Advanced Operating System

Lecture Notes

Dr. Professor, J.M. Khalifeh

قسم المعلوماتية

الوحدة الأولى

Unit-1

CPU Scheduling

(advanced Topics)

ملخص

إن جدولة وحدة المعالجة المركزية باستخدام الجدولة متعددة المستويات MLQ هي نوع من أنواع الجدولة التي يتم تطبيقها على مستوى نظام التشغيل بهدف تقسيم أنواع العمليات ومن ثم القدرة على إدارتها بشكل صحيح. حيث يتم تجميع العمليات في عدة طوابير باستخدام MLQ استنادًا إلى بارامترات معروفة مثل الأولوية أو الحجم اللازم في الذاكرة أو النوع. كل طابور له سياسة جدولة خاصة به والعمليات الموجودة في نفس الطابور متشابهة جدًا أيضًا. وسنقوم في هذه الوحدة التعرف إلى الجدولة متعدد المستويات.

كما أن أنظمة المعالجات المتعددة أصبحت شائعة، وقد وجدت طريقها إلى أجهزة سطح المكتب وأجهزة الكمبيوتر المحمولة وحتى الأجهزة المحمولة. إن ظهور المعالجات متعددة النواة، حيث يتم حزم نوى وحدة المعالجة المركزية المتعددة على شريحة واحدة، هو مصدر هذا الانتشار؛ أصبحت هذه الرقائق شائعة حيث واجه مهندسو الكمبيوتر صعوبة في جعل وحدة المعالجة المركزية واحدة أسرع بكثير دون استخدام (الكثير) من الطاقة. وبالتالي، لدينا جميعًا الآن عدد قليل من وحدات المعالجة المركزية المتاحة لنا، وهو أمر جيد، أليس كذلك؟

بالطبع، هناك العديد من الصعوبات التي تنشأ مع وجود أكثر من وحدة معالجة مركزية واحدة. أحد الصعوبات الأساسية هو أن التطبيق المعني يستخدم وحدة معالجة مركزية واحدة فقط؛ إن إضافة المزيد من وحدات المعالجة المركزية لا يجعل هذا التطبيق الفردي يعمل بشكل أسرع. ولحل هذه المشكلة، سيتعين علينا إعادة كتابة تطبيقنا ليعمل بالتوازي، ربما باستخدام المسالك. يمكن للتطبيقات متعددة المسالك توزيع العمل عبر وحدات معالجة مركزية متعددة وبالتالي تعمل بشكل أسرع عند توفير المزيد من موارد وحدة المعالجة المركزية.

تساعدنا الجدولة في الأنظمة متعددة النوى على إنشاء أنظمة متزامنة لتوزيع المسالك على معالجات متعددة النوى وأنظمة متعددة المعالجات. نظام المعالجات متعددة النوى هو معالج واحد به نوى تنفيذ متعددة في شريحة واحدة. وعلى النقيض من ذلك، يحتوي نظام المعالجات المتعددة على معالجات متعددة على اللوحة الأم أو الشريحة.

يتم في الزمن الحقيقي تحديد الترتيب الذي يتم به تنفيذ المهام في الزمن الحقيقي على وحدة المعالجة المركزية. وتضمن إعطاء الأولوية للمهام ذات الأولوية الأعلى على المهام ذات الأولوية الأقل، وتلبية قيود الزمن.

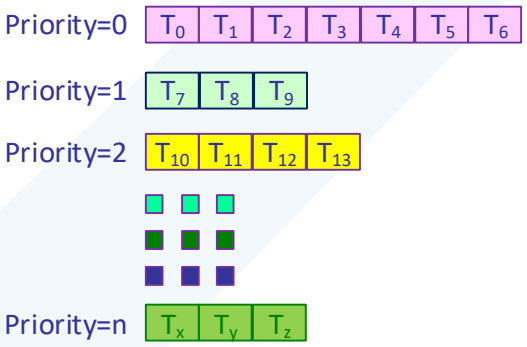
سنقوم في هذه الوحدة بإلقاء الضوء على الأمور المطروحة أعلاه.

أهداف الوحدة

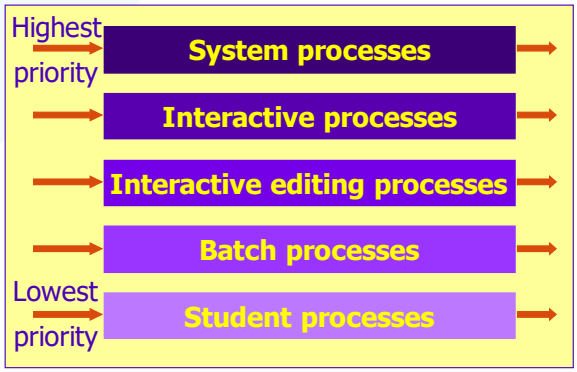
- التعرف على خوارزميات الجدولة متعددة المستويات.
- فهم آليات جدولة المسالك
- التعرف على آليات الجدولة في الأنظمة متعددة المعالجات ومتعددة النوى وفهم آلياتها
- فهم الجدولة في الزمن الحقيقي.

الجدولة متعددة المستويات Multilevel Queue Scheduling

في كل من جدولة الأولوية والجدولة الدورية RR، يمكن وضع جميع العمليات في قائمة انتظار واحدة، ثم يقوم المجدول باختيار العملية ذات الأولوية الأعلى للتشغيل. واعتمادًا على كيفية إدارة قوائم الانتظار، قد يكون من الضروري إجراء بحث O(n) لتحديد العملية ذات الأولوية الأعلى. وفي الممارسة العملية، غالبًا ما يكون من الأسهل وجود قوائم انتظار منفصلة لكل أولوية مميزة، وجدولة المستويات ليتم تنفيذها وفقا لأولوياتها. ويتضح ذلك في الشكل1. يعرف هذا النهج باسم قائمة الانتظار متعددة المستويات حيث أنه إذا كانت هناك عمليات متعددة في قائمة الانتظار ذات الأولوية الأعلى، يتم تنفيذها بترتيب دوري. وفي الشكل الأكثر عمومية لهذا النهج، يتم تعيين الأولوية بشكل ثابت لكل عملية، وتظل العملية في نفس قائمة الانتظار طوال مدة وقت تشغيلها.



شكل1: وجود طابور محدد لكل مستوى من الأولويات.



شكل 2: مثال على توزيع مستويات الجدولة وفق الأولويات.

يمكن أيضًا استخدام خوارزمية جدولة قائمة انتظار متعددة المستويات لتقسيم العمليات إلى عدة قوائم انتظار منفصلة بناءً على نوع العملية الشكل2. على سبيل المثال، يتم إجراء تقسيم بين العمليات الأمامية (التفاعلية) والعمليات الخلفية (الدفعات). هذان النوعان من العمليات لهما متطلبات مختلفة لوقت الاستجابة، وبالتالي قد يكون لهما جدول زمني مختلف. يمكن استخدام

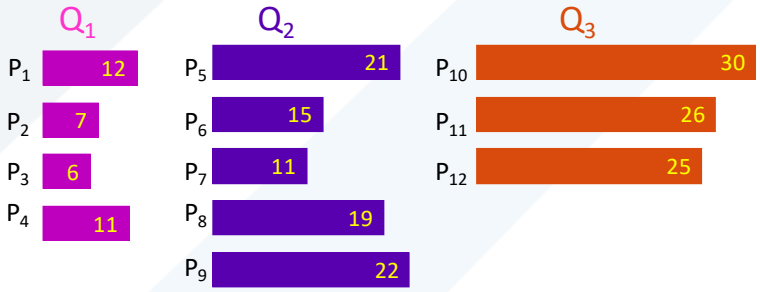
طوابير منفصلة للعمليات الأمامية والخلفية، وقد يكون لكل طابور خوارزمية جدولة خاصة به. يمكن جدولة طابور العمليات الأمامية بواسطة خوارزمية RR ، على سبيل المثال، بينما يتم جدولة طابور العمليات الخلفية بواسطة خوارزمية FCFS. ويؤمن هذا النهج اجراء الجدولة بين الطوابير.

هناك إمكانية أخرى تتمثل في تقسيم الوقت بين قوائم الانتظار. حيث يتم إعطاء وقت أكبر للعمليات في الطوابير الأعلى ذات الأولوية الأعلى. كما يمكن أن يمنع تنفيذ عملية من الطوابير الأدنى طالما هناك عملية في طابور ذي أولوية أعلى.

خوارزمية الجدولة متعددة المستويات مع التغذية الراجعة Multilevel Feedback Queue Scheduling

تعاني الخوارزمية السابقة بصيغتها المعروضة من عدم المرونة حيث قد تبقى بعض العمليات في المستويات الأدنى لفترات طويلة دون تنفيذ طالما هناك عمليات تدخل إلى الطوابير الأعلى. وهذا قد يؤدي إلى مجاعة العمليات في الطوابير الأدنى.

لذلك فإن السماح بنقل العمليات بين المستويات كأن يتم نقل المتبقي من العمليات التي يتم تنفيذها في المستويات الأعلى والتي تحتاج إلى زمن معالجة كبير إلى مستوي أدنى ليتم اكمالها وفق دورها في الطابور المتعلق بهذا المستوى. وهذا يتيح المجال أمام تنفيذ العمليات في المستويات الأدنى. أو قد يتم ترقية العمليات التي تقبع لفترة زمنية أكبر من حد معين لتحظى بأولوية أعلى مما هي عليه وبالتالي تكتسب الفرصة لتقليل زمن انتظارها وتنفيذها وفقا لخوارزمية المستوى الأعلى.



الشكل3 فصل العمليات وفق طوابير مختلفة.

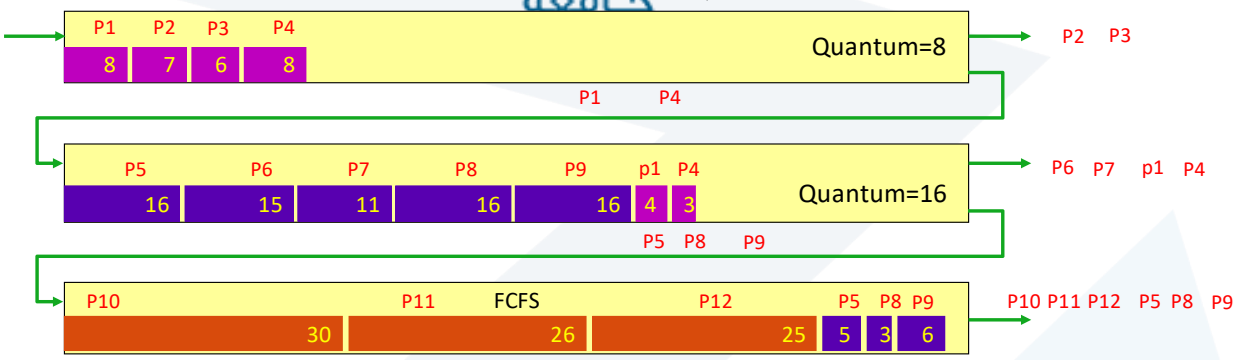
على سبيل المثال، إذا كان لدينا مجموعة عمليات تنتظر تنفيذها وفقا لثلاث قوائم، وفق الطوابير Q1 و Q2 و Q3. (الشكل 3). يقوم المجدول بدون تنفيذ هذه الخلفية أولاً بتنفيذ جميع العمليات في قائمة الانتظار Q1 فقط وعندما تكون قائمة الانتظار Q1 فارغة سيتم تنفيذ العمليات في قائمة الانتظار Q2. وعلى نحو مماثل، سيتم تنفيذ العمليات في قائمة الانتظار Q3 إذا كانت المستويات 1و2 فارغة. أي ستستبق العملية التي تصل إلى قائمة الانتظار 1 عملية في قائمة الانتظار 2.

يتم في مثالنا وضع عملية في قائمة الانتظار Q1 ويتم إعطاء أي عملية في قائمة الانتظار Q1 وقت كمي قدره q=8 مللي ثانية. وإذا لم تنتهي خلال هذا الوقت، يتم نقلها إلى ذيل قائمة الانتظار Q2. وحين اكمال الدور في المستوى Q1 يتم وضع العمليات التي لم تكتمل في ذيل المستوى الأدنى أي في ذيل المستوى Q2 ويتم إعطاء كل عملية في قائمة الانتظار Q2 قت كمي قدره q=16 مللي ثانية. إذا لم تكتمل أي عملية في المستوى Q2 يتم استبقاؤها ووضعها في قائمة الانتظار Q3 حيث يتم تنفيذ العمليات في قائمة الانتظار Q3 على أساس FCFS.

بشكل عام، يتم تعريف جدولة قائمة انتظار متعددة المستويات مع التغذية الراجعة بالمحددات التالية:

- عدد قوائم الانتظار
- خوارزمية الجدولة لكل قائمة انتظار
- الطريقة المستخدمة لتحديد متى يتم ترقية عملية إلى قائمة انتظار ذات أولوية أعلى
- أو الطريقة المستخدمة لتحديد متى يتم تخفيض رتبة عملية إلى قائمة انتظار ذات أولوية أقل

- الطريقة المستخدمة لتحديد قائمة الانتظار التي ستدخلها عملية ما عندما تحتاج هذه العملية إلى خدمة ما.



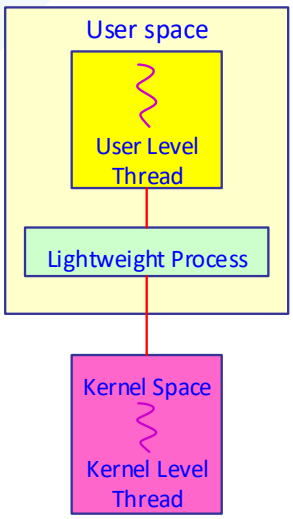
الشكل 4: متعدد المستويات تعليق طوابير.

إن تعريف جدولة قائمة انتظار متعددة المستويات مع التغذية الراجعة يجعلها أكثر خوارزميات جدولة وحدة المعالجة المركزية عمومية. ويمكن تكوينها لتتناسب مع نظام معين قيد التصميم. ومن المؤسف أنها أيضًا الخوارزمية الأكثر تعقيدًا، حيث يتطلب تحديد أفضل جدولة بعض الوسائل التي يمكن من خلالها تحديد القيم لجميع المعلمات.

جدولة المسالك Thread Scheduling

ملاحظة: *يتم ترجمة مصطلح Thread إلى اللغة العربية بمصطلحات عربية مختلفة مثل: مسار أو مسلك أو مسلك أو ترابط أو نيسب. سنستخدم هنا مصطلح مسلك.*

يتم في معظم أنظمة التشغيل الحديثة جدولة مسالك مستوى النواة بواسطة نظام التشغيل. في حين تتم إدارة مسالك مستوى المستخدم بواسطة مكتبة مسالك توفرها البيئات البرمجية المختلفة، وتكون النواة غير مدركة لها. ولتشغيل مسالك مستوى المستخدم على وحدة المعالجة المركزية، يجب في النهاية تعيينها إلى مسلك أو أكثر من مستوى النواة المرتبط بها. على الرغم من أن هذا التعيين قد يكون غير مباشر وقد يستخدم عملية خفيفة الوزن lightweight process LWP وذلك حسب نطاق التنافس ونطاق التخصيص كما في الشكل 5.



الشكل 5: تخصيص مسالك المستخدم

نطاق التنافس Contention Scope

تشير كلمة التنافس هنا إلى المنافسة بين مسالك مستوى المستخدم للوصول إلى موارد النواة. وبالتالي، يحدد هذا التحكم مدى حدوث التنافس. يتم تعريفه بواسطة مطور التطبيق باستخدام مكتبة المسالك.

يتم تصنيف نطاق التنافس إلى:

نطاق التنافس في العملية Process Contention Scope (PCS) :

يحدث التنافس بين المسالك داخل نفس العملية. تقوم مكتبة المسالك بجدولة مسلك PCS ذي الأولوية العالية للوصول إلى الموارد عبر LWPs المتاحة (الأولوية كما يحددها مطور التطبيق أثناء إنشاء المسلك).

نطاق التنافس في النظام System Contention Scope (SCS):

يحدث التنافس بين جميع المسالك في النظام. في هذه الحالة، يتم ربط كل مسلك SCS بكل LWP بواسطة مكتبة المسالك ويتم جدولته بواسطة مجدول النظام للوصول إلى موارد النواة.

في أنظمة التشغيل LINUX وUNIX، توفر مكتبة Pthread POSIX وظيفة Pthread_attr_setscope لتحديد نوع نطاق التنافس لمسلك أثناء إنشائه.

```
int Pthread_attr_setscope(pthread_attr_t *attr, int scope)
```

يشير المعامل الأول إلى المسلك الذي يتم تحديد النطاق له داخل العملية.

يحدد المعامل الثاني نطاق التنافس للمسلك المشار إليه. ويأخذ قيمتين.

```
PTHREAD_SCOPE_SYSTEM
PTHREAD_SCOPE_PROCESS
```

إذا لم تكن قيمة النطاق المحددة مدعومة بواسطة النظام، فستقوم الدالة بإرجاع ENOTSUP.

مجال التخصيص

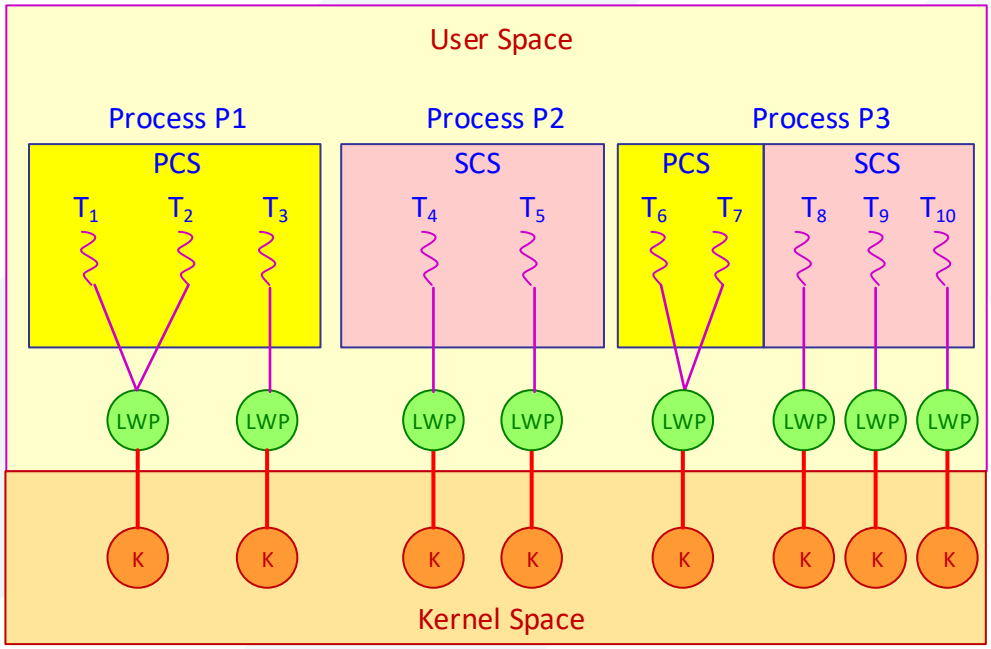
مجال التخصيص هو مجموعة من مورد واحد أو أكثر ينافس عليها المسلك. في نظام متعدد النواة، قد يكون هناك مجال تخصيص واحد أو أكثر حيث يتكون كل منها من نواة واحدة أو أكثر. يمكن أن يكون User-level Thread (ULT) واحدًا جزءًا من مجال تخصيص واحد أو أكثر. نظرًا لهذا التعقيد العالي في التعامل مع واجهات البنية التحتية للأجهزة والبرامج، لا يتم تحديد هذا التحكم. ولكن بشكل افتراضي، سيكون للنظام متعدد النواة واجهة تؤثر على مجال التخصيص لمسلك.

لنفترض أن هناك سيناريو، وهو نظام تشغيل به ثلاث عمليات P1 وP2 وP3 و10 مسالك، والمسالك على مستوى المستخدم (T1 إلى T10) مع مجال تخصيص واحد كما في الشكل 6. سيتم توزيع 100% من موارد وحدة المعالجة المركزية بين العمليات الثلاث. تعتمد كمية موارد وحدة المعالجة المركزية المخصصة لكل عملية ولكل مسلك على نطاق التنافس وسياسة الجدولة وأولوية كل مسلك يحددها مطور التطبيق باستخدام مكتبة المسالك وتعتمد أيضًا على جدول النظام. هذه المسالك على مستوى المستخدم لها نطاق تنافس مختلف.

في هذه الحالة، يحدث التنافس على مجال التخصيص على النحو التالي:

العملية P1

ستتنافس جميع مسالك العملية P1 في مستوى PCS وهي T1 و T2و T3 فيما بينها. يمكن لمسالك PCS لنفس العملية مشاركة LWP واحد أو أكثر. تتشارك T1 و T2في LWP ويتم تخصيص T3 لـ LWP منفصل. بين T1 و T2، يعتمد تخصيص موارد النواة عبر LWP على جدولة الأولوية الاستباقية بواسطة مكتبة المسالك. سيستبق المسلك ذو الأولوية العالية المسالك ذات الأولوية المنخفضة. بينما لا يمكن للمسلك T1 للعملية P1 أن يستبق المسلك T7 للعملية P3 حتى إذا كانت أولوية T1 أكبر من أولوية T7 . إذا كانت الأولوية متساوية، فإن تخصيص ULT لـ LWPs المتاحة يعتمد على سياسة جدولة المسالك بواسطة مجدول النظام (وليس بواسطة مكتبة المسالك، في هذه الحالة).



الشكل 6: مثال على تخصيص المسالك

العملية P2

ستتنافس كل من مسالك SCS للعملية P2 T4 و T5 مع العملية P1 ككل ومع مسالك SCS للعملية P3 T8 و T9و T10.

سيقوم مجدول النظام بجدولة موارد النواة بين مسالك P1 و T4و T5و T8و T9و T10ومسالك (T6 PCS) و (T7للعملية P3 مع الأخذ في الاعتبار كل منها كعملية منفصلة. هنا، لا تتحكم مكتبة المسالك في جدولة ULT لموارد النواة.

العملية P3

مزيج من مسالك PCS و .SCS. نضع في الاعتبار أنه إذا خصص مجدول النظام 50% من موارد وحدة المعالجة المركزية للعملية P3، وبعد ذلك 25% من الموارد مخصصة لمسالك نطاق العملية والـ 25% المتبقية تخصص لمسالك نطاق النظام. سيتم تخصيص مسالك PCS وهي T6 و T7 للوصول إلى 25% من الموارد بناءً على الأولوية بواسطة مكتبة المسالك. ستقوم مسالك SCS وهي T8 و T9 و T10 بتقسيم الموارد بنسبة 25% فيما بينها والوصول إلى موارد النواة عبر LWP و KLT منفصلين. تتم جدولة SCS بواسطة مجدول النظام.

ملاحظة:

بالنسبة لكل استدعاء نظام للوصول إلى موارد kernel، يتم إنشاء مسلك على مستوى kernel وربطه بـ LWP منفصل بواسطة مجدول النظام.

Number of Kernel Level Threads = Total Number of LWP

Total Number of LWP = Number of LWP for SCS + Number of LWP for PCS
Number of LWP for SCS = Number of SCS threads
Number of LWP for PCS = Depends on application developer

هنا:

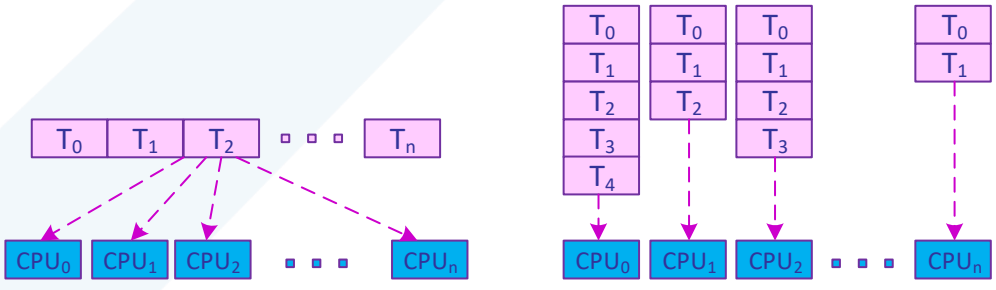
Number of SCS threads = 5
Number of LWP for PCS = 3
Number of SCS threads = 5
Number of LWP for SCS = 5
Total Number of LWP = 8 (=5+3)
Number of Kernel Level Threads = 8

جدولة المعالجات المتعددة *Multi-Processor Scheduling*

في الأنظمة التي تحتوي على أكثر من معالج واحد، تعمل جدولة المعالجات المتعددة على تخصيص المهام لوحدات المعالجة المركزية المتعددة. وهذا يعطي إنتاجية أعلى نظرًا لأنه يمكن معالجة العديد من المهام في وقت واحد في معالجات منفصلة. كما يتطلب الأمر تحديد وحدة المعالجة المركزية التي تتعامل مع مهمة معينة وموازنة الأحمال بين المعالجات المتاحة.

الأساليب المتبعة في جدولة المعالجات المتعددة

تتمثل إحدى الطرق في أن يتم التعامل مع كافة قرارات الجدولة ومعالجة الإدخال/الإخراج بواسطة معالج واحد يسمى الخادم الرئيسي بينما تقوم المعالجات الأخرى بتنفيذ كود المستخدم فقط. وهذا بسيط ويقلل من الحاجة إلى مشاركة البيانات. ويسمى هذا السيناريو بالكامل بالمعالجة المتعددة غير المتماثلة Asymmetric Multiprocessing الشكل 7 a وتستخدم الطريقة الثانية المعالجة المتعددة المتماثلة Symmetric Multiprocessing حيث يقوم كل معالج بالجدولة الذاتية الشكل 7 b. وقد تكون كافة العمليات في قائمة انتظار جاهزة مشتركة أو قد يكون لكل معالج قائمة انتظار خاصة به للعمليات الجاهزة. وتستمر عملية الجدولة من خلال جعل المجدول لكل معالج يفحص قائمة الانتظار الجاهزة ويختار عملية لتنفيذها.



(a) طابور عام مشترك (b) طابور لكل معالج

الشكل 7: تشكيل طوابير الجاهزية في الأنظمة متعددة المعالجات

الميل للالتزام بالمعالج "تقارب المعالج" *Processor Affinity*

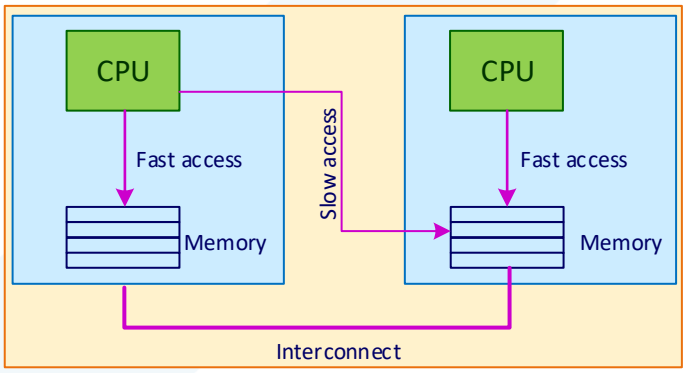
يعني أن العمليات تميل للالتزام مع المعالج الذي تعمل عليه حاليًا

تأثيرات معينة على ذاكرة التخزين المؤقت Caches. فالبيانات التي تم الوصول إليها مؤخرًا بواسطة العملية تملأ ذاكرة التخزين المؤقت للمعالج ونتيجة لذلك، غالبًا ما يتم تلبية الوصول المتتالي للذاكرة بواسطة العملية في ذاكرة التخزين المؤقت.

والآن إذا انتقلت العملية إلى معالج آخر، فيجب إبطال محتويات ذاكرة التخزين المؤقت للمعالج الأول وإعادة ملء ذاكرة التخزين المؤقت للمعالج الثاني. ونظرًا للتكلفة العالية المترتبة على إبطال ذاكرة التخزين المؤقت وإعادة ملؤها، تحاول معظم أنظمة SMP)المعالجة المتعددة المتماثلة) تجنب هجرة العمليات من معالج إلى آخر ومحاولة إبقاء العملية قيد التشغيل على نفس المعالج. وهذا ما يُعرف بالميل للالتزام بالمعالج. وهناك نوعان من تقارب المعالج:

- **التقارب الناعم Soft Affinity:** عندما يكون لدى نظام التشغيل سياسة محاولة إبقاء عملية قيد التشغيل على نفس المعالج ولكن دون ضمان قيامها بذلك، فإن هذا الموقف يسمى التقارب الناعم.
- **التقارب الثابت Hard Affinity:** يسمح التقارب الثابت للعملية بتحديد مجموعة فرعية من المعالجات التي قد تعمل عليها. تطبق بعض الأنظمة مثل Linux التقارب الناعم ولكنها توفر أيضًا بعض استدعاءات النظام مثل sched_setaffinity() التي تدعم التقارب الثابت.

يمكن أن تؤثر بنية الذاكرة الرئيسية للنظام على قضايا تقارب المعالج أيضًا. يوضح الشكل 8 بنية تتميز بالوصول غير المنتظم للذاكرة Non-uniform memory access (NUMA) حيث توجد شريحتا معالج ماديتان لكل منهما وحدة معالجة مركزية وذاكرة محلية خاصة بها. على الرغم من أن الترابط بين النظام يسمح لجميع وحدات المعالجة المركزية في نظام NUMA بمشاركة مساحة عنوان مادية واحدة، إلا أن وحدة المعالجة المركزية لديها وصول أسرع إلى ذاكرتها المحلية مقارنة بالذاكرة المحلية لوحدة معالجة مركزية أخرى. إذا كانت جدولة وحدة المعالجة المركزية وخوارزميات وضع الذاكرة في نظام التشغيل على دراية بـ NUMA وتعمل معًا، فيمكن تخصيص ذاكرة أقرب إلى مكان تواجد وحدة المعالجة المركزية لمسلك تم جدولته على وحدة معالجة مركزية معينة، وبالتالي توفير أسرع وصول ممكن إلى الذاكرة للمسلك.



الشكل 8: تنفيذ العمليات في الحالة NUMA

ومن المثير للاهتمام أن موازنة التحميل غالبًا ما تعاكس الميل إلى الاستفادة من فوائد تقارب المعالج. أي أن فائدة إبقاء المسلك قيد التشغيل على نفس المعالج هي أن المسلك يمكنه الاستفادة من وجود بياناته في ذاكرة التخزين المؤقت لهذا المعالج. يؤدي موازنة الأحمال عن طريق نقل مسلك من معالج إلى آخر إلى إزالة هذه الفائدة. على نحو مماثل، قد يؤدي نقل مسلك بين المعالجات إلى فرض عقوبة على أنظمة NUMA، حيث قد يتم نقل مسلك إلى معالج يتطلب أوقات وصول أطول إلى الذاكرة. بعبارة أخرى، هناك توتر طبيعي بين موازنة الحمل وتقليل أوقات الوصول إلى الذاكرة. وبالتالي، أصبحت خوارزميات الجدولة لأنظمة NUMA الحديثة متعددة النوى معقدة للغاية.

موازنة التحميل Load Balancing

موازنة التحميل هي الظاهرة التي تحافظ على توزيع عبء العمل بالتساوي عبر جميع المعالجات في نظام المعالجة المتماثلة SMP . موازنة التحميل ضرورية فقط في الأنظمة حيث يكون لكل معالج قائمة انتظار خاصة به من العمليات المؤهلة

للتنفيذ. عكس ذلك فإن موازنة التحميل غير ضرورية لأنه بمجرد أن يصبح المعالج خاملاً فإنه يستخرج على الفور عملية قابلة للتنفيذ من قائمة الانتظار المشتركة. في SMP المعالجة المتعددة المتماثلة، من المهم الحفاظ على توازن عبء العمل بين جميع المعالجات للاستفادة الكاملة من فوائد وجود أكثر من معالج واحد وإلا فإن معالجًا واحدًا أو أكثر سيبقى خاملاً بينما يكون لدى المعالجات الأخرى أحمال عمل عالية جنبًا إلى جنب مع قوائم المعالجات التي تنتظر وحدة المعالجة المركزية. هناك طريقتان عامتان لموازنة التحميل:

- **الترحيل بالدفع Push Migration:** في الترحيل بالدفع، تتحقق المهمة بشكل روتيني من الحمل على كل معالج وإذا وجدت خللًا في التوازن فإنها توزع الحمل بالتساوي على كل معالج عن طريق نقل العمليات من المعالجات المحملة إلى المعالجات الخاملة أو الأقل انشغالاً.
- **الترحيل بالسحبPull Migration :** يحدث عندما يقوم المعالج الخامل بسحب مهمة انتظار من معالج مشغول لتنفيذها.

الجدولة عند تعدد المعالجات غير المتجانسة Heterogeneous Multiprocessing Scheduling

في الأمثلة التي ناقشناها حتى الآن، تكون جميع المعالجات متطابقة من حيث قدراتها، مما يسمح لأي مسلك بالعمل على أي نواة معالجة. والفرق الوحيد هو أن أوقات الوصول إلى الذاكرة قد تختلف بناءً على موازنة الحمل وسياسات تقارب المعالج، وكذلك على أنظمة.NUMA

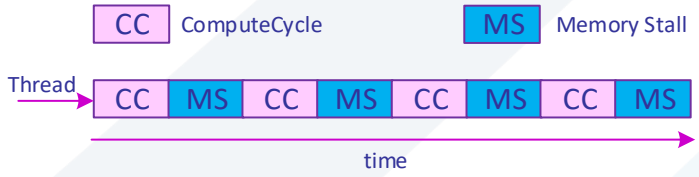
على الرغم من أن الأنظمة المحمولة تتضمن الآن بنيات متعددة النواة، إلا أن بعض الأنظمة مصممة الآن باستخدام نوى تعمل بنفس مجموعة التعليمات، ولكنها تختلف من حيث سرعة الساعة وإدارة الطاقة، بما في ذلك القدرة على ضبط استهلاك الطاقة للنواة. تُعرف مثل هذه الأنظمة باسم تعدد المعالجات غير المتجانسة .(HMP) لاحظ أن هذا ليس شكلاً من أشكال تعدد المعالجات غير المتماثلة حيث يمكن تشغيل مهام النظام والمستخدم على أي نواة. بل إن القصد من تعدد المعالجات غير المتجانسة هو إدارة استهلاك الطاقة بشكل أفضل من خلال تعيين مهام لنوى معينة بناءً على المتطلبات المحددة للمهمة.

هناك العديد من المزايا لهذا النهج. من خلال الجمع بين عدد من النوى الأبطأ مع النوى الأسرع، يمكن لجدول وحدة المعالجة المركزية تعيين المهام التي لا تتطلب أداءً عاليًا، ولكنها قد تحتاج إلى التشغيل لفترات أطول (مثل المهام الخلفية) إلى النوى الصغيرة، وبالتالي المساعدة في الحفاظ على شحن البطارية. وبالمثل، يمكن تعيين التطبيقات التفاعلية التي تتطلب المزيد من قوة المعالجة، ولكنها قد تعمل لفترات أقصر، إلى النوى الكبيرة. بالإضافة إلى ذلك، إذا كان الجهاز المحمول في وضع توفير الطاقة، يمكن تعطيل النوى الكبيرة كثيفة الطاقة ويمكن للنظام الاعتماد فقط على النوى الصغيرة الموفرة للطاقة. يدعم Windows 10 جدولة HMP من خلال السماح لمسلك باختيار سياسة جدولة تدعم بشكل أفضل متطلبات إدارة الطاقة الخاصة به.

جدولة المعالجات متعددة النوى

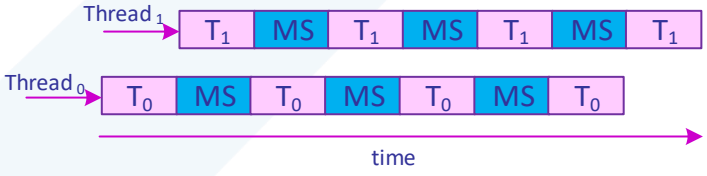
سمحت أنظمة SMP تقليديًا بتشغيل العديد من العمليات بالتوازي من خلال توفير معالجات مادية متعددة. ومع ذلك، فإن معظم أجهزة الكمبيوتر المعاصرة تضع الآن نوى حوسبة متعددة على نفس الشريحة المادية، مما يؤدي إلى معالج متعدد النواة. تحافظ كل نواة على حالتها المعمارية وبالتالي تبدو لنظام التشغيل وكأنها وحدة معالجة مركزية منطقية منفصلة. أنظمة SMP التي تستخدم معالجات متعددة النوى أسرع وتستهلك طاقة أقل من الأنظمة التي تحتوي كل وحدة معالجة مركزية فيها على شريحة مادية خاصة بها.

قد تؤدي المعالجات متعددة النوى إلى تعقيد مشكلات الجدولة. دعنا نفكر في كيفية حدوث ذلك. اكتشف الباحثون أنه عندما يصل المعالج إلى الذاكرة، فإنه يقضي قدرًا كبيرًا من الوقت في انتظار توفر البيانات. يحدث هذا الموقف، المعروف باسم توقف الذاكرة، في المقام الأول لأن المعالجات الحديثة تعمل بسرعات أسرع بكثير من الذاكرة. ومع ذلك، يمكن أن يحدث توقف الذاكرة أيضًا بسبب خطأ في ذاكرة التخزين المؤقت (الوصول إلى البيانات غير الموجودة في ذاكرة التخزين المؤقت). يوضح الشكل9 توقف الذاكرة Memory Stall. في هذا السيناريو، يمكن للمعالج أن يقضي ما يصل إلى 50 بالمائة من وقته في انتظار توفر البيانات من الذاكرة.



الشكل 9: توقف الذاكرة

لمعالجة ذلك طبقت العديد من التصميمات الحديثة للأجهزة نوى معالجة متعددة المسالك حيث يتم تعيين مسلكين (أو أكثر) للأجهزة لكل نواة. وبهذه الطريقة، إذا توقف مسلك من مسلك الأجهزة أثناء انتظار الذاكرة، يمكن للنواة التبديل إلى مسلك آخر. يوضح الشكل 10 نواة معالجة ثنائية المسالك حيث يتم تداخل تنفيذ المسلك 0 وتنفيذ المسلك 1. ومن منظور نظام التشغيل، يحتفظ كل مسلك من مسالك الأجهزة بحالته المعمارية، مثل مؤشر التعليمات ومجموعة السجلات، وبالتالي يظهر كوحدة معالجة مركزية منطقية متاحة لتشغيل مسلك برمجي. يتم توضيح هذه التقنية – المعروفة باسم تعدد مسالك الشريحة (CMT) - في الشكل 11. هنا، يحتوي المعالج على أربع نوى، حيث تحتوي كل نواة على مسلكين من مسلك الأجهزة. من منظور نظام التشغيل، يوجد ثماني وحدات معالجة مركزية منطقية.

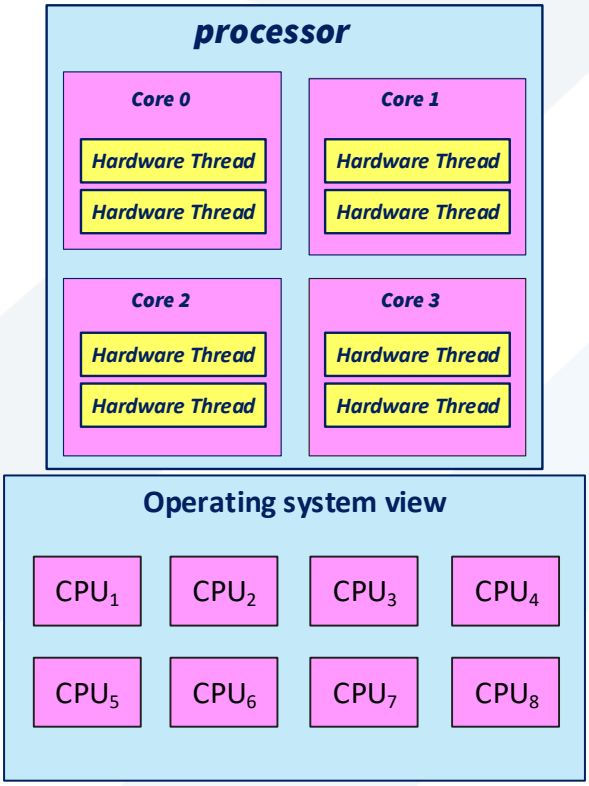


الشكل 10: تنفيذ المسالك في الأنظمة متعددة النوى

توجد طريقتان لتشغيل معالج متعدد المسالك:

التشغيل المتعدد المسالك الخشن Coarse-Grained Multithreading: في التشغيل المتعدد المسالك الخشن، يتم تنفيذ المسلك على المعالج حتى يحدث حدث تأخير طويل مثل توقف الذاكرة، وبسبب التأخير الناتج عن حدث تأخير طويل، يجب على المعالج التبديل إلى مسلك آخر لبدء التنفيذ. تكون تكلفة التبديل بين المسالك عالية حيث يجب إنهاء خط أنابيب التعليمات قبل أن يتمكن المسلك الآخر من بدء التنفيذ على نواة المعالج. بمجرد أن يبدأ هذا المسلك الجديد في التنفيذ، يبدأ في ملء خط الأنابيب بتعليماته.

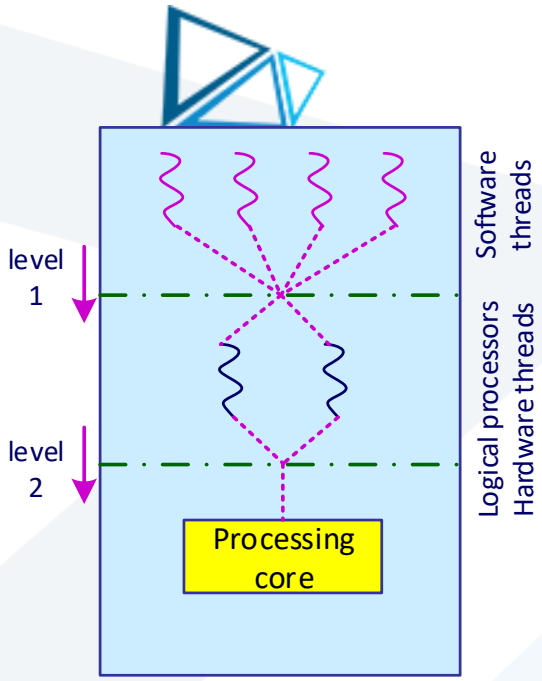
التشغيل المتعدد المسالك الدقيق Fine-Grained Multithreading: يقوم هذا التشغيل المتعدد المسالك بالتبديل بين المسالك على مستوى أدق بكثير بشكل أساسي عند حدود دورة التعليمات. يتضمن التصميم المعماري للأنظمة الدقيقة منطقًا للتبديل بين المسالك ونتيجة لذلك تكون تكلفة التبديل بين المسالك صغيرة.



الشكل 11: النوى الفيزيائية النوى الظاهرية

ومع ذلك، فإن التصميم المعماري للأنظمة الدقيقة يشمل المنطق لتبديل المسالك. ونتيجة لذلك، تكون تكلفة التبديل بين المسالك صغيرة. ومن المهم ملاحظة أن موارد النواة المادية (مثل ذاكرة التخزين المؤقت وخطوط الأنابيب) يجب أن تكون مشتركة بين مسالك الأجهزة الخاصة بها، وبالتالي لا يمكن لنواة المعالجة تنفيذ سوى مسلك فيزيائي واحد في كل مرة.

وبالتالي، فإن المعالج متعدد المسالك ومتعدد النوى يتطلب في الواقع مستويين مختلفين من الجدولة، كما هو موضح في الشكل 11، والذي يوضح نواة معالجة ثنائية المسالك. توجد على أحد المستويات قرارات الجدولة التي يجب أن يتخذها نظام التشغيل أثناء اختياره لمسلك البرنامج الذي سيتم تشغيله على كل مسلك أجهزة (وحدة المعالجة المركزية المنطقية). يحدد المستوى الثاني من الجدولة كيف يقرر لكل نواة أي مسلك أجهزة سيتم تشغيله. هناك العديد من الاستراتيجيات التي يمكن تبنيها في هذا الموقف. أحد هذه الأساليب هو استخدام خوارزمية بسيطة لجدولة مسلك الأجهزة إلى نواة المعالجة. لاحظ أن المستويين المختلفين للجدولة الموضحين في الشكل 12 ليسا بالضرورة متعارضين. في الواقع، إذا تم إعلام مجدول نظام التشغيل (المستوى الأول) بمشاركة موارد المعالج، فيمكنه اتخاذ قرارات جدولة أكثر فعالية. على سبيل المثال، بفرض أن وحدة المعالجة المركزية تحتوي على نواتين للمعالجة، وكل نواة تحتوي على مسلكين للأجهزة. إذا كان هناك مسلكان للبرامج يعملان على هذا النظام، فيمكن تشغيلهما إما على نفس النواة أو على نواة منفصلة. إذا تم جدولة كليهما للعمل على نفس النواة، فيجب أن يتشاركا في موارد المعالج وبالتالي من المرجح أن يستمرا بشكل أبطأ مما لو تم جدولتهما على نواة منفصلة. إذا كان نظام التشغيل على دراية بمستوى مشاركة موارد المعالج، فيمكنه جدولة مسالك البرامج على معالجات منطقية لا تتشارك الموارد.



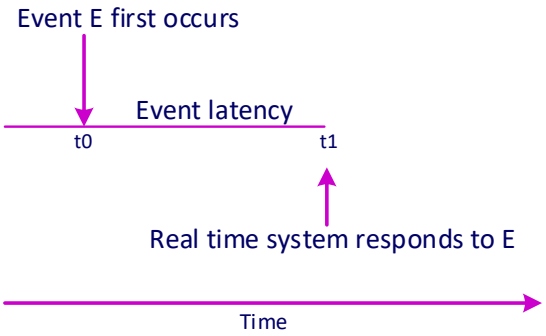
الشكل 12: مثال على الجدولة وفق مستويين

جدولة وحدة المعالجة المركزية في الوقت الحقيقي

تتضمن جدولة وحدة المعالجة المركزية لأنظمة التشغيل في الزمن الحقيقي قضايا خاصة. بشكل عام، يمكننا التمييز بين أنظمة الزمن الحقيقي الناعمة وأنظمة الزمن الحقيقي الصارمة. لا توفر أنظمة الزمن الحقيقي الناعمة أي ضمان بشأن موعد جدولة عملية الزمن الحقيقي الحرجة. فهي تضمن فقط إعطاء الأولوية للعملية على العمليات غير الحرجة. تتمتع أنظمة الزمن الحقيقي الصارمة بمتطلبات أكثر صرامة. يجب خدمة المهمة بحلول الموعد النهائي لها؛ الخدمة بعد انتهاء الموعد النهائي هي نفس عدم الخدمة على الإطلاق. في هذا القسم، نستكشف العديد من القضايا المتعلقة بجدولة العملية في أنظمة التشغيل في الزمن الحقيقي الناعمة والصارمة.

تقليل زمن الوصول Minimizing Latency

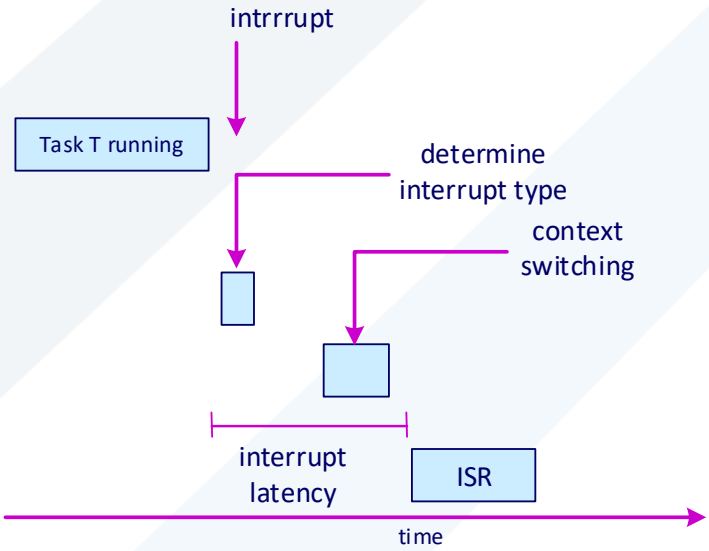
فكر في الطبيعة التي تعتمد على الأحداث لنظام الزمن الحقيقي. ينتظر النظام عادةً حصول حدث ما في الزمن الحقيقي. قد تنشأ الأحداث إما في البرنامج — كما هو الحال عند انتهاء صلاحية المؤقت— أو في الأجهزة— كما هو الحال عندما تكتشف مركبة يتم التحكم فيها عن بُعد أنها تقترب من عائق. عندما يحدث حدث ما، يجب على النظام الاستجابة له وخدمته بأسرع ما يمكن. نشير إلى زمن انتظار الحدث باعتباره مقدار الزمن الذي ينقضي من وقت حدوث الحدث إلى وقت خدمته (الشكل 13).



الشكل 13: تأخير الحدث

عادةً، تتطلب الأحداث المختلفة زمن انتقال مختلفًا. على سبيل المثال، قد يكون المطلوب زمن الانتقال لنظام المكابح المانعة للانغلاق من 3 إلى 5 مللي ثانية. أي أنه من الزمن الذي يكتشف فيه العجلة لأول مرة أنها تنزلق، يكون لدى النظام الذي يتحكم في المكابح المانعة للانغلاق من 3 إلى 5 مللي ثانية للاستجابة للموقف والتحكم فيه. قد تؤدي أي استجابة تستغرق وقتًا أطول إلى خروج السيارة عن السيطرة. على النقيض من ذلك، قد يتحمل النظام المضمن الذي يتحكم في الرادار في طائرة ركاب فترة زمن انتقال تصل إلى عدة ثوانٍ.

هناك نوعان من التأخير يؤثران على أداء أنظمة الزمن الحقيقي:



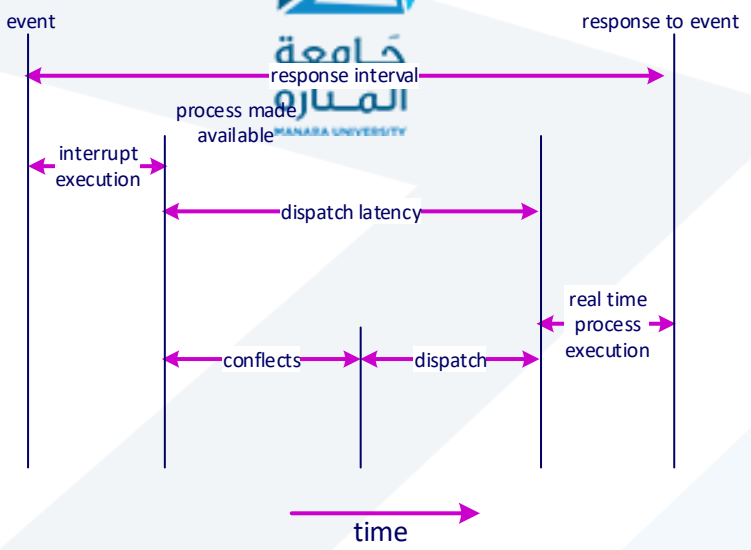
الشكل 14: تأخير المقاطعة

تأخير المقاطعة

يشير زمن انتقال المقاطعة إلى الفترة الزمنية من وصول المقاطعة إلى وحدة المعالجة المركزية إلى بداية الروتين الذي يخدم المقاطعة. عندما تحدث مقاطعة، يجب على نظام التشغيل أولاً إكمال التعليمات التي ينفذها وتحديد نوع المقاطعة التي حدثت. يجب عليه بعد ذلك حفظ حالة العملية الحالية قبل خدمة المقاطعة باستخدام روتين خدمة المقاطعة المحدد .(ISR) إن الزمن الإجمالي المطلوب لأداء هذه المهام هو زمن انتقال المقاطعة (الشكل 14).

من الواضح أنه من الأهمية بمكان بالنسبة لأنظمة التشغيل في الزمن الحقيقي أن تقل من زمن انتقال المقاطعة لضمان حصول المهام في الزمن الحقيقي على الاهتمام الفوري. في الواقع، بالنسبة لأنظمة الزمن الحقيقي الصارمة، لا ينبغي تقليل زمن انتقال المقاطعة فحسب، بل يجب تحديده لتلبية المتطلبات الصارمة لهذه الأنظمة.

إن أحد العوامل المهمة التي تساهم في تأخير المقاطعة هو مقدار الوقت الذي قد يتم فيه تعطيل المقاطعات أثناء تحديث هياكل بيانات النواة. تتطلب أنظمة التشغيل في الزمن الحقيقي تعطيل المقاطعات لفترات زمنية قصيرة جدًا فقط.



الشكل 15: تأخير الانتقال

تأخير الانتقال

إن مقدار الوقت المطلوب لموجه الجدولة لإيقاف عملية ما وبدء أخرى يُعرف باسم تأخير الانتقال. من المهم هنا تنفيذ المهام في الزمن الفعلي مع إمكانية الوصول الفوري إلى وحدة المعالجة المركزية يفرض على أنظمة التشغيل في الزمن الفعلي تقليل هذا الزمن أيضًا. إن التقنية الأكثر فعالية للحفاظ على تأخير الانتقال منخفضًا هي توفير نوى استباقية. بالنسبة لأنظمة الزمن الفعلي الصارمة، يتم قياس تأخير انتقال عادةً بعدة ميكروثانية.

يبين الشكل 15 مخططًا لتأخير الانتقال. تتكون مرحلة التعارض في المرتبطة بتأخير الانتقال من مرحلتين:

1. استباق أي عملية تعمل في النواة
2. تحرير الموارد اللازمة لعملية ذات أولوية عالية بواسطة العمليات ذات الأولوية المنخفضة

بعد مرحلة التعارض، تقوم مرحلة الانتقال بجدولة العملية ذات الأولوية العالية على وحدة معالجة مركزية متاحة.

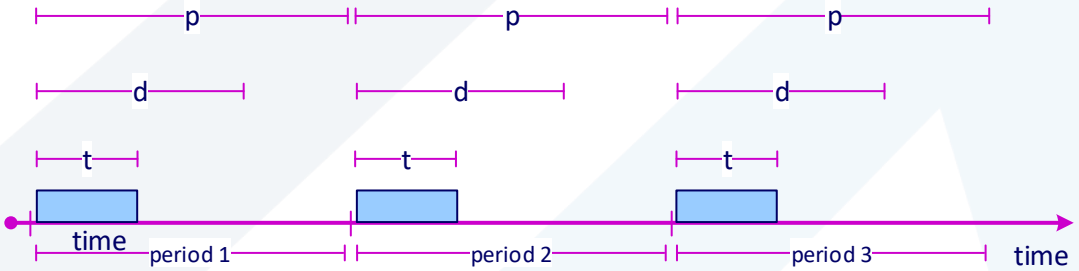
الجدولة القائمة على الأولوية Priority-Based Scheduling

إن أهم ميزة لنظام التشغيل في الوقت الفعلي هي الاستجابة فورًا لعملية في الزمن الحقيقي بمجرد أن تتطلب هذه العملية وحدة المعالجة المركزية. ونتيجة لذلك، يجب أن يدعم المجدول لنظام التشغيل في الزمن الحقيقي خوارزمية قائمة على الأولوية مع الاستباق. تذكر أن خوارزميات الجدولة القائمة على الأولوية تعين لكل عملية أولوية بناءً على أهميتها؛ يتم تعيين أولويات أعلى للمهام الأكثر أهمية من تلك التي تعتبر أقل أهمية. إذا كان المجدول يدعم الاستباق أيضًا، فسيتم استباق العملية الجارية حاليًا على وحدة المعالجة المركزية إذا أصبحت عملية ذات أولوية أعلى جاهزة للتشغيل.

إن توفير جدول زمني استباقي قائم على الأولوية يضمن فقط وظائف الوقت الحقيقي الناعمة. يجب أن تضمن أنظمة الوقت الحقيقي الصارم أيضًا أن المهام في الوقت الحقيقي سيتم خدمتها وفقًا لمتطلبات الموعد النهائي الخاصة بها، ويتطلب تقديم مثل هذه الضمانات ميزات جدولة إضافية. في بقية هذا القسم، نغطي خوارزميات الجدولة المناسبة لأنظمة الوقت الحقيقي الصارم.

قبل ذلك يجب أن نحدد خصائص معينة للعمليات التي سيتم جدولتها. أولاً، تعتبر العمليات دورية. أي أنها تتطلب وحدة المعالجة المركزية على فترات ثابتة (أدوار). بمجرد حصول عملية دورية على وحدة المعالجة المركزية، يكون لها وقت معالجة ثابت t ، وموعد نهائي d يجب أن تخدمها وحدة المعالجة المركزية بحلوله، وفترة p . يمكن التعبير عن العلاقة بين وقت المعالجة والموعد النهائي والفترة على النحو التالي: $0 \le t \le d \le p$. ومعدل المهمة الدورية هو 1/p يوضح الشكل 16 تنفيذ عملية دورية بمرور الوقت. يمكن للمجدولين الاستفادة من هذه الخصائص وتعيين الأولويات وفقًا لمتطلبات الموعد النهائي أو المعدل للعملية.

ما هو غير عادي في هذا الشكل من الجدولة هو أن العملية قد تضطر إلى الإعلان عن متطلبات الموعد النهائي الخاصة بها للمجدول. ثم باستخدام تقنية تُعرف باسم خوارزمية التحكم في القبول، يقوم المجدول بأحد أمرين. إما أن يقبل العملية، ويضمن اكتمال العملية في الوقت المحدد، أو يرفض الطلب باعتباره مستحيلًا إذا لم يتمكن من ضمان تنفيذ المهمة بحلول الموعد النهائي.



الشكل 16: تنفيذ عملية دورية

الجدولة ذات المعدل الرتيب Rate-Monotonic Scheduling

تقوم خوارزمية الجدولة ذات المعدل الرتيب بجدولة المهام الدورية باستخدام سياسة أولوية ثابتة مع الاستباق. إذا كانت هناك عملية ذات أولوية أقل قيد التشغيل وأصبحت عملية ذات أولوية أعلى متاحة للتشغيل، فستسبق العملية ذات الأولوية الأقل. عند دخول النظام، يتم تعيين أولوية لكل مهمة دورية بشكل عكسي بناءً على زمن الدور. كلما كانت الزمن أقصر، زادت الأولوية؛ وكلما طال الزمن، انخفضت الأولوية. والسبب وراء هذه السياسة هو تعيين أولوية أعلى للمهام التي تتطلب وحدة المعالجة المركزية بشكل أكثر تكرارا. علاوة على ذلك، تفترض الجدولة ذات المعدل الرتيب أن وقت معالجة العملية الدورية هو نفسه لكل دفعة وحدة معالجة مركزية. أي أنه في كل مرة تستحوذ فيها عملية على وحدة المعالجة المركزية، تكون مدة دفعة وحدة المعالجة المركزية هي نفسها.

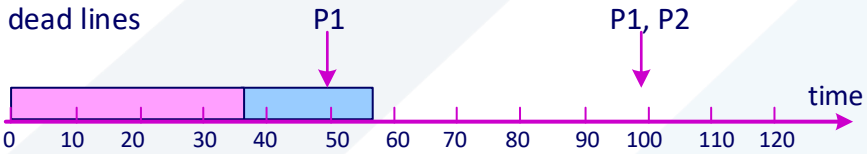
دعنا نفكر في مثال. لدينا عمليتان، P1 و P2. الأدوار الزمنية لـ P1 و P2 هي 50 و100 على التوالي— أي أن p1 = 50 و p2 = 100. أوقات المعالجة هي t1 = 20 لـ P1 و t2 = 35 لـ P2. يتطلب الموعد النهائي لكل عملية أن تكمل دفعة وحدة المعالجة المركزية بحلول بداية دورها التالي.

يجب أن نسأل أنفسنا أولاً ما إذا كان من الممكن جدولة هذه المهام بحيث تلبي كل منها مواعيدها النهائية. إذا قمنا بقياس استخدام وحدة المعالجة المركزية لعملية Pi كنسبة دفعتها الزمنية إلى دورها—ti/pi—فإن استخدام وحدة المعالجة المركزية لـ P1 هو 20/50=40% واستخدام وحدة المعالجة المركزية لـ P2 هو 35/100=35%، وذلك لإجمالي استخدام وحدة المعالجة المركزية بنسبة 75 بالمائة. لذلك، يبدو أنه يمكننا جدولة هذه المهام بطريقة تجعل كلاً منهما يلبي مواعيده النهائية ويترك وحدة المعالجة المركزية مع دورات متاحة.

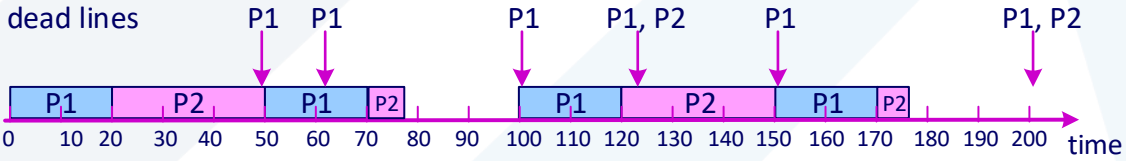
لنفترض أننا نعطي P2 أولوية أعلى من P1 يظهر تنفيذ كل من P1 و P2 في هذا الموقف في الشكل 17. وكما نرى، تبدأ P2

التنفيذ أولاً وتكتمل في الوقت 35. وفي هذه النقطة، تبدأP1 ؛ وتكمل دفعة وحدة المعالجة المركزية في الوقت 55. ومع ذلك، كان الموعد النهائي الأول لـ P1 في الوقت 50، لذا فقد تسبب المجدول في عدم وصول P1 إلى موعدها النهائي.

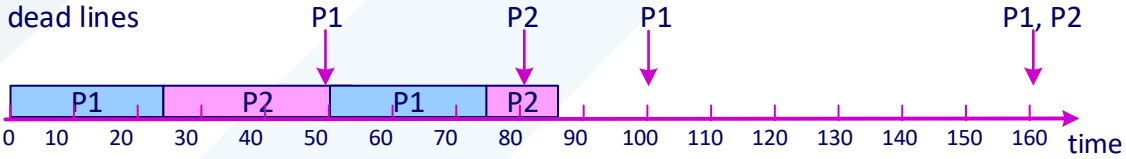
لنفترض الآن أننا نستخدم الجدولة ذات المعدل الرتيب، حيث تخصص لـ P1 أولوية أعلى من P2 لأن دور P1 أقصر من دور P2. يظهر تنفيذ هذه العمليات في هذا الموقف في الشكل18 تبدأ P1 أولاً وتكمل دفعة وحدة المعالجة المركزية في الوقت 20، وبالتالي تفي بموعدها النهائي الأول. تبدأ P2 في العمل في هذه النقطة وتعمل حتى الوقت 50. في هذا الوقت، تسبقها P1، على الرغم من أنها لا تزال لديها 5 مللي ثانية متبقية في دفعة وحدة المعالجة المركزية. تكمل P1 دفعة وحدة المعالجة المركزية في الوقت 70، وفي هذه النقطة يستأنف المجدول P2. يكمل P2 دفعة وحدة المعالجة المركزية في الوقت 75، وفي أيضًا بموعده النهائي الأول. يظل النظام خاملاً حتى الوقت 100، عندما يتم جدولة P1 مرة أخرى.



الشكل 17: الجدولة وفق الأولوية البحتة حيث أولوية P2 أعلى من أولوية P1



الشكل 18: الجدولة ذات المعدل الرتيب



الشكل 19: عدم القدرة على مراعاة زمن الانتهاء في خوارزمية الجدولة ذات المعدل الرتيب

تعتبر الجدولة ذات المعدل الرتيب مثالية بحيث إذا تعذر جدولة مجموعة من العمليات بواسطة هذه الخوارزمية، فلا يمكن جدولتها بواسطة أي خوارزمية أخرى تعين أولويات ثابتة. دعنا نفحص بعد ذلك مجموعة من العمليات التي لا يمكن جدولتها باستخدام خوارزمية الرتيبة المعدلة.

افترض أن العملية P1 لها دور p1 = 50 وفترة وحدة المعالجة المركزية t1 = 25. بالنسبة إلىP2 ، تكون القيم المقابلة هي p2 = 80 وt2 = 35. ستقوم الجدولة الرتيبة المعدلة بتعيين أولوية أعلى للعمليةP1 ، حيث أن لها فترة أقصر. يبلغ إجمالي استخدام وحدة المعالجة المركزية للعمليتين (20/50) + (35/80/) = 0.94، وبالتالي يبدو من المنطقي أن يتم جدولة العمليتين مع ترك وحدة المعالجة المركزية مع 6 بالمائة من الوقت المتاح. يوضح الشكل 19. جدولة العمليتين P1 وP2. في البداية، تعمل العملية P1 حتى تكمل دفعة وحدة المعالجة المركزية في الوقت 25. ثم تبدأ العملية P2 في العمل وتستمر حتى الوقت 50، عندما تسبقها العملية P1. في هذه المرحلة، لا يزال لدى العملية P2 10 مللي ثانية متبقية في دفعة وحدة المعالجة المركزية. تعمل العملية P1 حتى الوقت 75؛ وبالتالي، تنهي العملية P2 دفعتها في الوقت 85، بعد الموعد النهائي لاستكمال دفعة وحدة المعالجة المركزية في الوقت 80.

على الرغم من كونها مثالية، إلا أن الجدولة ذات المعدل الرتيب لها حد: استخدام وحدة المعالجة المركزية محدود، وليس من الممكن دائمًا تعظيم موارد وحدة المعالجة المركزية بالكامل. أسوأ حالة لاستخدام وحدة المعالجة المركزية لجدولة N عملية هي:

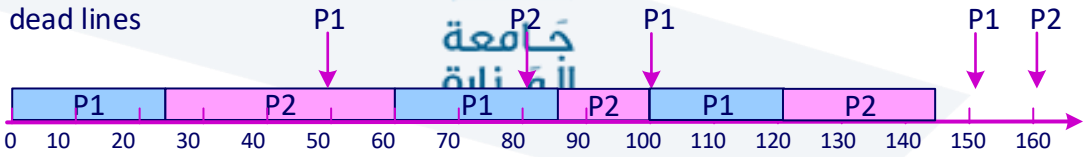
$$N(2^{1/N} - 1).$$

مع وجود عملية واحدة في النظام، يكون استخدام وحدة المعالجة المركزية 100 بالمائة، لكنه ينخفض إلى حوالي 69 بالمائة مع اقتراب عدد العمليات من اللانهاية. مع وجود عمليتين، يكون استخدام وحدة المعالجة المركزية محدودًا بحوالي 83 بالمائة. إن الاستخدام المشترك لوحدة المعالجة المركزية للعمليتين المجدولين في الشكل 17 والشكل 18 هو 75 بالمائة؛ وبالتالي، فإن خوارزمية الجدولة ذات المعدل الرتيب تضمن جدولتهما بحيث يمكنهما تلبية مواعيدهما النهائية. وبالنسبة للعمليتين المجدولين في الشكل 19، فإن الاستخدام المشترك لوحدة المعالجة المركزية هو حوالي 94 بالمائة؛ وبالتالي، فإن الجدولة الرتيبة المعدلة لا يمكنها ضمان إمكانية جدولتهما بحيث تلبي مواعيدهما النهائية.

الجدولة وفق أقرب موعد نهائي أولا Earliest-Deadline-First Scheduling

تحدد الجدولة وفق أقرب موعد نهائي أولاً الأولويات بشكل ديناميكي وفقًا للموعد النهائي. فكلما كان الموعد النهائي مبكرًا، زادت الأولوية؛ وكلما كان الموعد النهائي متأخرًا، انخفضت الأولوية. وفقًا لسياسةEDF ، عندما تصبح العملية قابلة للتشغيل، يجب عليها الإعلان عن متطلبات الموعد النهائي للنظام. قد يتعين تعديل الأولويات لتعكس الموعد النهائي للعملية القابلة للتشغيل حديثًا. لاحظ كيف يختلف هذا عن الجدولة ذات المعدل الرتيب، حيث تكون الأولويات ثابتة.

لتوضيح جدولة EDF، نقوم مرة أخرى بجدولة العمليات الموضحة في الشكل 19، والتي فشلت في تلبية متطلبات الموعد النهائي بموجب الجدولة ذات المعدل الرتيب. تذكر أن P1 لها قيم p1 = 50 وt1 = 25 وأن P2 لها قيم p2 = 80 و .t2 = 35 يظهر جدولة EDF لهذه العمليات في الشكل 20. تتمتع العملية P1 بأسبق موعد نهائي، لذا فإن أولويتها أعلى من أولوية العملية P2. تبدأ العملية P2 في العمل عند نهاية دفعة وحدة المعالجة المركزية للعملية P1. ومع ذلك، بينما تسمح الجدولة ذات المعدل الرتيب للعملية P1 بسبق العملية P2 في بداية فترتها التالية في الوقت 50، تسمح جدولة EDF للعملية P2 بمواصلة العمل. تتمتع العملية P2 الآن بأولوية أعلى من العملية P1 لأن موعدها النهائي التالي (في الوقت 80) يسبق موعد العملية P1 (في الوقت 100). وبالتالي، تلبي كل من العملية P1 والعملية P2 مواعيدهما النهائية الأولى. تبدأ العملية P1 مرة أخرى في العمل في الوقت 60 وتكمل دفعة وحدة المعالجة المركزية الثاني في الوقت 85، وتلبي أيضًا موعدها النهائي الثاني في الوقت 100. تبدأ العملية P2 في العمل عند هذه النقطة، فقط لتسبقها العملية P1 في بداية فترتها التالية في الوقت 100. يتم استباق العملية P2 لأن العملية P1 لها موعد نهائي أسبق (الوقت 150) من العملية P2 (الوقت 160). في الوقت 125، تكمل P1 دفعة وحدة المعالجة المركزية الخاصة بها وتستأنف P2 التنفيذ، وتنتهي في الوقت 145 وتفي بموعدها النهائي أيضًا. يظل النظام خاملاً حتى الوقت 150، عندما يتم جدولة P1 للعمل مرة أخرى. وعلى عكس خوارزمية المعدل الرتيب، لا تتطلب جدولة EDF أن تكون العمليات دورية، ولا يجب أن تتطلب العملية مقدارًا ثابتًا من وقت وحدة المعالجة المركزية لكل دفعة. المتطلب الوحيد هو أن تعلن العملية عن موعدها النهائي للجدول عندما تصبح قابلة للتشغيل. تكمن جاذبية جدولة EDF في أنها مثالية من الناحية النظرية – من الناحية النظرية، يمكنها جدولة العمليات بحيث يمكن لكل عملية تلبية متطلبات الموعد النهائي الخاص بها وسيكون استخدام وحدة المعالجة المركزية 100 بالمائة. ومع ذلك، من الناحية العملية، من المستحيل تحقيق هذا المستوى من استخدام وحدة المعالجة المركزية بسبب تكلفة تبديل السياق بين العمليات ومعالجة المقاطعات.



الشكل 19: الجدولة وفقا لـ EDF

جدولة المشاركة النسبية *Proportional Share Scheduling*

تعمل جداول المشاركة النسبية عن طريق تخصيص حصص T بين جميع التطبيقات. يمكن للتطبيق تلقي N حصة من الوقت، وبالتالي ضمان حصول التطبيق على N/T من إجمالي وقت المعالجة. على سبيل المثال، افترض أن إجمالي T = 100 حصة سيتم تقسيمها بين ثلاث عمليات، Aو Bو C. يتم تخصيص 50 حصة لـA ، ويتم تخصيص 15 حصة لـB ، ويتم تخصيص 20 حصة لـ C. يضمن هذا المخطط أن يكون لدى A 50 بالمائة من إجمالي وقت المعالجة، وأن يكون لدى B 15 بالمائة، وأن يكون لدى C 20 بالمائة.

يجب أن تعمل جداول الحصص النسبية جنبًا إلى جنب مع سياسة التحكم في القبول لضمان حصول التطبيق على حصصه المخصصة من الوقت. ستقبل سياسة التحكم في القبول العميل الذي يطلب عددًا معينًا من الحصص فقط إذا كانت الحصص الكافية متاحة. في مثالنا الحالي، قمنا بتخصيص 50 + 15 + 20 = 85 حصة من إجمالي 100 حصة. إذا طلبت عملية جديدة D 30 سهمًا، فسوف يرفض مراقب القبول دخول D إلى النظام.