطقية التالية إلى عناوين حقيقية :

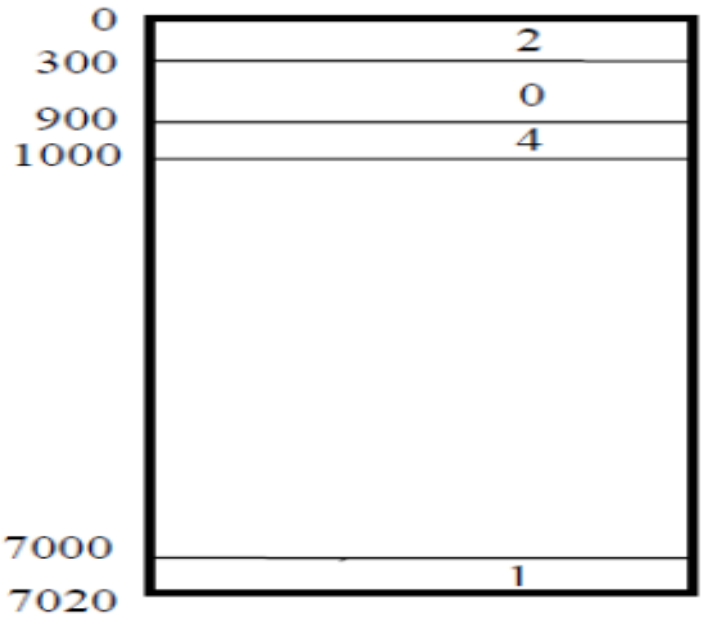
- 0, 500
- 1, 10
- 2, 500
- 3, 2400
- 4, 99

أرسم شكل الذاكرة مع توضيح هل توجد فراغات خارجية أم لا ؟

الحل:

```
0,500   →   800
1,10    →   7010
2,500   →   –
3,2400  →   3400
4,99    →   999
```

العنوان المنطقي 2,500 لا يمكن تحويله لأن 500 أكبر من طول المقطع (الحد هو 300).



من الشكل أعلاه نجد أنه لا يوجد فراغ خارجي.

1. أذكر خمسة من أهداف إدارة الذاكرة ؟
2. ما الفرق بين العنوان المنطقي والعنوان الحقيقي ؟
4. ما الفرق بين الذاكرة بالتجزئة الثابتة والذاكرة الديناميكية ؟ و ماهي عيوب كل منهما ؟
5. ماهي الفراغات وما علاجها ؟

7.2. إذا كان لدينا ذاكرة مقسمة إلى الأجزاء التالية:

100، 500، 200، 300، 600

وأردنا تحميل العمليات التالية فيها:

P1=5, p2=100, p3=300, p4=500, p5=200, P6=600

فإن :

- P6 ستنتظر في طريقة الأول ff
- P6 ستنتظر في طريقة الأنسب bf،
- P6 ستنتظر في طريقة الأسوأ wf