نظم الزمن الحقيقي

الدكتور أحمد وبي
نظم الزمن الحقيقي

الدكتور أحمد وبي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

هذا الكتاب منشور تحت رخصة المشاع المبدع – النسب للمؤلف – حظر الاشتقاق (CC– BY– ND 4.0)

https://creativecommons.org/licenses/by-nd/4.0/legalcode.ar

يحق للمستخدم بموجب هذه الرخصة نسخ هذا الكتاب ومشاركته وإعادة نشره أو توزيعه بآية صيغة وآية وسيلة للنشر ولاية غاية تجارية أو غير تجارية، وذلك شريطة عدم التعديل على الكتاب وعدم الاشتقاق منه وعلى أن ينسب للمؤلف الأصلي علىشكل الآتي حصراً:

أحمد وبي، نظم الزمن الحقيقي، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018

https://pedia.svouonline.org/

Real Time Systems

Ahmad Wabbi

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

Published under the license:

Creative Commons Attributions- NoDerivatives 4.0 International (CC-BY-ND 4.0)

https://creativecommons.org/licenses/by-nd/4.0/legalcode

Available for download at: https://pedia.svouonline.org/
الفهرس

الفصل الأول: مفاهيم نظام الزمن الحقيقي
1- المصطلحات الأساسية في نظم الزمن الحقيقي System
2- Real-time Systems
3- determinism
4- CPU utilization
5- عوامل تتعلق بتصميم نظام الزمن الحقيقي
6- أمثلة على نظام الزمن الحقيقي.
7- بعض المعتقدات الخاطئة المتعلقة بنظم الزمن الحقيقي.
8- لمحة تاريخية مختصرة
9- التطورات النظرية
10- بعض نظام الزمن الحقيقي المبكرة.
11- برامج الزمن الحقيقي المبكرة.
12- نظام تشغيل الزمن الحقيقي التجارية.

الفصل الثاني: تشغيل الزمن الحقيقي
1- نواة نظام تشغيل الزمن الحقيقي.
2- pseudo-kernels
3- polling loop
4- Synchronized polling loop
5- Cyclic Executive
6- State-driven coding
7- Co-routines
8- الروتينات التعاونية
Interrupt-driven systems

Interrupt Service Routines (ISR)

context switch

preemptive-priority

hybrid systems

foreground/background

Task Control Block (TCB)

Task Control Block (TCB)

initialization

theenhine

context switch

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

Task Control Block (TCB)

foreground/background

round-robin

round-robin

cyclic executive

cyclic executive

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate

Rate
الفصل الثالث: تصميم برامجيّات نظام الزمن الحقيقي

1- هندسة الاحتياجات
2- أنواع الاحتياجات
3- توصيف احتياجات نظام الزمن الحقيقي
4- الطرق الصورية لتوصيف الاحتياجات البرمجية
5- Finite State Machine (FSM)
6- شبكات بتيري
7- آلة الحالات المنتهية
8- مقدمة

الفصل الرابع: لغات برمجة نظام الزمن الحقيقي

1- لغة التجميع
2- Assembly
<table>
<thead>
<tr>
<th>Page</th>
<th>Arabic</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>للغات الإجرائية</td>
<td>procedural languages</td>
</tr>
<tr>
<td>65</td>
<td>للغات غرضية التوجه</td>
<td>object-oriented languages</td>
</tr>
<tr>
<td>67</td>
<td>لمحة عامة عن لغات برمجة الزمن الحقيقي</td>
<td>General overview of real-time languages</td>
</tr>
<tr>
<td>67</td>
<td>- لغة ADA95</td>
<td>- ADA95 language</td>
</tr>
<tr>
<td>67</td>
<td>- لغة C</td>
<td>- C language</td>
</tr>
<tr>
<td>68</td>
<td>- لغة C++</td>
<td>- C++ language</td>
</tr>
<tr>
<td>68</td>
<td>- لغة C#</td>
<td>- C# language</td>
</tr>
<tr>
<td>69</td>
<td>- لغة Fortran</td>
<td>- Fortran language</td>
</tr>
<tr>
<td>69</td>
<td>- لغة Java</td>
<td>- Java language</td>
</tr>
<tr>
<td>70</td>
<td>- لغة real-time java</td>
<td>- real-time java language</td>
</tr>
<tr>
<td>71</td>
<td>- تحقيق لغة Real-time Java</td>
<td>- Real-time Java language</td>
</tr>
<tr>
<td>71</td>
<td>- لغة Occam 2</td>
<td>- Occam 2 language</td>
</tr>
</tbody>
</table>
| 71   | - لغات الزمن الحقيقي المخصصة | - Real-time languages dedicated
الفصل الأول: مفاهيم نظم الزمن الحقيقي

سنقدم في هذا الفصل المفاهيم الأساسية والمصطلحات اللازمة لفهم ماهية نظم الزمن الحقيقي. سنقدم كذلك بعض الأمثلة الواقعية عنها ولمحة تاريخية عن تطورها.

1- المصطلحات الأساسية في نظم الزمن الحقيقي

System

1-1- مفهوم النظام

يمكن تعريف النظام على أنه تقابل بين مجموعة مداخل inputs ومخارج outputs. يمكن نمذجة أي كينونة في الطبيعة على شكل نظام عندما لا تكون التفاصيل الداخلية للنظام مهمة، يمكن التعبير عنه على شكل صندوق أسود له عدة مداخل وعدة مخارج كما يلي:

```
+----------------+
|     نظام      |
|  l₁  l₂  ...  lₙ |
|       |       |
|      ⋮      ⋮      ⋮
<p>| | |
|       |       |</p>
<table>
<thead>
<tr>
<th>o₁  o₂  ...  oₘ</th>
</tr>
</thead>
</table>
```

النظام الحاسوبي computer systems هو نظم مكونة من برامج وعتاد مادي hardware. يشتمل العتاد المادي جميع الأجهزة الإلكترونية والكهرمائية والميكانيكية في الحاسب كالمعالج والذاكرة الرئيسية وأجهزة الدخول/خرج. أما البرامج فهي مجموعه تعليمات ينفذها العتاد المادي (المعالج) لحل مسألة ما، وتقسم إلى نوعين: برامج النظام system software والتطبيقات البرمجية software applications.

تغلب برامج النظام العتاد المادي وتوفر للتطبيقات البرمجية واجهة تخطيط مجردة تمكنها من الوصول إلى خدمات العتاد المادي بسهولة. غالباً ما تجتمع برامج النظام الأساسية مثل مجدول scheduler والذاكرة memory ومبادئ الأجهزة device drivers في كتلة واحدة هي نظام التشغيل operating system.

أما التطبيقات البرمجية فهي مصممة لحل مشاكل محددة تهم المستخدم مثل حسابات الرواتب وإدارة المستودعات وتصفح الويب والألعاب وغيرها.

تتم الاستخدام في النظم الحاسوبية أقنية المعلومات الرقمية الواضحة من الأجهزة الحاسوبية كالكاميرا والميكروفونات sensors والبرمجيات الأخرى. وهي تأتي غالباً من حسابات وكشافات وغيرها.
وغيرها من الأجهزة التي تولد معطيات رقمية أو تمثيلية digital أو تمثيلية analog. أما مخرجات النظام الحاسوبي فهي على الأغلب رقمية وتتوافق إلى تمثيلية لإرسالها إلى أجهزة الخرج المختلفة كالشاشات والساعات وغيرها كما هو مبين في الشكل التالي.

بالنظر إلى النظام السابق، نجد أن هناك تأخر زمني بين لحظة ورود المداخل (وتسمى أيضاً المحرضات stimulus) وحالة ظهور المخرجات الموافقة لها (وتسمى أيضاً الإجابة response).

تعريف:
يدعى الزمن الفاصل بين لحظة ورود مداخل النظام وحالة ظهور قيم الخرج على مخارجه زمن استجابة النظام.

1- Real-time Systems

تعريف:
نظام الزمن الحقيقي هو نظام يجب أن يحقق قيود صريحة (حدود محددة) على زمن الاستجابة. يمكن أن يؤدي عدم احترام هذه القيود لنتائج خطيرة، أو حتى فشل النظام بالكامل.

ماذا يعني "فشل النظام"؟ في بعض الحالات الخطيرة مثل فشل مكوك فضاء أو فاعل نووي، يظهر فشل النظام واضحًا للعين، ينتج عنه كارثة مؤلمة. لكن في حالات أخرى مثل آلات الصرف، لا يظهر مفهوم الفشل بوضوح.

الإثارة: Automatic teller machines

تعريف:
هو عدم تمكن النظام من أداء مهامه حسب المواصفات المحددة له.

الفشل: failure

الإثارة: 
هو نظام لا يمكنه أن يحقق أحد أو بعض المتطلبات المذكورة في failed system دفتر شروطه.
توجد تعريف أخرى لنظام الزمن الحقيقي يتفق كلها بضرورة تحقيق النظام لقيود زمنية لكي يكون صحيحاً. على سبيل المثال، يمكن تعريف نظام الزمن الحقيقي كما يلي:

**تعريف:**

نظام الزمن الحقيقي هو نظام تعتمد صحته منطقياً على من صحة مخارجه بالإضافة لاحترامها لقيودها الزمنية.

يمكن تقسيم نظام الزمن الحقيقي إلى نظم استجابة لأحداث Reactive ونظام مصممة Embedded. تستجيب النظم من النوع الأول لأحداث آتية من البيئة المحيطة بها، مثل نظام مراقبة الحريق الذي تستجيب لارتفاع درجة الحرارة بتشغيل أجهزة الإنذار والإطفاء. أما النظام المصممة فهي نظم حساسية ضرورية يحتوي في أنظمة أخرى ليست حساسية ضرورية على سبيل المثال، يوجد في معظم السيارات الحديثة نظام حساسية مصممة لتحكم بقفز الالكهال والكوابح وأكياس الهواء وغيرها. كذلك تحوي العديد من الأجهزة المنزلية الحديثة على نظم مصممة مثل أجهزة التلفاز والغسالات الآلية وأجهزة الستيريو.

من البديهي أيضاً أن النظم المعقدة كالأطارات وسفن الفضاء والآلات الصناعية تحوي كلها على نظم مضمنة. ينطبق شرط الزمن الحقيقي خصوصاً على هذه النظم. في الطائرات مثلًا، يجب معالجة المعطيات الآتية من مقياس التسارع بدور محدد يعتمد على مواصفات الطائرة (كل 10 ميلي ثانية على سبيل المثال). وإذا فشل النظام في ذلك، فسيحصل على معلومات خاطئة عن موقع وسرعة الطائرة، مما يمكن أن يؤدي إلى انحرافها عن مسارها في أحسن الأحوال، أو تلطيخها في أسوأ الأحوال. أما في المفاعلات النووية، فتدور عدم الاستجابة السريعة للمشكلات الحرارية إلى تعطلها أو انفجارها في أسوأ الأحوال. أخيراً، يجب على نظام حزز الديكور في شركة طيران الاستجابة لطلبات الزبائن خلال زمن معقول من وجهة نظر الزبائن، وإلا اعتبار نظاماً غير عملي. باختصار، ليس من الضروري أن يجيب نظام الزمن الحقيقي خلال زمن قصير، يجب فقط أن يكون له قيود زمنية محددة (أجزاء من الثانية أو عدة ساعات) يستجيب خلالها.

**تعريف:**

نظام الزمن الحقيقي الرخو هو نظام تنقص أهمية الأعمال التي soft real-time system يؤديها إذا تجاوزت حدوده الزمنية، لكن لا تصبح عديمة القيمة تماماً.

**تعريف:**

نظام الزمن الحقيقي الصعب هو نظام يؤدي تعديه لحدوده الزمنية hard real-time system لنتائج كارثية وتوقف كامل للنظام، حتى ولو حدث ذلك لمرة واحدة.
تعريف:

نظام الزمن الحقيقي القاسي هو نظام لا يؤدي تعديه لحدوده الزمنية لنتائج كارثية إذا حدث ذلك لعدد قليل من المرات. لكن إذا حصل ذلك لعدد أكبر من المرات، سيؤدي لنتائج كارثية وتعطل كامل للنظام.

في الحقيقة، يجب أن يُصنف أي نظام عمله وفق أحد الأصناف الثلاث السابقة. بين الجدول التالي أمثلة عن نظم زمني حقيقي رخوة وصعبة وقاسية.

<table>
<thead>
<tr>
<th>الشرح</th>
<th>تصنيف نظام الزمن الحقيقي</th>
<th>النظام</th>
</tr>
</thead>
<tbody>
<tr>
<td>تعدي النظام لحدوده الزمنية لا يؤدي لنتائج كارثية بل يؤدي فقط لأداء متناقص للنظام</td>
<td>رخو</td>
<td>ATM</td>
</tr>
<tr>
<td>تعدي النظام لحدود القيادة الزمنية يجعل الآلة تخرج عن السيطرة وتخرب المحاصيل المفيدة بدلاً من الأعشاب الضارة</td>
<td>قاسي</td>
<td>نظام التحكم المضمن بالالة مؤمنة لإزالة الأعشاب الضارة</td>
</tr>
<tr>
<td>يمكن أن يؤدي عدم إطلاق الصاروخ ضمن الحد الزمني المحدد بعد الضغط على الزر إلى عدم إصابة الهدف وحود نتائج كارثية</td>
<td>صعب</td>
<td>نظام التحكم بأسلحة طائرة حيث يؤدي الضغط على الزر لإطلاق صاروخ جو-جو</td>
</tr>
</tbody>
</table>

تجدر الإشارة إلى أن الحدود الفاصلة بين الأنواع الثلاث السابقة ليست قاطعة تماماً. ويمكن لنظام أن ينتقل من تصنيف لأخر في ظروف خاصة. على سبيل المثال، إذا تعدي جهاز الصراف الآلي حدوده الزمنية لعدد كبير من المرات، سيؤدي ذلك إلى درجة عالية من عدم الرضا لدى الكثير من زبائنه المصرف. وبالتالي تتخفف من أرباح المصرف لخسائره مما يمكن أن يؤدي إلى إفلاسه بالكامل في أسوأ الأحوال. هذا يدل على أنه يمكن أن يتغير تصنيف نظام زمني حقيقي حسب الظروف الموضوع فيها.

hard real-time

سيدور معظم نقاشنا في هذا الكتاب حول نظم الزمن الحقيقي الصعبة. لذلك، سنستعمل المصطلح "نظام زمن حقيقي" ليدل على نظام زمني حقيقي صعب ومضمن ما لم تحدد غير ذلك.

deadlines

من المهم عند دراسة نظم الزمن الحقيقي أن نناقش ماهية الزمن لأن حدودها الزمنية هي عملياً لحظات على محور الزمن. السؤال الذي يطرح نفسه هو: "من أين تأتي الحدود الزمنية وما الذي يحدد قيمها؟" يتطرق الجواب بالظواهر الفيزيائية التي يعتمد عليها نظام التحكم المدروس. على سبيل المثال، يجب تحديث الصورة 30 مرة في الثانية في نظام العرض (كالشاشات).
وأجهزة الإسقاط) للحصول على صورة متحركة ذات جودة مقبولة لأن العين الإنسانية في هذه الحالة لا تحصل بإتقان الحركة بسبب دوام الانطباع الشبكي. إذا لم يحقق جهاز العرض هذه السرعة، تصبح حركة الصورة مقطعة. الظاهرة البصرية التي يعتمد عليها النظام هنا هي العين الإنسانية. في نظام الملاحة المختلفة، يجب حساب تسارع الجسم المدروس (طائرة، سيارة، صاروخ، ...) بناءً على السرعة القصوى للجسم. لهذا فإن معدل حساب تسارع السيارة أقل من معدل حساب سرعة الصاروخ لأن السرعة القصوى للسيارة أقل بكثير من السرعة القصوى للصاروخ.

تستعمل نظام الزمن الحقيقي في بعض الحالات تحت قيد زمنية فرضتها متطلبات النظام. إن الأسلوب الطبيعي لتنفيذ التعليمات في أي نظام برمجي هو التنفيذ التسلسلي. في التشغيل التسلسلي للبرنامج، يتحوَّل هذا البرنامج سجلًا خاصًا يسمى "سجل عداد البرنامج". يحتوي "سجل عداد البرنامج" على رقمي بثرة على عنوان التعليمات المعروضة في "سجل عداد البرنامج". إذا لم يحقق جهاز العرض هذه القدرة خفيفة، فإن هذه القيود أقوى بكثير من الحاجة. لذلك فإنه من الضروري قبَّل تصميم أي نظام زمني حقيقية دراسة القيود الزمنية التي تفرضها متطلبات النظام المهمة ومحاولة تخفيفها قدر الإمكان بما يناسب البيئة الفيزيائية التي يتحكم النظام بها.

الحداثات أو العوامل 1-3 events والاحتمالية

إن الأسلوب الطبيعي لتنفيذ التعليمات في أي نظام برمجي Program هو التنفيذ التسلسلي. يحتوي المعالج الذي ينفذ هذا البرنامج سجلًا خاصًا يسمى "سجل عداد البرنامج". يحتوي "سجل عداد البرنامج" على رقمي بثرة على عنوان التعليمات المعروضة في "سجل عداد البرنامج". عندما يكون تنفيذ البرنامج سلسلةً تجري زياده قيمة هذا السجل بمقدار معدل للحصول على عنوان التعليمة التالية بعد الانتهاء من تنفيذ التعليمات الحالية. لكن هناك بعض الحالات التي تغير هذا التنفيذ التسلسلي وتجعل المعالج يقفز لمكان آخر في البرنامج لتنفيذ التعليمات الموجودة فيه. المثال التقليدي على مثل هذه الحالات هو تنفيذ تعليمة if-then-goto.

من الأمثلة الأخرى على الحالات التي تغير من تنفيذ التسلسلي التنفيذ: استدعاء التوابع في اللغات البرمجية مثل "C، باسكال" وإنشاء الأغراض objects واستدعاء طرائقها methods في "Java، C++ و Object Oriented Languages" اللغات الفرعية التوجه في المظهر على شكل معيّن في مخطط الانتهاء التالي:

تعريف

أي شيء يحدث ويتسبب في عداد البرنامج تغير لأسلوب غير تنفيذ "event" برنامجة (وهو ما يعتبر "حدثًا"

تعتبر Tasks or Jobs عادة ما تكون نظام الزمن الحقيقي من عدد من الإجراءات أو غير دورية Periodic وكل منها يؤدي عملاً محددًا. يمكن أن تكون هذه الإجراءات دورية أو غير دورية
تحدث استجابة لحدث ما (تحدث إلا إجابة لحدث ما). على سبيل المثال، الإجراء الذي يقوم بالحفاظ على سرعة الطائرة في الجو هو إجراي دوري يقوم دورياً بقراءة سرعة الطائرة الحالية ويحسب الفرق بينها وبين السرعة المطلوبة لحفاظ الطائرة عليها ويرسل على أساسها أوامر التحكم لمحركات الطائرة. أما في نظام مراقبة الحريق فيؤدي ارتفاع درجة الحرارة لإقلاع إجراء إطفاء الحريق، وهو إجراء غير دوري، ويُسلم على أساسها أوامر التحكم لمحركات الطائرة.

تعريف

الإجراي هو اللحظة التي يصبح فيها هذا الإجراي جاهزاً للتنفيذ (Release time) لحظة الإطلاق (إطلاق الإشارة في النظام الحاسوبي) من الأحداث المتزامنة synchronous، والأحداث الغير متزامنة asynchronous.

هناك نوعان من الأحداث: الأحداث المتزامنة synchronous والأحداث الغير متزامنة asynchronous. تحدث الأحداث المتزامنة في لحظات يمكن التنبؤ بها في تدفق تنفيذ البرنامج، مثل تعليمة القفز الشرطي المعينة (المعينة) المبينة في مخطط التدفق أعلاه. رغم أن القفز المشروط لا يتحقق دائمًا، إلا أنه يمكن التنبؤ به. أما الأحداث الغير متزامنة، فتحدث في لحظات لا يمكن التنبؤ بها طوال فترة تنفيذ البرنامج وتسببها مصادف من خارج النظام.

يتضح الإشارة إلى أننا نعتبر المؤقت الذي يولد نبضات منتظمة (كل 5 ميلي ثانية على سبيل المثال) حدثًا غير متزامن. فعلى الرغم من أن المؤقت متزامن، وهو أمر مستحيل عمليًا، فإن اللحظات التي يولد فيها برنامج تدفق تنفيذ البرنامج تخضع لعوامل عديدة لا يمكن التنبؤ بها، مثل اللحظة التي بدأ فيها المؤقت بالعمل نسبًا لبداية تنفيذ البرنامج وزمان تأخير انتشار الإشارة في النظام الحاسوبي نفسه وغيرها من العوامل. لذلك لا يمكن التنبؤ لحظات المقاطعات تلك.

ذكرنا سابقاً أن الأحداث التي لا تحدث بدور منتظم تسمى أحداثًا غير دورية aperiodic. تسمى الأحداث الغير دورية والتي قد تحدث بالأحداث المتزامنة synchronous لتصنيف الأحداث وبعض الأمثلة عليها.

<table>
<thead>
<tr>
<th>دورية</th>
<th>غير دورية</th>
<th>متزامنة</th>
</tr>
</thead>
<tbody>
<tr>
<td>periodic</td>
<td>aperiodic</td>
<td>synchronous</td>
</tr>
</tbody>
</table>

- تعليمات برنامج في حلقة، إجراءات مجودة باستعمال تعلية فقر تقليدية، مجمع النفايات في اللغات الحديثة.
- تعليمات تقليدية، تعليمات في إكمال الاستثناءات (أخطاء زمن التنفيذ).
exception or traps

---

 Möqt مولدة

 مقاطعات منتظمة

 مقاطعات منتظمة ولكن ليس لها دور ثابت

 استثناءات مولدة

 خارجياً، أحداثاً عشوائية

---

 From the image, it appears to be a page discussing the concepts of garbage collection, exception or traps, and asynchronous events. The text seems to be discussing the challenges in system control and the need for deterministic and temporal determinism. The text is in Arabic and English, and it appears to be discussing the importance of controlling systems and the need to predict all possible conditions for correct functioning. The page number is also visible.
انشغالية المعالج (CPU utilization)

تُعرّفان الشكل مصطلحاً هاماً يستعمل لقياس أداء نظام الزمن الحقيقي. يقوم المعالج باستمرار بجلب التعليمات من الذاكرة وفك ترميزها وتنفيذها طالما بقي مغذٍ بالنظام الكهربائي. إما أن تكون هذه التعليمات هي من تعليمات برنامج التحكم المفيدة، أو تعليمات أقل أهمية ليس لها علاقة بالحدود الزمنية للنظام (مثل تعليمات النشاط المستخدم على سبيل المثال)، أو تعليمات (وهي تعليمة NO-Operation). تعليمة آلة لا يؤدي تنفيذها لأي نتائج وهي اختصار لـ (No-Operation).

يُقابِل قياس الزمن الذي يقضيه المعالج في وضع الخمول (Idle) وذلك لأنه لا يوجد NOP (وهو عندما ينفذ المعالج تعليمات فقط لأنه لا يوجد) باليوم المعالج في الزمن الحقيقي الذي يتطلبه النظام.

تعريف

الانشغالية المعالج (U) أو معامل تحميله الزمني CPU utilization هي النسبة الزمنية التي لا يكون المعالج فيها في وضع الخمول.

نقول عن نظام أنه مُحتَل تحميلًا زائداً إذا كانت قيمة U أكبر من 100% من غير المفضل أن تكون انشغالية النظام عالية، لأن أي تغيير أو إضافة لمهام جديدة يمكن أن يؤدي للتحميل الزائد. وذلك فإن النظام ذات الانشغالية المنخفضة ليست بالضرورة جيدة لأنها تدل على سوء التصميم الهندسي للنظام وأنه كان من الممكن تخفيف كلفة النظام باستعمال تجهيزات مادية أقل أداء وكلفة.

عادة ما تكون الانشغالية في النظام الحديث حوالي 50%، بينما يمكن لبعض النظم التي لا يُوقَّع نموها أن تكون انشغالية بعض 80%. على كل حال، تعتبر القيم الأمثلية للانشغالية في نظام الزمن الحقيقي ذات المهام الدورية والمستقلة هي 70%. بين الجدول التالي بعض القيم الممكنة للانشغالية والحالات النموذجية الموافقة لها:

<table>
<thead>
<tr>
<th>التطبيقات النموذجية</th>
<th>المنطقة التي يعمل فيها النظام</th>
<th>الانشغالية (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>متعددة</td>
<td>إمكانيات معالجة زائدة جداً - المعالج أقوى بكثير من الحاجة</td>
<td>0-25</td>
</tr>
<tr>
<td>متعددة</td>
<td>امنة جداً</td>
<td>26-50</td>
</tr>
<tr>
<td>متعددة</td>
<td>امنة</td>
<td>51-68</td>
</tr>
<tr>
<td>النظم المضمنة</td>
<td>الحد النظري</td>
<td>69</td>
</tr>
</tbody>
</table>
يمكن حساب $U$ بجمع معاملات انشغالات الإجراءات الدورية وغير الدورية المشاركة. يفرض أن النظام يحتوي على $n \geq 1$ إجراء دوري دور كل منها يساعي $p_i$ (وبالتالي يكون معدل تنفيذها $e_i$) إذا علمنا أن زمن التنفيذ الأسوأ $(	ext{في أسوأ الأحوال})$ للإجراء $i$ هو $e_i$.

$$u_i = \frac{e_i}{p_i}$$

(1.1)

بذلك تكون الانشغالية الكلية $U$ للنظام مساوية لـ:

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

(1.2)

عادة ما يكون الحد الزمني للإجراء الدوري رقم $i$ (رمزه $d_i$) محدداً ببداية دورة $e_i$ التالية. القيمة المنطقية العظمى لـ $d_i$ هي $e_i$ هي $d_i$. يمكن أن تكون عملية تحديد قيمة قبل أو بعد تنفيذ البرنامج صعبة جداً أو مستحيلة في بعض الأحيان، عندما تقترر قيمتها تقريراً. في حال الإجراءات الغير دورية والمتباعدة $sporadic$ والزمن الفاصل بين حواسيب $aperiodic$، نحسب $u_i$ بدلالة الحالة الأسوأ لزمن تنفيذها $e_i$. تختلف انشغالية المعالج عن أدائه $throughput$ خلال المعالج في الثانية. عادة ما يستعمل هذا القياس لمقارنة أداء المعالج المختلفة عند تنفيذها تطبيقات محددة.

2- عوامل تتعلق بتصميم نظام الزمن الحقيقي

تعتبر نظم الزمن الحقيقي فرعاً معقداً من هندسة النظم الحاسوبية وتأثر بكل من نظرية التحكم وهندسة البرمجة وبحوث العمليات (بسبب الحاجة لجدولة الإجراءات $(scheduling)$ المحادثة). يبين الشكل التالي بعض فروع هندسة الحاسوب والبرمجة التي تؤثر بتحليل وتصميم نظم الزمن الحقيقي.
يجب الانتباه إلى عدة مشاكل يجب حلها عند تصميم وتحقيق نظم الزمن الحقيقي وتتضمن:

- اختيار العتاد المادي والبرمجيات وإيجاد الحل الأمثل بالتكلفة الأقل.
- تحديد مواصفات النظام وتصميمه وتحليل التمثيل الأفضل لتصريفه الزمني.
- فهم الفروق بين لغات البرمجة المختلفة ومدى ملاءمتها لنظم الزمن الحقيقي.
- تصميم النظام بحيث تكون وثوقيته وتحمله للأخطاء أكبر ما يمكن.
- اختيار النظام بدقة.
- الاستفادة من تقنية النظم المفتوحة والمتوافقة قدر الإمكان. النظام المفتوح هو مجموعة مستقلة وقابلة للتوسع من التطبيقات التي تتعاون لعمل نظام متكامل. على سبيل المثال، يستعمل حالياً عدد من نسخ نظام التشغيل المفتوح Linux في تطبيقات الزمن الحقيقي. تقاس التوافقية بمدى تحقيق النظام لمقاييس النظم المفتوحة مثل مقياس كوربا CORBA.
- قياس وتقدير أزمان الاستجابة للأقراص ومحاولة تصغيرها قدر الإمكان، وإجراء تحليل لإمكانية جدولة الإجراءات، أي إمكانية إيجاد ترتيب ما لجدولة الإجراءات بحيث تحتتم جميعها حدودها الزمنية. تدخل هذه الدراسة في نطاق نظرية الجدولة، وهي جزء من بحوث العمليات.

يمكن بالطبع استعمال التقانات الهندسية لنظم الزمن الحقيقي الصعبة لتصميمه. يمكن استخدام الأنواع الأخرى من نظم الزمن الحقيقي للحصول على نظام أفضل وأكثر وثوقية ومتانة. هذه أحد أساليب أهمية دراسة نظم الزمن الحقيقي.
3- أمثلة على نظم الزمن الحقيقي

إن نظم الزمن الحقيقي المضمونة متنوعة جداً وتوجد حتى في الألعاب والأجهزة المنزلية. ببين الجدول التالي بعض العينات من نظم الزمن الحقيقي ومجالات تطبيقاتها.

<table>
<thead>
<tr>
<th>التطبيق</th>
<th>المجال</th>
</tr>
</thead>
<tbody>
<tr>
<td>الطيران</td>
<td></td>
</tr>
<tr>
<td>شاشات الإظهار</td>
<td></td>
</tr>
<tr>
<td>التعدد الوسائط</td>
<td></td>
</tr>
<tr>
<td>الألعاب</td>
<td></td>
</tr>
<tr>
<td>نظم المحاكاة</td>
<td></td>
</tr>
<tr>
<td>جراحة الروبوتات</td>
<td></td>
</tr>
<tr>
<td>جراحة البعيدة</td>
<td></td>
</tr>
<tr>
<td>التصوير الطبي</td>
<td></td>
</tr>
<tr>
<td>خطوط التجميع المؤتمتة</td>
<td></td>
</tr>
<tr>
<td>التفتيش الآلي</td>
<td></td>
</tr>
<tr>
<td>المصاعد</td>
<td></td>
</tr>
<tr>
<td>السيارات</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

توصيف صوري لهذه النظم، بل توصيفها بطريقة تظهر أهمية نظم الزمن الحقيقي في حياتنا العملية. سنناقش لاحقاً طريقة توصيف هذه النظم بطريقة دقيقة وصورة.

 لنعتبر نظام القياس العطالي في الطائرة. يبين الشكل التالي الحركات الممكنة التي يمكن أن تقوم بها الطائرة والتي يجب على النظام قياسها.
حسب مواصفات النظام، تطلب تجهيزات خاصة بن巴斯ات قياس التسارع على المحاور $x$ و $y$ و $z$ إلى برنامج التحكم كل 10 ميلي ثانية. يحسب البرنامج تسارع الطائرة على المحاور الثلاث، بالإضافة إلى قيم اللف والانحدار والزيغ الحالية للطائرة. يصل كذلك للبرنامج قيمة درجة الحرارة كل ثانية. يجب عليه حساب شعاع السرعة اعتمادًا على هذه المعطيات كل 40 ميلي ثانية والظهورها للطيار.

النافذ مثل آخر وهو نظام مراقبة لヶفاع لمفاعل نووي عليه التعامل مع ثلاثة أحداث يُطل عليها بمطلقات. يحدث الحدث الأول عندما تكتشف نقاط المراقبة الأمنية أي اختراق أمني للمفاعل. يجب على النظام في هذه الحالة الاستجابة خلال ثانية واحدة لإطلاق أجهزة الإنذار. يدل الحدث الثاني والأهم على حدوث ارتفاع كبير لدرجة الحرارة في المفاعل، ويجب الاستجابة له خلال ميلي ثانية واحدة فقط. أخيرًا، يجب تحديث شاشة الأظهار 30 مرة في الثانية.

المثال التالي هو نظام حجز التذاكر لشركة طيران. قررت إدارة الشركة أنه يجب ألا يتجاوز زمن الانتظار الكلي لأي عملية حجز 15 ثانية حتى لا يؤدي ذلك إلى عدم رضا الزبائن. يجب كذلك ألا يسمح النظام بالحجز الزائد عن عدد المقاعد المتوفرة. يمكن لعدة وكلاء أن يحاولوا الوصول إلى قاعدة المعطيات في نفس الوقت لحجز نفس المقعد. لذلك يجب تزويدي النظام بإمكانات الاتصال والنقل اللازمة لذلك.

المثال الأخير هو نظام تحكم بإشارات المرور على تقاطع طرق. يُعد إليه السيارات من أربعة اتجاهات (شمال، جنوب، شرق، غرب). يتحكم النظام بإشارات السيارات والمشاة ويتلقى المعلومات من العالم الخارجي عن طريق حساسات تحت الأرض لقياس معدل تدفق السيارات في كل اتجاه، بالإضافة إلى كاميرات المراقبة والأزرار التي تضغطها المشاة عند رغبتهم بقطع الشارع. يجب على إشارات المرور أن تعمل بتزامن تمام والاستجابة كذلك لأحداث غير متزامنة مثل الضغط على زر المشاة، والإشارة حواردة مميزة.
4- بعض المعتقدات الخاطئة المتعلقة بنظم الزمن الحقيقي

من المهم أن نذكر بعض المعتقدات الشائعة الخاطئة لفهم طبيعة نظم الزمن الحقيقي فهماً كاملاً.

بل ن_pressenter

"نظام الزمن الحقيقي" هي من مراوات "النظام السريع": يأتي هذا الاعتقاد الخاطئ من حقيقة أن العديد من نظم الزمن الحقيقي تتعامل حقاً مع حدود زمنية من رتبة الميلي ثانية، كما هو حال نظم الملاحة الجوية. لكن هناك كذلك نظم زمن حقيقي أخرى ببطيئة مثل نظام حجز تذاكر الطيران. المهم في نظام الزمن الحقيقي هو النجاح في تحقيق الحدود الزمنية ولا تصلح بالسرعة.

تُرار خوارزمية الجدولة RM (Rate Monotonic) من أنظمة تشغيل زمنية أكثر شعبية جدولت الإجراءات الدولية شهرةً في أدبيات نظم الزمن الحقيقي منذ عام 1970. وهي بسيطة وتعطي عادة نتائج جيدة، لكنها ليست دائماً الحل السحري لمشكلة الزمن. سندرس هذه الخوارزمية بالتفصيل في الفصول اللاحقة.

هناك طرق عامة ومقبولة دائماً لتصميم ووضع مواصفات نظام الزمن الحقيقي: لا توجد للأسف مثل هذه الطرق وذلك لصعوبة إيجاد حلول عامة لكل الحالات. يجب دراسة وتصميم كل نظام على حدة حسب خصوصيته.

لا حاجة لبناء نظام تشغيل الزمن الحقيقي من الصفر وذلك لتوفير الكثير من الحلول التجارية: صحيح أن هناك العديد من نظم تشغيل الزمن الحقيقي الشائعة ذات الكلفة المقبولة، لكنها ليست مناسبة لكل تطبيقات الزمن الحقيقي. تحتاج أحياناً لكتابة نظام تشغيل مناسب لتطبيق محدد من الصفر، أو تعديل نظام تشغيل Jáhec ليتناسب مع خصوصية التطبيق. كذلك فإن اختيار نظام التشغيل المناسب من بين النظم المتوفرة ليس بالأمر السهل.

تتعلق معظم دراسة نظم الزمن الحقيقي بدراسة نظرية الجدولة: غالباً ما تكون النتائج التي تحققها الباحثين في نظرية الجدولة غير عملية وليس لها قيمة كبيرة هندسياً لذلك سنناقش في هذا الكتاب الخوارزميات القابلة للتطبيق عملياً فقط.

5- لملحة تاريخية مختصرة

يرتبط تطور نظام الزمن الحقيقي بشكل أو بآخر بتطور الحاسوب. إن العديد من نظام الزمن الحقيقي الحديث مثل نظم التحكم بمحيطات توليد الطاقة النووية ونظم أسلحة الطائرات والصواريخ هي نظم معقدة جداً، لكن يركز الكثير منها على مفاهيم كلاسيكية طورت ما بين الأربعينيات والستينيات من القرن العشرين.
التطورات النظرية

إن معظم نظريات نظم الزمن الحقيقي مشتقة من نظريات فروع العلوم الأخرى التي تعمدها والتي ذكروا سابقاً، وتأثرت بشكل خاص بالتطورات النظرية لبحث العمليات التي حصلت في نهاية الأربعينيات ونظرية الأرطال. T

تبلورت النتائج النظرية المتعلقة بوثوقية وتنبؤية نظم الزمن الحقيقي في الثمانينيات والتسعينيات من القرن العشرين. وجرت كذلك في هذه الفترة دراسة المشاكل المتعلقة بنظم الزمن الحقيقي متعددة المعالجات.

بعض نظم الزمن الحقيقي المبكرة

يمكن اعتبار أن بداية ظهور مفهوم نظام الزمن الحقيقي بشكل واضح كان في النظامين US Navy Whirlwind (وهو نظام محاكاة طيران طورته شركة IBM لصالح البحرية الأمريكية) و SAGE (وهو نظام محاكاة للدفاع الجوي طور لصالح القوى الجوية الأمريكية US Air Force) في عام 1947 في بدايات الخمسينيات. وحقق هذين النظامين المعايير التي تجعلهما نصفي نظام زمن حقيقي.

من الأمثلة الأخرى على نظم الزمن الحقيقي المبكرة نجد النظام SABRE، وهو نظام حجز الخطوط الجوية طورته الخطوط الجوية الأمريكية عام 1959.

برمجيات الزمن الحقيقي المبكرة

كتب برمجيات الزمن الحقيقي الأولية بلغة الألفة أو لغة التجميع Assembly، ثم استعملت لاحقاً اللغات ذات المستوى الأعلى لهذا الغرض مثل Fortran و CMS-2. وADA-95 سميت لحل بعض المشاكل التي اكتسبتها من النسخة الأصلية.

لكن يستعمل عدد قليل فقط من المبرمجين لغة ADA-95 اليوم، بينما يستعمل الأغلبية لغة C أو C++ لغة التجميع. يعود السبب على الأغلب لتعقيد لغة ADA-95 وعدم توفر مترجماتها لها على نظام التشغيل الشائع. وفي الآونة الأخيرة، ظهر توجه لاستعمال لغة البرمجة غرضية التوجه (بالأخص جافا JAVA) في نظام الزمن الحقيقي المضمنة. سناذا لغات البرمجة لنظم الزمن الحقيقي لاحقاً في فصل مستقل.
نظم تشغيل الزمن الحقيقي التجارية

صنعت نظم تشغيل الزمن الحقيقي التجارية الأولية للعمل على الحواسيب العملاقة IBM في العام 1962 مثل نظام Basic Executive، والذي يحقق العديد من خوارزميات الجدولة في الزمن الحقيقي. وفي العام 1963، صدر النظام الذي سمح بأن تقيم ببرامج المستخدم والنظام على القرص. Basic Executive II


في أواخر السبعينيات وأوائل الثمانينيات، ظهرت نظم التشغيل المخصصة للمعالجات الصغيرة مثل أنظمة VRTX و MROS 68K و RMX-80 وغيرها. خلال العشرين سنة الماضية، ظهر الكثير من نظم تشغيل الزمن الحقيقي واختفى العديد منها أيضاً.
الفصل الثاني: نظم تشغيل الزمن الحقيقي

1- نواة نظام تشغيل الزمن الحقيقي

الإجراء (ويسمى أيضاً المهمة) task هو برنامج قيد التنفيذ، وهو الوحدة المنطقية التي يجدولها نظام التشغيل. الجدولة scheduling هي عملية انتقاء الإجراء الذي سيقوم المعالج بتنفيذه في لحظة محددة من بين كل الإجراءات الجاهزة للتنفيذ. يُمثل كل إجراء في نظام التشغيل ببنية معطيات (تسجيلة record) تحتوي على الأقل على حالة الإجراء الحالية (جاهز ready، مُحمّل، مُحذوف middleware) وفي حالة تنفيذ executing، ورقم تعريف للإجراء identity number، بعض الواصفات الأخرى (زمن التنفيذ الكلي ...). وجموعة المصادر (ملفات، أجهزة ..) التي يحجزها هذا الإجراء.

يشترك بالمصدر مع بعض النياسب الأخرى lightweight thread هو إجراء خفيف يتألف كل إجراء من عدة نياسب (واحد على الأقل) تشاركه مصادره التي يحجزها. يصعب عادة Mشاركة المصادر بين عدة إجراءات، لكن المشاركة سهلة بين النياسب المكونة لنفس الإجراء.

يجب على نظام تشغيل الزمن الحقيقي أن يؤمن ثلاثة خدمات للإجراءات: الجدولة scheduling والتوظيف tread واتصال والتزامن بين الإجراءات dispatching والاتصال والتزامن بين الإجراءات synchronization العمليات التنظيمية. توفر النواة الصغرية kernel هذا الهدف، كما ذكرنا سابقاً، يحدد المجدد scheduler جدولة الإجراءات التوائي multitasking، بينما يقوم الموزع dispatcher بالعمليات التنظيمية اللازمة لتنفيذ الإجراءات. أخيراً، يضمن الجزء الثالث أن الإجراءات تتعاون وتتواصل فيما بينها.

يبد الشكل التالي طبقات نظام التشغيل المختلفة والمهمات التي تقوم بها.

المستخدم

- برنامج shell
- برنامج دعم للأقراص والملفات
- التشفير Executive

نواة التشغيل

- النواة الصغرية micro-kernel
- النواة الصغيرة nano-kernel

الناذروخاء

إدارة النياسب

النواة عمليات التزامن والاتصال بين الإجراءات. توفر semaphore وصانداق.
البريد وغيرها من الطرق الأخرى. الجزء "التنفيذي" 
من نظام تشغيل Executive 
الزمن الحقيقي هو نواة تتضمن عمليات إدارة كل الذاكرة وأجهزة الدخول/الخراج وغيرها من 
العمليات المعقدة. معظم أنظمة تشغيل الزمن الحقيقي التجارية هي فقط هذا الجزء "التنفيذي" ولا 
تتضمن الطبقات الأدنى. أخيراً، تتضمن طبقة نظام التشغيل الأعلى واجهات التخاطب مع المستخدم 
والأمن وإدارة الملفات وغيرها.

مهما كان بنية نظام التشغيل المستعمل، يبقى الهدف الأساسي منه هي تلبية متطلبات الزمن 
الحقيقي وتأمين بيئة عمل متعددة المهام متينة ومرنة.

1-1-1 أشباه النوى

يمكن الحصول على إمكانيات الزمن الحقيقي بحلول بسيطة دون الحاجة إلى نواة كبرى من نظام 
التشغيل. هذه الحلول معقدة إذا وجدنا أنها تكفي في التطبيق ما، لأن نظام الحفظ الناتج أبسط وأسهل 
تحليلها. سنستعرض فيما يلي أهم هذه الأساليب.

1-1-1 حلقة الاستقصاء

عادة ما تُستخدم حلقة الاستقصاء للاستجابة السريعة لجهاز واحد. وهي تتتألف من حلقة غير منتهية 
تحوي تعليمات اختبار متكرر لعلام flag 
يدل على حدوث أو عدم حدوث حدث ما. تسمى عملية 
الفحص المتكررة هذه "استقصاء" polling.

على سبيل المثال، نفترض أننا بحاجة لنظام برمجي لمعالجة حزم packets 
من المعطيات بمعدل 
غير منتهية. نفترض أنه لدينا علم اسمه packet_here 
عمليات حزم من المعلومات. بين البرنامج البسيط التالي حلقة استقصاء 
بتعمل مع حزم المعطيات الواردة:

```
for ( ; ; ) {
    if (packet_here) {
        /* check flag */
        process_data();
        /* process data */
        packet_here = 0;
        /* reset flag */
    }
}
```

تناسب هذه الطريقة حالة نظام مكون من معالج وحيد مخصص للتعامل مع دخل/خرج سريع. عادة 
ما تُستخدم هذه الطريقة كإجراء خلفي background task في النظام المقعد بالمطاطارات، أو 
كإجراء دوري. تدور حلقة الاستقصاء في هذه الحالة عدداً قليلاً من المرات كي لا تستأثر بالمعالج 
ولتسمح لباقي الإجراءات بالتشفير، وتقوم بباقي الإجراءات بتنفيذ المعالجة التي لا علاقة لها بالحدث.
ملاحظة: هذه الطبقة مشابهة للطريقة السابقة، لكنها تضيف تأخيرا زمنيا بين لحظة الاستجابة للحدث ولحظة
العمر (أي قبل السطر "packet_here=0;" في المثال السابق). يستخدم من هذه reset
الخوارزمية لاستجابة للأحداث التي تتصف بما يسمى "بارتداد المفتاح". تحدث هذه الظاهرة عندما يكون مسبب الحدث مفتاحاً ميكانيكيأً أو كهربإياً، لأنه من المستحيل صنع مفتاح يغير حالتته بحذاء دون اهتزاز لتماساته. بين الشكل التالي مخطط زمنياً لمفتاح لحظة إغلاق تماساته. يظهر جلياً من المخطط أن اهتزاز تواصلات المفتاح سبب عدة الأحداث متتالية، الأولى منها حدث في اللحظة $t_0$ وهو صحيح والأحداث التالية التي حدثت في اللحظات $t_1$ و $t_2$ هي خاطئة. عند إضافة تأخير زمني كافى (أكبر من الفرق الزمني بين $t_0$ و $t_2$ قبل تصير علام الحدث، لن تسبب الأحداث الخاطئة استجابات من النظام لأنه يكون عناها في حالة انتظار انتقالة فترة التأخير الزمني.

على سبيل المثال، إذا فرضنا أن ارتداد المفتاح يزول بعد 20 ميلي ثانية، يمثّل البرنامج التالي المكتوب بلغة C حالة استقصاء متزامنة مناسبة له. تعليمة (20) pause هي استدعاء لخدمة من نظام التشغيل تجمد تنفيذ البرنامج لمدة 20 ميلي ثانية.

```c
for ( ; ; ) {  
    if (flag) {  
        process_data();  
    }  
    pause(20);  
    flag = 0;  
} /* do forever */
/* check flag */  
/* process data */  
/* wait 20 ms */  
/* reset flag */
```
تتميز حلقات الاستقراض بأنها مناسبة جداً للاستجابة لأقنيّة المعطيات السريعة، وخاصة عندما تحدث الأحداث بحذار غير منتظمة ويكون المعالج مخصصاً للمعالجتها فقط. لكنها غير كافية عندما يكون النظام معقداً، وهي تضيع الكثير من وقت المعالج وخصوصاً عندما تكون الأحداث قليلة الحدوث.

**Cyclic Executive**

نظام "التنفيذ الدوري" هو نظام غير مقام بالمقاطعات (أي لا يتعلق بالمقاطعات في عمله) ويعطي انتباطاً بال التنفيذ المتوازي لعدة إجراءات قصيرة على معالج سريع. يتوقف النظام من حلقة غير منتهية تستدعي الإجراءات على التالي كما هو مبين في المثال التالي:

```c
for ( ; ; ) { /* do forever */
    Process_1();
    Process_2();
    ...
    Process_n();
}
```

نلاحظ في هذا المثال أن زمن المعالج موزّع بالتساوي على جميع الإجراءات لأنه يدور عليهم بأسلوب "الشريط الدوار". يمكن بسهولة تعديل الحلقة السابقة لإعطاء إجراءات محددة أزمان تنفيذ أكبر (وبالتالي أولوية وسرعة استجابة أكبر) كما هو مبين في المثال التاليشي الذي زمن تنفيذ يساوي ضعف زمن تنفيذ الإجراءات الأخرى.

```c
for ( ; ; ) { /* do forever */
    Process_1();
    Process_2();
    Process_3();
    Process_3();
}
```

لنأخذ على سبيل المثال برنامج لعبة فيديو بسيطة تتألف من مدفع يتحرك لليمين واليسار في أسفل الشاشة وطلق الصواريخ على سفن فضاء تتحرك أعلى الشاشة. المهام الأساسية التي يجب أن ينفذها البرنامج هي تلبية أزرار لوحة المفاتيح وتحريك السفن الأعداء واختبار حدوث تصادم بين الصاروخ وسفينة فضاء أو بين صواريخ الأعداء والمدفع) وتحديث الشاشة. بالتالي، يمكن أن يكون الشكل العام للبرنامج كما يلي:

```c
for ( ; ; ) { /* do forever */
```
لا يوجد أي كود يمكن قراءته بشكل طبيعي من الصورة المقدمة.
الروتينات التعاونية

وتسمى أيضاً "تعددية المهام التعاونية" وتحتاج لبرمجة منضبطة وتطبيقات مناسب لكي تعمل بنجاح. تتألف النواة التي تعمل بهذا الأسلوب من إجرائين أو أكثر مكتوبين بأسلوب البرمجة المقامة بالحالة المشروحة في الفقرة السابقة. بعد الانتهاء من تنفيذ الحالة الفعالة من الإجراء الفعالة، يُستدعى موزع هجومي الذي يقوم باختيار الإجراء التالي الذي سيتم استدعاؤه. وبعد الانتهاء من تنفيذ الحالة الفعالة من هذا الإجراء، يُستدعى الموزع من جديد وهكذا.

تتصل الإجراءات فيما بينها باستعمال المتحولات العامة global variables، وأية معلومات يحتاج الإجراء للحفاظ عليها من انتقاء لآخر (من قبل الموزع) يجب كذلك أن يخزن في متحولات عامة.

لناخذ على سبيل المثال نظاماً تعاونياً مكوناً من إجرائين يعملان "على التفرع" وبشكل مستقل. بعد تنفيذ المرحلة phase_a1 من الإجراء process_a، يمكن تنفيذ الإجراء المركزي (بسبب تعليمة "return") الذي يستدعى بدوره phase_b1. بعد الانتهاء منphase_b1، ينتقل التنفيذ للموزع من جديد الذي يستدعي بدوره process_a و process_a، وهذا يبين البرنامج التالي الشكل العام للإجراءات بلغة C. حلاة الموزع لا تظهر في هذا البرنامج، لكنها بديهية.

```c
void process_a () {
    switch(state_a) {
        case 1: phase_a1();
    }
}
```
```c
void process_a( )  {
    switch(state_a)  {
    case 1: phase_a1( );
            return;
    case 2: phase_a2( );
            return;
    case 3: phase_a3( );
            return;
    case 4: phase_a4( );
            return;
    case 5: phase_a5( );
            return;
    }
}

void process_b( )  {
    switch(state_b)  {
    case 1: phase_b1( );
            return;
    case 2: phase_b2( );
            return;
    case 3: phase_b3( );
            return;
    case 4: phase_b4( );
            return;
    case 5: phase_b5( );
            return;
    }
}
```

المتحولات `state_a` و `state_b` هي متحولات حالة عامة لكل من الإجراءين `process_a` و `process_b` على الترتيب. يمكن بسهولة تعديل البرنامج السابق لجدولة الإجراءات بدلاً من إضافة وقت انتظار حدوث حدث. يجب طبعاً في هذه الحالة أن تتحول حلقة الاستئصال إلى فحص للعالم ثم الانتقال للموزع.

باختصار، يُعتبر أسلوب الروتينات التعاونية من أسهل خوارزميات الجدولة "العادلة" `fair`، يمكن تحقيقها بسهولة. كذلك يمكن لعدد مبرمجين أن يشاركون كتابة الإجراءات وليس من الضروري معرفة عدد الإجراءات مسبقاً. أخيراً، تملك بعض لغات البرمجة (مثل ADA) بنية جاهزة لتحقيق الروتينات التعاونية بسهولة.

هناك العديد من نظم الزمن الحقيقي الضخمة التي جرى تحقيقها باستخدام الروتينات التعاونية، مثل نظام `IBM` التي طورته شركة `Customer Information Control System (CICS)`
لمعالجة الاقطارات التي يقوم بها الزبائن، ونظم OS/2 Presentation المطور من نفس الشركة. الأسف لا يمكن استعمال هذا الأسلوب إلا مع الإجراءات من قبل Managerv التي يمكنها أن تترك المعالج بفواصل منتظمة والتي يمكن تقسمها لأجزاء منتظمة. كذلك فإن طريقة الاتصال الوحيدة المتاحة بين الإجراءات هي باستعمال المجلات العامة، وهو أمر غير مرغوب به عموماً.

**Interrupt-driven systems**

في هذا النوع من النظم، يتألف البرنامج الرئيسي من حلقة فارغة (لا تحتوي تعليمات) غير منتهية بالإضافة إلى الإجراءات المختلفة التي تجدول بامتثال معايير مادية في إعداد محاكاة الفئة Dispatcher. تجري عملية التوزيع . تجري عملية التوزيع المادية und utilitarian, غير استخدام الأسلوب إلا مع الإجراءات التي تتطلب لإجراء الشيائية المادية، لكن مصدر المقاطعة هو تعليمات خاص مهندس البرنامج المكتوبة لأجله. يوجد نوعان لل)}. تجري عملية التوزيع．

من المهم فهم كيفية عمل المقاطعات عند كتابة برامج برمجيات النظم المضمنة، لأنه غالباً ما يتطلب برنامج الزمن الحقيقي تلبية مقاطعات تجهيزات النظام المختلفة. إذا أن يقوم مهندس البرمجيات كتبة سوأءة الجهاز device driver من الصفر (وهي هنا تلعب دور روتينات تخديم المقاطعة)، بما يتناسب مع خصوصية الجهاز المادي generic، أو يقوم بتعديل وتحسين سوأءة جهاز عامة software المعالج حسب ترتيب وروده وأولوياتها. إذا كان التصميم المادي للنظام (المعالج بشكل أساسي) يدعم عدة مقاطعات، عندما يقوم العداد المادي أيضاً بعملية التوزيع (أي استدعاء الإجراء الموافق للمقاطعة). أما إذا دعم المعالج مقاطعة واحدة فقط، عندما تصبح مسؤولية التوزيع مقاطعة على عات أرف تلقي المقاطعة وذلك بسوا محاكاة الفئة من مصدر المقاطعة ثم التقفر إلى الإجراء المسؤول عن تلبيتها.

**Interrupt Service Routines (ISR)**

منظفات المقاطعات: وهي إشارات تولدها التجهزات المحيطية والتي يقوم بها الزبائن، ونظم OS/2 Presentation المطور من نفس الشركة. الأسف لا يمكن استعمال هذا الأسلوب إلا مع الإجراءات من قبل Managerv التي يمكنها أن تترك المعالج بفواصل منتظمة والتي يمكن تقسمها لأجزاء منتظمة. كذلك فإن طريقة الاتصال الوحيدة المتاحة بين الإجراءات هي باستعمال المجلات العامة، وهو أمر غير مرغوب به عموماً.

- مقاطعات مادية: وهي إشارات تولدها التجهزات المحيطية والتي يقوم بها الزبائن، ونظم OS/2 Presentation المطور من نفس الشركة. الأسف لا يمكن استعمال هذا الأسلوب إلا مع الإجراءات من قبل Managerv التي يمكنها أن تترك المعالج بفواصل منتظمة والتي يمكن تقسمها لأجزاء منتظمة. كذلك فإن طريقة الاتصال الوحيدة المتاحة بين الإجراءات هي باستعمال المجلات العامة، وهو أمر غير مرغوب به عموماً.

- محاكاة الفئة: تلبية مقاطعات المادية، لكن مصدر المقاطعة هو تعليمات خاص مهندس البرنامج المكتوبة لأجله. يوجد نوعان لل}. تجري عملية التوزيع．

- مقاطعات برمجية: تشبه المقاطعات المادية، لكن مصدر المقاطعة هو تعليمات خاص مهندس البرنامج المكتوبة لأجله. يوجد نوعان لل}. تجري عملية التوزيع．
توصيف المقاطعات المادية بأنها ذات طبيعة غير متزامنة، أي أنها يمكن أن تحدث في أي لحظة. 
وإذن، ينطلق البرنامج الذي كان المعالج ينفذه وينتقل التنفيذ (فوراً أو بعد حين) إلى 
روتين تنفيذ المقاطعة ISR. يحتاج مطوري النظام عادةً لكتابة لبعض المقاطعات المادية، 
ومن المهم عندما أن يفهم جيداً ممّا تتألف حالة المعالج وما هي السجلات التي يجب أن 
ISR يحفظها.

عندما يتشارك أي برنامج مع روتن تنفيذ مقاطعة ما بعض المصادر المشتركة، فإن الطرق 
الوحيدة لتنسيق عملية الوصول إليها هي حسب المقاطعة قبل الوصول إليها في البرنامج وتأهيلها. 
من جديد بعد الانتهاء منها. السبب هو أنه لا يمكن استعمال طرق التزامن أو المعنى المتبادل 
المتعدد داخل روتن تنفيذ المقاطعة لأن ذلك يمكن أن يسبب تجميد فترة طويلة وغير محدودة بانتظار تحرير المصدر، وهذا أمر غير مرغوب فيه لأنه يمكن أن 
يؤثر على أداء النظام حسب فترة غير محدودة. ولذا، يمكن أن تكون المقاطعات متحركة، تصبح استجابة 
للنظام المناسبة الحالية ضعيفة جداً. لذلك يجب أن يكون الجزء من البرنامج الذي يحجب 
المقاطعات أقصر ما يمكن.

نقول عن ISR أنه "قابل لإعادة الدخول" reentrant إذا سمح بتلبية أي مطالبة قبل أن ينتهي 
الروتين تفويده (حتى ولو كانت المقاطعة الجديدة هي نفسها التي يقوم بتلبيتها حالياً) لكن حدث لسبب آخر). 
وهو ما يسمى تزامن حالة النظام (وما يسمى بالأمر) قبل أن يقوم بعمله حتى يكون بالإمكان متابعة تنفيذ الإجراء المقاطع بعد الانتهاء من تلبية المقاطعة.

context switch - 2

تعزز تبديل السياق بأنه عملية حفظ معلومات كافية (سياق) عن إجراء زمن حقيقي نتيجة مقاطعاته، 
بحيث يمكن متابعة تنفيذه لاحقاً بعد الانتهاء من تنفيذ روتن تخيد المقاطعة. يستعمل هذا المصطلح 
أيضاً عندما ينقل المزود التنفيذ من إجراء لأخر: تحفظ معلومات الأولى وتسترجع معلومات الثاني 
لإنهاء التنفيذ بدلاً من الأول.

. system stack يتضمن معالم السياق عادةً على هامش النظام السياق. تؤثر الزمن اللازم لتبديل 
السياق تأثيراً كبيراً على زمن استجابة النظام. لذلك، يجب أن يكون أمثلية وأصغر ما يمكن. عادةً ما 
يحتوي السجلات العامة للمعالج لحظة المقاطعة:

- محتويات السجلات العامة للمعالج لحظة المقاطعة.
قيمة سجل عداد البرنامج (في حال وجوده).

قيم سجلات المعالج الحسابي المساعد.

سجل صفحة الذاكرة.

مواقع الذاكرة التي تُعين تجهيزات دخل/خرج.

. program counter

. math coprocessor

-memory-mapped I/O

مواعظ الذاكرة التي تُعين تجهيزات دخل/خرج.

عادةً، يقوم روتين تخزين المقاطعات بتحجيم المقاطعات أثناء قيامه بتبديل السياق لأنها تُعتبر عملية حرجة يجب أن تتم دون مقاطعة، ثم يُفعلها من جديد بعد الانتهاء من التبديل.

يستعمل أسلوب تخزين السياق على المكدس في النظم المضمَّنة على الأغلب، لأن عدد إجراءات الزمن الحقيقي أو الإجراءات المقدمة بالأحداث محدد في هذه النظم. عندما تحدث المقاطعات، يستدعى المعالج روتين تخزين المقاطعة المناسب الذي يقوم بحفظ سياق الإجراء قيد التنفيذ (أو في منطقة محددة وساقطة في الذاكرة إذا كان النظام وحيد المقاطعة).

يُبين شبه البرنامج التالي جزءاً من نظام زمن حقيقي مكتوب بلغة C. ويتكون من برنامج رئيسي يحوي حلقة فارغة وغير منتهية، وثلاثة روتينات تخزين مقاطعة تستعمل أسلوب تخزين السياق على مكدس.

```c
void main( )
/* initialize system, load interrupt handlers */
{
    init();
    while(TRUE); /* infinite wait loop */
}

void int1 ( ) /* interrupt handler 1 */
{
    save(context); /* save context on stack */
    task1( ); /* execute task 1 */
    restore(context); /* restore context from stack */
}

void int2( ) /* interrupt handler 2 */
{
    save(context); /* save context on stack */
    task2( ); /* execute task 2 */
    restore(context); /* restore context from stack */
}
```
void int3( ) /* interrupt handler 3 */
{
    save(context); /* save context on stack */
    task3( ); /* execute task 3 */
    restore(context); /* restore context from stack */
}

يجب على الإجراءات (init) أن تحفظ عناوين التوابع الثلاثة int1 و int2 و int3 في مكان محدد من الذاكرة اسمه شعاع المقاطعات interrupt vector. عندما ترد مقاطعة إلى المعالج، يستخدم نظام مراقبة بمحطة توليد كهرباء رأس قيد التنفيذ روتين تخديم المقاطعة الموافق لها. تقوم الإجراءات init() بحفظ قيم السجلات الهامة على save() باسترراجاها منه restore().

preemptive-priority 3-1
نقول عن إجراء ذو أولوية أعلى أنه يقوم بشغف (اغتصاب، سرقة) المعالج من الإجراءات في التنفيذ والذي أولويته أقل إذا قاطعه وجعل المعالج ينفذ تعميماته. تُسمى النظم التي تستعمل أسلوب الجدولة first وهذا بدلاً من الأساليب الأخرى (الشريط الدوار round-robin أو القادم أولاً first come first serviced) بنظام الأولوية الشفعة. تعتمد أولويات المقاطعات على أهمية والحاج المهمات المنطة بها على سبيل المثال، الطريقة البديلة للتوثيق نظام مراقبة بمحترد توليد كهرباء نووية هي طريقة الأولوية الشفعة. صحيح أن التعامل مع دخيل على المحطة هو أمر مهم، لكن شيء أكبر من التعامل مع إنكار ارتفاع درجة حرارة نواة المفاعل، وله بالتالي الأولوية الأكبر.

يمكن أن تكون أولويات المقاطعات ساكنة أو ديناميكية (قابلة للتغيير). النظم ذات المقاطعات الساكنة أقل مرونة لأنه لا يمكن تغيير أولويات المقاطعات أثناء عمل النظام بما يتناسب مع تغيير احتياجاته، بينما يمكن عمل ذلك في النظم ذات المقاطعات الديناميكية.

تعاني نظام الأولوية الشفعة من مشكلة استثناء الإجراءات ذات الأولوية الأعلى بالمصادر المتاحة في النظام، مما يؤدي لحرم إجراءات ذات الأولوية الأدنى منها. تقول في هذه الحالة أن "الإجراءات ذات الأولوية الأقل تعاني من مشكلة "المجاعة" starvation".

Rate Monotonic
هناك نوع خاص من نظم الأولوية الشفعة المقددة بالمقاطعات تسمى النظم "رتبة المعطل Monotonic الأعلى للاجراء الدوري الذي له أعلى معدل تنفيذ (أي يُقدّر عدداً أكبر من المرات في الثانية). هذا الأساليب شائع في النظام المضمّنة وخصوصاً نظم الطيران، وهذا الكثير من الدراسات حوله. لناحه على سبيل المثال نظام الملاحة في الطائرة. الإجراء الذي يحصل المعطيات من جهاز قياس التسارع كل 10 ميلي ثانية له أعلى أولوية. يأتي في الدرجة الثانية الإجراء الذي يحصل معطيات
التوافزن كل 40 ملي ثانية. أما الإجراء الذي يظهر المعطيات على شاشة الطيار كل ثانية فله الأولوية الأدنى. سندرس النظم المتداخلة في الفقرات التالية بالتفصيل.

hybrid systems

النظام الهجين تحتوي هذه النظم على مقاطعات تحدث دورياً تعبر عن الإجراءات الدورية، بالإضافة إلى مقاطعات غير دورية تعبر عن الإجراءات غير الدورية أو المتبدعة. يمكن استعمال المقاطعات غير الدورية لمعالجة الأخطاء الحرة التي تحدث استثنائياً، ويجب معالجتها فوراً، وبالتالي يكون لها الأولوية الأعلى. عادة ما نجد هذا الخليط من المقاطعات في النظم المضمونة أيضاً.

هناك نوع آخر من النظم الهجينة موجود في نظام التشغيل التجاري وهو مزيج من النظم السريعة مع أساليب الشريط الدوار round-robin في هذه النظم، يمكن دائماً للإجراءات ذات الأولوية الأعلى أن تقوم بـshutting preempt الإجراءات ذات الأولوية الأدنى. لكن إذا وجد المجدول عدة إجراءات لها نفس الأولوية واجهزة للتنفيذ، يقوم بجدولتها بطريقة الشريط الدوار التي سندرسها لاحقاً.

foreground/background

هذا النوع من النظم هو تحسين النظم المقامة بالمقاطعة، حيث تستبدل حلقة الاستقصاء التي لا تقوم بأعمال مثيرة بحلقة تؤدي عملية معالجة مفيدة بانتظار حدوث الحدث، وهو النوع الأكثر استعمالاً لتحقيق النظم المضمونة. تتألف هذه النظم من عدد من إجراءات الزمن الحقيقي المقامة بالمقاطعة، وتسمى "الواجهة الأمامية"، بالإضافة إلى عدد من الإجراءات غير المقيدة بالمقاطعة وتمس "الواجهة الخلفية". بين النماذج التالي نماذج من هذا النوع.
تعمل إجراءات الواجهة الأمامية بطريقة الشريط الدوار أو الأولوية الشفعة أو النظم الهجينة. أما إجراء الواجهة الخلفية فهو قابل لأن يشفع من قبل أي إجراء من الواجهة الأمامية. لذلك فهو يمثل الإجراء ذو الأولوية الأدنى في النظام.


4-1. المعالجة في الواجهة الخلفية

بما أن الواجهة الخلفية هي إجراء غير مباشر بالمقاطعة، فهي تحتوي على المعالجة الغير حرجة زمنياً في النظام. صحيح أن إجراء الواجهة الخلفية له أدنى أولوية، لكنه يُنفذ دونماً إذا كانت انشغالية النظام على سبيل المثال، يمكن زيادة عدد في الواجهة الخلفية لإعطاء قياس عند إنشغالية النظام أو لاكتشاف حالة دخول أخذ أجزاء الواجهة الأمامية بحلقة غير متصلة (عندها لا تحدث أي زيادة في العدد لفترة طويلة).

يمكن أيضاً أن تحتفظ الواجهة الخلفية بجدول لكل إجراء واجهة أمانية يُصرف في بداية تنفيذ الإجراء المؤقت. إذا وجد إجراء الواجهة الخلفية أن هناك عداد لا يُصرف بالمعدل الكافي، فهذا يُفيد على أن الإجراء المؤقت له لا يُنفذ بالعدل المطلوب وأن هناك خطأ ما في النظام. يشبه هذا الأمر "مؤقت المتاح مادياً في بعض الأنظمة، ولكنه برامج بالطريقة التي watchdog timer"، كلب الحراسة.

4-2. التهيئة

يمكن أيضاً وضع عمليات اختبار ذاتي منخفضة الأولوية في إجراء الواجهة الخلفية. على سبيل المثال، يمكن في الكثير من الأنظمة إجراء اختبار كامل لمجموعة تعليمات المعالج في الواجهة الخلفية. هذا الأمر يجعل النظام متينًا وموثوقًا جداً. أخيرًا، يمكن أن نضع في الواجهة الخلفية عمليات تحديث لشاشات الإظهار منخفضة الأولوية أو الطباعة على طابعة أو التخاطب مع أي تجهيزات دخول/خرج بطانية ومنخفضة الأولوية.

4-3. التهيئة

التهيئة هي الجزء الأول من إجراء الواجهة الخلفية، وتتضمن الخطوات التالية:

- تحجب المقاطعات
- تهيئة شعاع المقاطعة والمكدس
- إجراء اختبار ذاتي للنظام
- عمليات تهيئة أخرى للنظام
تأهيل المقاطعة من جديد

من المهم أن تُحَجِّب المقاطعات في بداية عملية التهيئة، لأن بعض النظم تقلع بمقاطعات مؤهلة بينما لم تجهز الأمور الضرورية لاستقبال المقاطعات بعد، مثل شعاع المقاطعة والمكدس. من المهم كذلك إجراء اختبار ذاتي للنظام قبل تفعيل المقاطعات. بعدها، يصبح النظام جاهزاً للعمل.

1- 4- 4 - عمليات الزمن الحقيقي

إن عمليات الزمن الحقيقي في الواجهة الأمامية في نظام واجهة أمامية/واجهة خلفية تقابل المقدمة على إجراء واجهة أمامية، حيث يمكن تتبع حالة نظام فيه مقاطعة وحيدة ويحتوي على إجراء واجهة أمامية واحد EPI واجراء الواجهة الخلفية. يُفترض أن تعليم DPI تخطي المقاطعة وأن DPI يُحَجِّب المقاطعة آلياً عند حدوثها (ضمنياً دون استعمال EPI) إلى أن تُغَلَّب صرامةً باستدعاء DPI.

من أجل تحقيق عملية تبديل السياق، يجب حفظ قيم سجلات المعالج الثمانية على المكدس R1-R8 من قبل إجراء الواجهة الأمامية وذلك لأن إجراء الواجهة الخلفية يستعملها ولا يُدِر أن تضع قيمتها عند حدوث المقاطعة. سيعمل إجراء الواجهة الأمامية حتى ينتهي، لذلك لا حاجة لحفظ سياقه لأنه لن يُقَاطع. نفترض أخيراً أن عنوان روتين تخذي المقاطعة مخزَّن في العناوين 5 من الذاكرة (وهو شعاع المقاطعة).

تؤدي تعليمات التجميع التالية مهمة تهيئة هذا النظام البسيط:

\[
\begin{align*}
DPI & \quad; \text{disable interrupts} \\
\text{STORE} & \ & \text{&handler,5} \quad; \text{put interrupt handler address in location 5} \\
EPI & \quad; \text{enable interrupts}
\end{align*}
\]

إذا فرضنا أن reg0 وحتى reg7 تمثل عناوين في الذاكرة تستعملها لحفظ قيم سجلات المعالج، عندما يمكن أن نكتب روتين تخذي المقاطعة كما يلي:

\[
\begin{align*}
\text{DPI} & \quad; \text{redundantly disable interrupts, not required} \\
\text{STORE} & \ & \text{R0,} \ & \text{reg0} \quad; \text{save register 0} \\
\text{STORE} & \ & \text{R1,} \ & \text{reg1} \quad; \text{save register 1} \\
\text{STORE} & \ & \text{R2,} \ & \text{reg2} \quad; \text{save register 2} \\
\text{STORE} & \ & \text{R3,} \ & \text{reg3} \quad; \text{save register 3} \\
\text{STORE} & \ & \text{R4,} \ & \text{reg4} \quad; \text{save register 4} \\
\text{STORE} & \ & \text{R5,} \ & \text{reg5} \quad; \text{save register 5}
\end{align*}
\]
STORE R6,&reg6 ; save register 6
STORE R7,&reg7 ; save register 7

JU @APP ; execute real-time application program

LOAD R7,&reg7 ; restore register 7
LOAD R6,&reg6 ; restore register 6
LOAD R5,&reg5 ; restore register 5
LOAD R4,&reg4 ; restore register 4
LOAD R3,&reg3 ; restore register 3
LOAD R2,&reg2 ; restore register 2
LOAD R1,&reg1 ; restore register 1
LOAD R0,&reg0 ; restore register 0

EPI ; re-enable interrupts
RI ; return from interrupt

توجد في العديد من المعالجات تعليمات خاصة مثل save و restore تقوم بحفظ واسترجاع قيم جميع سجلات المعالج الهامة في مواقع متتالية من الذاكرة. لاحظ أن روتين المقاطعة هذا لا يسمح بأن يقاطع لأنه لا يعيد تفعيل المقاطعة حتى نهايةه. إذا أردنا أن نسمح بقاطع المقاطعة، يجب استعمال مكدس تخزين قيم سجل المعالج بدلاً من المواقع السكنية reg0 حتى reg7.

يمكن لإجراءات الواجهة الخلفية أن يشمل استدعاءاً لتابع التهيئة، بالإضافة لأية عمليات معالجة غير حرة زمنياً. على سبيل المثال، يبين البرنامج التالي المكتوب بلغة C مثل هذا الإجراء:

```c
void main ( )
/*allocate space for context variable */
int reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7;
/*declare other global variables here */
{
    init();    /*initialize system */
    while (TRUE)    /*background loop */
        background();    /* non-real-time processing here */
}
```

تتميز نظام الواجهة الأمامية/الواجهة الخلفية عادةً بزمن استجابة جيد لأنها تعتمد على العداد المادي لجدولة إجراءاتها، وهي الخيار المفضل لدى الكثير من مصممي نظم الزمن الحقيقي المضمنة. لكن توجد فيها نقطة ضعف أساسية وهي ضرورة كتابة الواجهات البرمجية المعقدة (سواءات الأجهزة).
للكثير من الأجهزة المحوسبة يدوياً، هذا الأمر صعب ومليء بالأخطاء والمشاكل. بالإضافة إلى ذلك، يناسب هذا النوع من النظم التطبيق البينية المواجهة الأمامية ثانياً ومعروفاً مسبقاً، لذلك يصعب استعمالها في التطبيق البينية المواجهة الأمامية. وعلى الرغم من أن اللغات التي تدعم الحجز الدينياميكي للذاكرة يمكنها التعامل مع عدد متغير من الإجراءات، لكن يبقى الأمر غير سهل.

4-5 نظم تشغيل الزمن الحقيقي كاملة المواصفات

يمكن تمديد نموذج الواجهة الأمامية/واجهة الخلفية ليصبح نظام تشغيل حقيقية كامل المواصفات بإضافة مهام أساسية في أي نظام تشغيل مثل واجهات شبكة وسواقات أجهزة وأدوات تنفيذ متقدمة وغيرها. هناك العديد من النظم التجارية المبنية بهذا الأسلوب debugging

Task Control Block (TCB)

1-5 نموذج كتلة تحكم الإجراء

هذا النموذج هو الأكثر استخداماً في نظم تشغيل الزمن الحقيقي التجاري الكاملاً المواصفات لأنه on-line يسمح بأن يكون عدد الإجراءات متغيراً. وهو يستعمل أيضاً في النظم النافعة المتصلة حيث يزيد عدد الزبائن المتصلين بالنظام ويستمر باستمرار، ويخصص لكل زبون متصل إجراء مستقل. يمكن استعمال عدة خوارزميات جدولة مع هذا النموذج، مثل الشريط الدوار والأولوية التشغيلية أو أية تركيبة منهما، على الرغم من أنه عادةً ما تستعمل خوارزمية الشريط الدوار معه. المشكلة الوحيدة في نموذج TCB هو التأخير الزمني الكبير الذي تتطلبه عملية الجدولة إذا كان عدد الإجراءات المنشأة كبيراً.

ينشئ النظام في هذا النموذج بنية معطيات خاصة (تسجيلة) لكل إجراء يعمل في الذاكرة (record). تسمى هذه التسجيلة "كتلة تحكم الإجراء", وتحتوي على الأقل على المعلومات التالية:

- قيمة سجل عدد البرنامج PC
- قيم سجلات المعالج في اللحظة التي قطعت فيها هذا الإجراء (السياق)
- رقم أو اسم تعريف للإجرا
- حالة الإجراء (جاهز، متوقف، في حالة سبات،...)
- أولوية الإجراء (في حال استعمال مفهوم الأولوية)

يذكر النظام كتلة التحكم هذه في بنية معطيات مناسبة (لاحقة متارية) على الأغلب.

1-5-1 حالات الإجراء

يدير نظام التشغيل الإجراءات بحفظ حالة كل منها في كتلة تحكمه. عادةً ما يكون الإجراء في إحدى الحالات الأربعة التالية:

- executing
الإجراءات قيد التنفيذ هو الإجراء الذي ينفذ المعالج في هذه اللحظة، وهو وحيد إذا احتوى النظام على معالج واحد فقط. إذا كان النظام متعدد المعالجات، يمكن لعدة إجراءات أن تكون في حالة "قيد التنفيذ". يمكن لإجراة أن يدخل هذه حالة بعید إنشائه (إذا لم يكن هناك أي إجراء آخر في حالة "جاهز للتنفيذ"), أو من حالة "جاهز للتنفيذ" (إذا كان مؤهلاً للتنفيذ حسب أولويته أو موقعه في رتبة الشريط الدوار للإجراءات الجاهزة للتنفيذ). عندما ينتهي تنفيذ الإجراء، ينتقل إلى الحالة "متوقف".

الإجراءات التي حالتها "جاهز للتنفيذ" هي تلك الإجراءات التي يمكن أن تنتقل لحالة "قيد التنفيذ" لست "قيد التنفيذ". يدخل الإجراء حالة "جاهز للتنفيذ" إذا كان "قيد التنفيذ" وانتهى حرصه الزمني التي خصصها المجدول له، أو إذا جرى شفعه لأولوية أعلى، كذلك، إذا كان الإجراء في حالة "متوقف"، وحدث الحدث الذي ينتظر، تصبح حالة الإجراء "جاهز للتنفيذ". الإجراءات في حالة "متوقف" أو "محتفظ" هي تلك الإجراءات التي تنتظر توفر مصدر محدد، في الاقتصاد ليست "جاهزة للتنفيذ". عندما يتوفر هذا المصدر، ينتقل الإجراء لحالة "جاهز للتنفيذ".

تُستخدم حالة السبات في النظام الذي يكون فيها عدد كتل التحكم محدوداً. يمكن توصيف هذه الحالة بأنها حالة إجراء موجود لكنه غير متاح لمجدول نظام التشغيل، لامتلاج جدول كتل التحكم. عند إنشاء إجراء جديد، يمكن أن يوضع في حالة سبات إلى أن يتوفر أحد الإجراءات الأخرى ويصبح مكانه متاحاً في جدول كتل التحكم. عندما يتوقف من حالة سبات إلى حالة "جاهز للتنفيذ".

1-5-2 إداره الإجراءات

يمكن النظر إلى نظام التشغيل على أنه الإجراء ذو الأولوية الأعلى في النظام. إن أية مقاطعة أو استدعاء لخدمة من خدمات النظام (مثل طلب مصدر ما) تستدعي نظام التشغيل. يحتفظ نظام التشغيل بعلامتين مترابطتين من كتل التحكم. الأولى لإجراءات الجاهزة والثانية للإجراءات المتوقفة. ويعتبر كذلك بجدول الموارد المتاحة وأخر لطلبات المصادر التي طلبتها الإجراءات. يحتوي كل كتلة إجراء نفس المعلومات التي يحتفظ بها روتين تدمير المقاطعة على المكدس. يبين الشكل التالي مثالاً لنبية كتلة تحكم الإجراء.

<table>
<thead>
<tr>
<th>رقم تعريف الإجراء</th>
<th>الأولوية</th>
<th>حالة الإجراء</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
 impaired نموذج TCB بأنه مناسب عندما لا يكون عدد إجراءات النظام محدداً خلال مرحلة تصميمه، أو يمكن أن يتغير عددها أثناء عمله. أي أنه نموذج مرن جداً.

الأسلوب الأبسط لعمل المجدول في هذا النموذج هو على الشكل التالي: عند استدعائه (دورياً أو عند حدوث حدث)، يفحص المجدول اللائحة المتزامنة للإجراءات الجاهزة للتنفيذ. إذا وجد إجراء في هذه اللائحة، عندما تنقل كتلة لائحة الإجراءات الجاهزة إلى آخر لائحة الإجراءات الجاهزة، وينفي المجدول أحد الإجراءات الجاهزة للتنفيذ لفظه إلى حالة "قيد التنفيذ".

يمكن تبسيط عملية إدارة الإجراءات بتغيير حالاتها فقط دون نقلها من لائحة مترابطة لأخرى. على سبيل المثال، بفرض أن عدد كتل التحكم الأعظمي ثابتًا، يمكن أن يحتفظ النظام بالعدد الأعظمي من كتل التحكم في لائحة واحدة وتكون كلها في حالة سبات عند إيقافه. يمكن بعدها إضافة إجراءات جديدة للنظام باختيار أحد الكتل التي في حالة سبات وتغيير حالاتها إلى "جاهز للتنفيذ". أثناء عمل النظام، تتغير حالة الإجراءات بين القيم المتاحة لها ("قيد التنفيذ"، "جائحة التنفيذ"، "متوقف") حسب ما يطرأ عليها من أحداث وتغيرات في نفس اللائحة. عند انتهاء تنفيذ إجراء ما، يزال من النظام بتغيير حالته إلى "سكات". ميزة هذا الأسلوب أنه يلغي الحاجة إلى النافذة الديناميكية التي تتطلبها إدارة كتل التحكم، كما أنه يجعل أداء النظام محدداً وقابلًا للتبليغ به لأن لائحة كتل التحكم لها طول ثابت.

1- 5- 3- إدارة المصادر

بالإضافة إلى عملية الجدولة، يتم نظام التشغيل بمعرفة حالة كل مصدر متاح في النظام (جهاز دخل/خرج، ذاكرة، ...). وينفي المجدول جميع الإجراءات الموجودة في رتل الإجراءات المتوقفة. فإذا وجد إجراء متوفرًا لأنه ينتظر توفر مصدر ما، ووجد أن هذا المصدر أصبح متوفرًا، عندما يغير حالته إلى "جاهز للتنفيذ" وينقل كتلة تحكمه إلى رتل الإجراءات الجاهزة للتنفيذ.

<table>
<thead>
<tr>
<th>السجل 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>......</td>
</tr>
<tr>
<td>السجل n</td>
</tr>
</tbody>
</table>

سجل عداد البرنامج
سجلات حالة المعالج
مؤشر لكتلة التحكم التالية
نتجه في البداية لصياغة بعض المفاهيم النظرية بشكل رياضي وواضح لكي تتمكن من فهم نظام تشغيل الزمن الحقيقي. فنستعرض بعد ذلك خوارزميات جدولة إجراءات الزمن الحقيقي المختلفة وتناقش مزايا ومساوئ كل منها.

تتنصف معظم نظام الزمن الحقيقي بأنها تشاركية concurrent، أي أن طبيعة تفاعلها مع الأحداث الخارجية تتطلب عادة أن تقوم عدة إجراءات بالاتصال مع العالم الخارجي في نفس الوقت (على التفرع). تحدثنا سابقاً عن الإجراءات ورأينا أن لكل منها حالة يغيرها المجدول حسب حدوث أحداث عينة (انتهاء الحصة الزمنية لإجراة، طلب الإجراء لمصدر محجوز من قبل إجراء آخر...)، ويبين الشكل التالي الحالات المختلفة للإجراءات والأحداث التي تسبب الانتقالات فيما بينها.

توجب الإشارة أن بعض نظام التشغيل يستعمل مصطلحات مختلفة عن المصطلحات المستعملة هنا، لكن الأفكار العامة تبقى نفسها.

2- جدولة الإجراءات

تُعتبر الجدولة من المهمات الأساسية لنظام التشغيل. لكي نحقق الاحتياجات الزمنية لبرنامج زمن حقيقي، نحتاج لاستعمال خوارزميات جدولة مختلفة عن تلك المستعملة في نظام التشغيل العادي. نحتاج كذلك لأساليب ترتيب مؤقتها تستخدم مصادر النظام من قبل إجراءات الزمن الحقيقي حسب أولوياتهم وقيودهم الزمنية. هناك نوعين أساسيين لخوارزميات الجدولة في الزمن الحقيقي: خوارزميات ما قبل التنفيذ وخوارزميات زمن التنفيذ. الهدف المشترك لهذه الخوارزميات هو تحقيق الشروط الزمنية لإجراءات الزمن الحقيقي.

تقوم خوارزميات ما قبل التنفيذ بجدولة الإجراءات مسبقًا بحيث تكون الجدولة الناتجة محققة للقيود الزمنية ولا يوجد تضارب في استعمال المصادر (أي لا يستعمل أكثر من إجراء نفس المصدر في
نفس اللحظة)، وذلك قبل بدء التنفيذ الفعلي للإجراءات. تحاول هذه الخوارزميات أيضاً تخفيف كلفة تبديل السياق قدر الإمكان، مما يزيد احتمال إيجاد جدولة محققة لجميع القواعد الزمنية.

أما في خوارزميات زمن التنفيذ، فتعطي أولويات ساكنة للإجراءات، وتخصص المصادر حسب هذه الأولويات. تعتمد هذه الخوارزميات على تقنيات معقدة في زمن التنفيذ لتحقيق الاكتمال والالتزام بين الإجراءات.

1-1-1. العبء على المعالج وصفات المهام

يُعرّف العبء على المعالج بأنها مجموعة الإجراءات التي يجب على المعالج تنفيذها في لحظة محددة. يمكن لمعالج أن ينفذ إجراء واحداً على الأكثر في لحظة ما، وكل إجراء يُنفذ على معالج واحد على الأكثر في لحظة ما. يوصَف كل إجراء زمن حقيقي 

بالواصفات الزمنية التالية:

- قيد الأسقية (precedence constraints): يحدد فيما إذا كان هناك إجراء (أو إجراءات) يجب أن يُنفذ قبل تنفيذ 
 زمن الوصول أو التحرير (arrival or release): رمزه 

- تنفيذ التكرار رقم 

- الطور (phase): رمزه 

- زمن الاستجابة (response time): وهو الفترة الزمنية بين لحظة تحرير الإجراء ولحظة الانتهاء تنفيذه.

- الحد الزمني المطلق (absolute deadline): رمزه 

- الحد الزمني النسبي (relative deadline): رمزه 

- نوعية السماحة (laxity type): هو مقدار الإلحاح في تنفيذ الإجراء.

- الدور (period): هو الفترة الزمنية التي تفاضل بين لحظة التحرير لإجراء دوري ولحظة التحرير التالية مباشرة.

- اللازم لإتمام تنفيذ الإجراء (execution time): رمزه 

- الزمن (الأعظمي) اللازم للإجراء (urgency): رمزه 

هناك بعض العلاقات الرياضية التي تربط بين هذه الواصفات الزمنية:

\[
\varphi_i = r_i,1 \quad \text{and} \quad r_i,k = \varphi_i + (k - 1) \cdot p_i
\]

إذا فرضنا أن 


هذا الحد الزمني المطلق للتكرار رقم 

إذن لدينا:

\[
d_{i,j} = \varphi_i + (j - 1) \cdot p_i + D_i
\]
وإذا كان الحد الزمني النسبي لإجراء دوري $p_i$ يساوي دوره $\tau_i$ يتبع لدينا:

$$d_{i,k} = r_{i,k} + p_i = \varphi_i + k \times p_i$$

حيث $k$ هو عدد صحيح موجب أكبر أو يساوي 1، وهو يعبر عن التكرار رقم $k$ للإجراءات.

1-2- نموذج مبسط للإجراءات

سنعمد نموذجًا مبسطًا للإجراءات لكي نتمكن من توصيف بعض خوارزميات الجدولة النظامية المستعملة في نظام الزمن الحقيقي. يفترض هذا النموذج الفرضيات التالية:

- كل الإجراءات الموجودة في النظام دورية
- الحد الزمني النسبي لكل إجراء يساوي دوره
- كل الإجراءات مستقلة عن بعضها البعض، أي لا توجد قيود أسبقية بينها
- لا يجري أي إجراء قسمًا غير شفعي (أي غير قابل للمقاطعة)، وكل فتيلة السياق متماصلة
- لكل إجراء احتياجات معالجة فقط (أي يحتاج فقط للمعالج) وتُتم الانتفاعات الأخرى

(ذاكرة، دخل/خرج، ...)

من المهم جداً في نظام الزمن الحقيقي أن تكون الجدولة الناتجة عن خوارزمية الجدولة قابلة للتنبؤ predictable، أي أنه يمكننا معرفة الإجراء الذي سيُنفذ لاحقًا في كل لحظة. تستعمل العديد من نظم تشغيل الزمن الحقيقي خوارزمية الشريط الدوار الذي يُتميز بأنه بسيط وقابل للتنبؤ.

2- خوارزمية جدولة الشريط الدواّر

في هذه الخوارزمية، يجري تنفيذ الإجراءات تسلسلاً حتى تنتهي وغالبًا ما توجد في نظام التنفيذ quantum or cyclic executive الدوري. يُخصَص لكل إجراء شريحة (حصة) زمنية ثابتة slice يُنفذ خلالها. تستعمل ساعة ذات دور ثابت يساوي حالة الزمنية لمقاطعة النظام دوريًا. يستمر تنفيذ الإجراء حتى ينتهي تنفيذه أو تنتهي حبسه الزمني بعد مقطع الساعة. إذا لم ينته تنفيذ الإجراء، يُحفظ سياقه ويوضع في نهاية حل الإجراءات الجاهزة. بعد ذلك، يسترخى النظام سياق الإجراء التالي الموجود في بداية الرئل ويتتابع تنفيذه. إذا تعلقت هذه الخوارزمية جدولة عادلة غير شفعة للإجراءات التي لها نفس الأولوية بتوزيع زمن المعالج عليها بالتساوي.

يمكن إدخال مفهوم الأولوية الشفعة لخوارزمية الشريط الدوار، وذلك بالسماح للإجراءات ذات الأولوية الأعلى بشفع المعالج إذا كان ينفذ إجراء أقل أولوية لحظة وصولها. بين الشكل التالي مثالًا توضيحياً لهذه الفكرة.

وصول الإجراء وتنفيذه
الإجراءان A و C لهما نفس الأولوية والإجراء B له أولوية أعلى. يبدأ A بالتنفيذ لبعض الوقت، إلى أن يصل الإجراء B لفترة زمنية. يشغله A ويستمر بالتنفيذ حتى انتهاءه. بعدها يعود A للتنفيذ إلى أن تنتهي حصته الزمنية وينتقل التنفيذ للإجراء C حتى انتهاء حصته الزمنية. وهكذا.

**cyclic executive**

تستخدم هذا الأسلوب أيضاً بكثرة في العديد من أنظمة الوظائف الحقيقي لأنها تتيح جدولة قابلة للتنبؤ. وهو يقوم بتسليط ومراقبة تنفيذ إجراءات دورية حسب جدول محدد مسبقاً قبل التنفيذ. يمكن التعبير عن هذا الأسلوب ببساطة على أنه جدول استدعاءات متتابع، كل منها هو إجراء، في حلقة غير منتهية.

يتخذ هذا الأسلوب قرارات الجدولة دوريًا ومسبقاً وليس في أية لحظة مثل الشريط الدوار. تسمى الفترات الزمنية (المتساوية) التي تفصل بين اتخاذ قرارات الجدولة بالأطر أو frames، والرموز لطولها بالمترز، وتسمى "حجم الإطار". الدور الكبير هو المضاعف المشترك الأصغر لجميع أدوار إجراءات الأدوار الصغيرة major cycle or hyper period النظام.

بما أن قرارات الجدولة تأخذ فقط في بداية كل إطار، لذلك لا يحدث أي تداخل إطارين. إن أي إطار دوري هو مضاعف صحيح غير سالب من حجم الإطار. يتبع الشكل التالي بعض القواعد المفروضة على حجم الإطار f.

- **Frame k**
  - t
  - t'
  - t + f

- **Frame k + 1**
  - t + 2f
  - t' + p, t + 3f

- **Frame k + 2**
  - t + D
يجب أن يكون الإطار كبيراً كفاية بحيث يبدأ وينتهي أي إجراء ضمن إطار واحد. هذا يقتضي أن يكون حجم الإطار $f$ أكبر من كل أزمان التنفيذ $e_i$ للإجراءات. أي:

$$ C_1: f \geq \max_{i \in n}(e_i) $$

ولكي يكون طول الجدول الدوري أصغر ما يمكن، يجب أن يحتوي الدور الكبير على عدد صحيح من الأطر:

$$ C_2: \lfloor \frac{p_i}{f} \rfloor - \frac{p_i}{f} = 0 $$

والتأكد من أن كل إجراء ينتهي قبل حده الزمني، يجب أن يكون الإطار صغيراً بحيث يكون هناك إطاراً واحداً على الأقل بين كل تحرير لإجراء وبين حده الزمني. العلاقة التالية مشتقة من الحالة الأسوأ التي تحدث عندما يبدأ دور إجراء ما مباشرة بعد بدء الإطار، ولا يمكن بالتالي تحريره قبل بدء الإطار التالي:

$$ C_3: 2f - \gcd(p_i, f) \leq D_i $$

حيث $\gcd$ هو القاسم المشترك الأعظم و $D_i$ هو الحد الزمني النسبي للإجراء $i$.

سنوضح عملية حساب حجم الإطار $f$ باستخدام مثال عديدي. ليكن لدينا نظاماً يحتوي على الإجراءات الدورية المبينة في الجدول التالي:

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$p_i$</th>
<th>$e_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>15</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>22</td>
<td>3</td>
<td>22</td>
</tr>
</tbody>
</table>

قيمة الدور الكبير لهذه الإجراءات هي 660 (وهو المضاعف المشترك الأصغر للقيم 15 و 20 و 22). نطبق الشروط الثلاثة السابقة $C_3$ و $C_2$ و $C_1$ و نفصل على العلاقات التالية:

$$ C_1: \forall i f \geq e_i \Rightarrow f \geq 3 $$
$$ C_2: \lfloor \frac{p_i}{f} \rfloor - \frac{p_i}{f} = 0 \Rightarrow f = 2, 3, 4, 5, 10, \ldots $$
$$ C_3: 2f - \gcd(p_i, f) \leq D_i \Rightarrow f = 2, 3, 4, 5 $$

نستنتج مما سبق أن القيم الممكنة لحجم الإطار $f$ هي إما 3 أو 4 أو 5.
فكرة الخوارزمية

في النظم ذات الأولويات الثابتة، يُعطى لكل إجراء أولوية محددة نسبةً لباقي الإجراءات. تُعتبر خوارزمية RM من أهم خوارزميات الجدولة في هذه النظم، وهي تعطي جدولة أمثلية في نموذج الإجراءات المبسط المذكور سابقاً. يُعطي لكل إجراء في هذه الخوارزمية أولوية تتناسب عكراً مع دوره: فأولوية الإجراء الذي له دور كبير أصغر من أولوية الإجراء الذي له دور قصير.

سنتروح طريقة عمل هذه الخوارزمية على المثال التالي. ليكن لدينا مجموعة الإجراءات التالية:

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( e_i )</th>
<th>( p_i )</th>
<th>( u_i = e_i/p_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>1</td>
<td>4</td>
<td>0.25</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>2</td>
<td>5</td>
<td>0.4</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>5</td>
<td>20</td>
<td>0.25</td>
</tr>
</tbody>
</table>

نفترض أن زمن تحرير كل الإجراءات الثلاثة هو 0. يبين الشكل التالي طريقة جدولة هذا النظام باستعمال خوارزمية RM. بما أن الإجراء \( \tau_3 \) له أصغر دور، فإن أولويته أعلى من أولويات باقي الإجراءات، لذلك يُشغَّل أولأً. لاحظ أنه في اللحظة 4 يصل التكرار الثاني للإجراء \( \tau_1 \) الذي يقوم بتشغيل الإجراء \( \tau_3 \) لأن أولويته أعلى.

ном

وهي القيمة "الانشغالية" \( u_i \) التي يُشغِّل فيها إجراء له دور \( \tau_i \) وزمن تنفيذه \( e_i \) ووقت تنفيذه \( p_i \) المعالج. تذكر أن انشغالية المعالج في حالة \( n \) إجراء تعطى بالعلاقة 1.2 وهي:

\[
U = \sum_{i=1}^{n} \frac{e_i}{p_i}
\]
بعض نتائج خوارزمية RM

من المهم عملياً معرفة ما هي الشروط الواجب تحققها في نظام أولويات ثابتة ليكون قابلاً للجدولة (أي لكي توجد جدولة واحدة على الأقل تحقق جميع القيود الزمنية لجميع إجراءات النظام). تحدد النظرية التالية شروطاً على انشغال النظام لكي يكون قابلاً للجدولة حسب خوارزمية RM نفترض أن الحد الزمني النسبي لأي إجراء يساوي دوره.

نظري:

تكون أية مجموعة من الإجراءات الدورية عددها n قابلة للجدولة حسب خوارزمية RM إذا كانت RM

انشغال المعالج $U$ أصغر أو تساوي $\left(1 - \frac{1}{2^n}\right)$.

تجدر الملاحظة أنه عندما تصبح $n$ كبيرة جداً، يمكن البرهان على أن قيمة الحد السابق تصبح:

$$\lim_{n \to \infty} \left(2^n - 1\right) = \ln 2 \approx 0.69$$

الجدول التالي قيم الحد السابق من أجل عدة قيم لـ $n$:

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>الحد</td>
<td>1.0</td>
<td>0.83</td>
<td>0.78</td>
<td>0.76</td>
<td>0.74</td>
<td>0.73</td>
<td>0.69</td>
</tr>
</tbody>
</table>

بيين الشكل التالي مخططاً بين كيفية تغير حد الاتساغالية بدلاة عدد الإجراءات:

تجدر الإشارة إلى أن حد الاتساغالية هو شرط كافٍ وغير لازم، أي أنه يمكن في بعض الأحيان بناء نظام له الاتساغالية أعلى من الحد ويجود قابلً للجدولة حسب $RM$ رغم ذلك. على سبيل المثال، انشغال النظام السابق المكون من الإجراءات $\tau_1$ و $\tau_2$ و $\tau_3$ تساوي 0.9، وهي أعلى من حد الاتساغالية 0.78، ورغم ذلك فهو قابل للجدولة حسب $RM$ كما رأينا.
Earliest-Deadline- First (EDF)

على عكس نظم الأولويات الساكنة، تتغير أولويات الإجراءات أثناء عمل النظام في نظام الأولويات الديناميكية. تحتوي إستراتيجية EDF من أنواع أولويات هذه النظم، حيث تعتمد أرباب أولويات الإجراءات على حدودها الزمنية بدلاً من أزمان تنفيذها. في أية لحظة، الإجراء الذي له أقرب حد زمني هو الإجراء ذو الأولوية الأعلى.

تعطي النظرية التالية الشرط الواجب تحقيقه لكي تكون مجموعة إجراءات قابلة للجدولة حسب إستراتيجية EDF:

نظرية

في نظام زمن حقيقي يحتوي على n إجراء دوري حدودها النسبية تساوي دورها، الشرط اللازم لكي تكون مجموعة إجراءات قابلة للجدولة حسب EDF هو:

\[
\sum_{i=1}^{n} \left( \frac{e_i}{p_i} \right) \leq 1
\]

لنأخذ على سبيل المثال نظاماً يحتوي على الإجراءين التاليين:

<table>
<thead>
<tr>
<th>(\tau_1)</th>
<th>(p_1)</th>
<th>(e_1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

يبيّن الشكل التالي طريقة جدولتها حسب إستراتيجية EDF:

رغم أن لحظة تحرير كلا الإجراءين هي اللحظة 0، بدأ الإجراء \(\tau_1\) بالتنفيذ لأن له حد زمني أقرب. في اللحظة 2 = \(t\), ينتهي تنفيذ \(\tau_1\). في اللحظة 5 = \(t\), يبدأ التكرار الثاني للإجراء \(\tau_1\), لكنه لا يقوم بتشغيل إجراء حتى اللحظة 15 = \(t\). في اللحظة 15 = \(t\), يقوم
يشتفع $\tau_2$ لأن حده الزمني (وهو 20) أقرب من الحد الزمني للاجراء $\tau_2$ (وهو 21). يتابع $\tau_2$ تنفيذه بعد انتهاء $\tau_2$ في اللحظة $t=17$.

\section*{E\textsuperscript{2}F 2-5 خصائص الخوارزمية E\textsuperscript{2}F}

إن خوارزمية E\textsuperscript{2}F هي خوارزمية أمثلة في نظام تشععي ذي معالج وحيد. أي أنه إذا وجدت جدولة مقبولة للنظام، عُدفت للخوارزمية E\textsuperscript{2}F في نقطة بالتأكيد.

\section*{E\textsuperscript{2}F و RM 2-5-2 مقارنة الخوارزميتين E\textsuperscript{2}F و RM}

تُقارن خوارزميات جدولة الإجراءات الدورية عادة بمقدار اشغالالم المعالج التي تنتج عنها. فمن المنطقي إبقاء المعالج مشغولاً، كـ RM يمكنه هناك عملًا مفيدًا يقوم به. تحقق حدود الزمنية للإجراءات تتميز خوارزميات الأولوية الديناميكية بأنها أفضل من خوارزميات الأولوية الثابتة حسب هذا المعيار.

تُعتبر الخوارزميات RM أكثر رونفاً من E\textsuperscript{2}F، وتعطي اشغالالم أفضل. لكن عادة ما تكون الجدولة الناتجة عن خوارزميات الأولوية الثابتة ذات تنبؤية أكبر من خوارزميات الأولوية الديناميكية. وفي حال زيادة العبء على النظام (التحميل الزائد)، تتميز RM بأنها أكثر ثباتًا من E\textsuperscript{2}F، إذ أن نفس الإجراء ذو الأولوية المنخفضة الذي يتعدي حده الزمني سيتعدي حده الزمني في المستقبل ولن تتأثر الإجراءات ذات الأولوية المرتفعة. بالمقابل، يصعب معرفة الإجراء الذي سيتعدي حده الزمني عند حدوث التحميل الزائد.

تُتبني أولويته أعلى من E\textsuperscript{2}F تبقى أولويته أعلى من الإجراءات التي لم تتعدي حدها الزمني بعد. إذا استمر هذا الإجراء بالتنفيذ، يمكن أن يؤدي ذلك إلى تعدي الإجراءات الأخرى لحدودها الزمنية. من الأفضل في هذه الحالة أن يترك هذا الإجراء المعالج تلقائيًا بتبديل الحدود الزمنية للإجراءات الأخرى.

على سبيل المثال، بين الجدول التالي مجموعة إجراءات دورية في نظام زمني حقيقي. عند جدولة هذه الإجراءات حسب خوارزمية E\textsuperscript{2}F، ستتعدي كل من الإجراءين $\tau_1$ و $\tau_2$ حدودهما الزمنية إذا سمحت للأجراء الذي تتعدي حده الزمني بالاستمرار بالتنفيذ. بينما سيتعدي $\tau_2$ فقط حده الزمني إذا استعملنا إجراءية RM.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
$\tau_i$ & $r_i$ & $e_i$ & $p_i$ \\
\hline
$\tau_1$ & 0 & 2 & 5 \\
\hline
$\tau_2$ & 0 & 4 & 6 \\
\hline
\end{tabular}
\end{table}
3- مزامنة الإجراءات ومشاركة المصادر

افتراضنا حتى الآن أن إجراءات النظام مستقلة ويمكن أن يجري تشغيلها في أي وقت. هذه الفرضيات غير واقعية عملياً في معظم النظم، إذ تحتاج الإجراءات أن تتصل وتتفاعل مع بعضها البعض. سنقدم في هذا القسم المشاكل الناتجة عن مشاركة المصادر وخوارزميات الجدولة المستعملة في هذه الحالة.

س- المصادر

المصادر هي أية تسهيلات يقدمها النظام للإجراءات لمساعدتها في تأدية عملها. فيما يلي بعض الأمثلة على المصادر المشتركة في النظام:

- الملفات
- التجهيزات المختلفة (طابعات، شاشات، ...)
- المحولات المشتركة في الذاكرة التي غالباً ما تُستخدم للاتصال بين الإجراءات

هناك الكثير من التقنيات البرمجية المستخدمة كمحولات مشتركة في الذاكرة لتسهيل الاتصال بين الإجراءات مثلاً:

- الدارئ buffer
- صندوق البريد mail box
- الأرتال queues

2- الأقسام الحرجة

مهمًا كأن المصدر الذي يحتاج الإجراء للوصول إليه، عادةً ما يحتاج أن يصل إليه وحيداً ودون مقاطعة، ولا يمكن أن تحدث حالات خاصة تجعل نتيجة التنفيذ غير صحيحة. سنبدأ في الورقة التالية كيف يمكن أن يؤدي الوصول إلى مصدر مشترك من قبل إجرائيين في نفس الوقت دون تنسيق (مزامنة) إلى نتائج خاطئة. لنفترض أنه لدينا الإجرائيين التاليين: $\tau_1$ و $\tau_2$:

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $x = i;$</td>
<td>(4) $y = i;$</td>
</tr>
<tr>
<td>(2) $x = x + 1;$</td>
<td>(5) $y = y - 1;$</td>
</tr>
<tr>
<td>(3) $i = x;$</td>
<td>(6) $i = y;$</td>
</tr>
</tbody>
</table>

المتحول $x$ محلي في الإجراء $\tau_1$ والمحول $y$ محلي في الإجراء $\tau_2$. المتحول $i$ هو متحول مشترك بين الإجرائيينقيمته الابتدائية صفر وهو المصدر المشترك الذي نتحدث عنه. إذا فرضنا أن
كلما من الإجراءين ينفذ مرة واحدة، تكون القيمة النهائية الصحيحة في i هي الصفر. لكن لو فرضنا أن t1 بدأ بالتنفيذ وجرى شغل بين التعيينتين (1) و (2)، وانتقل التنفيذ للإجراء 2، ثم عاد التنفيذ للإجراء 1، بعد الانتهاء من التنفيذ الكامل للإجراء 2، عندما تكون القيمة النهائية في i تساوي 1، وهي قيمة خاطئة.

تسمى مجموعة التعليمات التي تتعامل الإجراء فيها مع المصدر المشترك "القسم الحرج". يمكن أن يحتوي إجراء ما على أقسام حريحة لنفس المصدر أو لمصادر مختلفة. إذا دخل إجراءان قسمين حرجين لنفس المصدر وفي نفس الوقت، يؤدي ذلك لحدث أخطاء في نتائج التنفيذ، مما يمكن أن يؤدي لنتائج كارثية في بعض الحالات. لذلك، هناك حاجة كبيرة لتقنيات تمنع حدوث مثل هذا الوضع في معظم نظم الزمن الحقيقي العملية. من هذه التقنيات ذكر "العالم" و"المراقب" و"اختبر وضع" و"semaphore" والعالم بالتخصص لبساطتها وفعاليتها ولكونها موجودة في معظم نظم التشغيل.

3-3 العلامات semaphores

العالم S هو موقع في الذاكرة يعمل كقفل يحمي القسم الحرج المتعلق بمصدر ما. عادةً ما تكون قيمة تمثل "القسم الحرج" (صفرًا أو واحدًا): عندما تكون S=1، يدل ذلك على أن القفل مفتوح ويمكن لأي إجراء الدخول للقسم الحرج. وعندما S=0، يدل ذلك على أن القفل مغلق ويجري حالياً إجراء واحد يعمل في القسم الحرج، ويجب على جميع الإجراءات التي تريد الدخول إليه الانتظار.

يُعرَف على العلام عمليتان أساسيتان هما P(S) (إغلاق وإغلاق) و V(S) (تحرير). فيما يلي خوارزمية عمل كلٍّ منهما:

```
void P(int S) {
    While (S==0);
    S=0;
}

void V(int S) {
    S=1;
}
```

تفحص العملية قيمة العلامة P(S). طالما بقيت قيمةته تساوي 0، يبقى الإجراء يدور في حلقته P(S) (أي يجسد الإجراء الذي استدعى في هذه الحالة). عندما تصبح قيمة S تساوي 1، يخرج الإجراء من حلقته الانتظار ويقفز العلامة (يجعل قيمةته تساوي 0). تقوم العملية V(S) بفتح العلامة بجعل قيمةته تساوي 1.
للتحقيق التنسيق بين الإجراءات كما يلي: قبل الدخول إلى القسم الحرج، يستدعي الإجراء العملية \( P(S) \) وعند الانتهاء منه، يستدعي \( V(S) \) كما هو مبين في الشكل التالي:

\[
\begin{align*}
\text{Process}_1 & : \\
& \vdots \\
& P(S) \\
& \text{critical region} \\
& V(S) \\
& \vdots \\
\text{Process}_2 & : \\
& \vdots \\
& P(S) \\
& \text{critical region} \\
& V(S) \\
& \vdots
\end{align*}
\]

توجد الإشارة إلى أننا نحتاج لعمل مستقل لحماية كل مصدر مشترك. وإذا احتاج قسم حرج ما للوصول إلى عدة مصادر مشتركة، عندما يجب استدعاء العملية \( P \) من أجل كل العلامات الموافقة لهذه المصادر قبل الدخول إليه لحجزها كلها.

غالباً ما تكون العملية \( P \) و \( V \) جزءاً من نواة نظام التشغيل أو استدعاءات لخدمات النظام، لأنها يجب أن تُنفذ دون مقاطعة لكي تعمل بشكل صحيح. سنتبع أهمية هذا الأمر في المثال التالي.

 لنفترض أن التابع \( P \) السابق قد جرى ترجمته إلى لغة الآلة كما يلي:

\[
\begin{align*}
\text{LOAD} & \ R1,S \\
\text{TEST} & \ R1,0 \\
\text{JEQ} & \ @1 \\
\text{STORE} & \ S,0
\end{align*}
\]

نفترض كذلك أنه يوجد إجراءان يريدان الدخول للقسم الحرج ويستدعيان بالتالي التابع \( P \) قبل الدخول. القيمة البدائلية للعلامة \( S \) تساوي 1، مما يعني أن القسم الحرج حر. بدأ الإجراء الأول بتنفيذ تعليمات لغة الآلة السابقة. قبل أن يصل التنفيذ لتعميم STORE، تحدث مقاطعة وينتقل التنفيذ للإجراء الثاني الذي يستدعي نفس التابع السابق. بما أن الإجراء الأول قوطع قبل تخزين القيمة 0 في \( S \)، يجد الإجراء الثاني أن قيمة \( S \) مازالت 1، فيجعلها 0 ويدخل في القسم الحرج. قبل أن يفرغ هذا الإجراء من استخدام المصدر المشترك والخروج من القسم الحرج، تحدث مقاطعة ويعد التنفيذ للإجراء الأول الذي يتابع التنفيذ من تعليمات التي توقف عندها ويدخل في القسم الحرج. بهذا يدخل كلا الإجراءين إلى القسم الحرج مما يمكن أن يؤدي لمشكلة جدية كـ "المنع المتبادل" الذي سنتحدث عنه فيما يلي.

أخيراً، على الرغم من بساطة العلامات، يجب استعمالها بحذر وفهم عملها جيداً، ولا أدى سوء استخدامها لحدث مشاكل منطقية مثل "المنع المتبادل".
3-4- المنع المتبادل

يمكن أن تحدث هذه المشكلة الخطيرة عندما تحاول عدة إجراءات الوصول إلى مصادر على الأقل، مما يؤدي لتوقفها كلها إذا جرى تنفيذها بحالة خاصة سنموتها في المثال التالي. لنفترض أنه لدينا الإجراءين التاليين:

نفترض أننا لدينا الإجراءين التاليين:

<table>
<thead>
<tr>
<th>Task_A</th>
<th>Task_B</th>
</tr>
</thead>
</table>
| ...
| P(S)  |
| use resource 1 |
| ...
| P(R)  |
| stuck here |
| use resource 2 |
| V(R)  |
| V(S)  |
| ...
|                  |
|                  |

<p>|</p>
<table>
<thead>
<tr>
<th>Task_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(S)</td>
</tr>
<tr>
<td>use resource 1</td>
</tr>
<tr>
<td>P(R)</td>
</tr>
<tr>
<td>stuck here</td>
</tr>
<tr>
<td>use resource 2</td>
</tr>
<tr>
<td>V(R)</td>
</tr>
<tr>
<td>V(S)</td>
</tr>
</tbody>
</table>
| ...
|                  |
|                  |

إنه منع المتبادل مشكلة خطيرة جداً لأنها تحدث باحتمال صغير جداً ومن الصعب معرفة أنها موجودة. يأتي اختيار أن تحدث. كذلك فإن حلولها ليست سهلة ولا تنتج عديدة غير مرغوبة، من أهمها إضافة عبء حسابي على النظام يؤدي لاضعاف أدائه. سوف نتطرق إلى إحدى الطرق.

يمكن التعبير عن هذه الحالة بالخطط التالي:

إنه المنع المتبادل مشكلة خطيرة جداً لأنها تحدث باحتمال صغير جداً ومن الصعب معرفة أنها موجودة. يأتي اختيار أن تحدث. كذلك فإن حلولها ليست سهلة ولا تنتج عديدة غير مرغوبة، من أهمها إضافة عبء حسابي على النظام يؤدي لاضعاف أدائه. سوف نتطرق إلى إحدى الطرق.
السيرة للناقل一致 حدوث المنع المتباين والمسمى "خوارزمية المصري". يمكن الاطلاع على حلول المنع المتباين الأخرى في الكتب التي تدرس نظام التشغيل بشكل عام.

3- 5- خوارزمية المصري

تُستخدم هذه الخوارزمية لنفاذي الحالات الغير آمنة التي يمكن أن تؤدي لحدوث المنع المتباين. اقترحها ديجكسترا عام 1968 وهي مشابهة لعمل المصرف في المدن الصغيرة من حيث الحفاظ على مخزون من السيولة النقدية على سبيل الاحتفاظ. يدوع الناس أموالهم في المصرف ويمكنهم سحبها في أي لحظة. لا يحتفظ المصرف بكل الأموال السائلة في حوزته، بل يستثمر 95% منها ويحتفظ بنسبة 5% كسيولة نقدية احتياطية في حال أراد بعض المودعين سحب جزءاً من إيداعاتهم. إذا طلب العديد من الزبائن سحب إيداعاتهم، ينعدم المصرف من تلبية جميع الطلبات. تعتمد خوارزمية المصري في وجود عدة مصادر من نوع واحد فقط، لكن يمكن تمديدها بحيث تعمل مع عدة أنواع من مصادر.

تتآمك الخوارزمية في كل لحظة من أن عدد المصادر المحجوزة من قبل إجراءات النظام زائد عدد المصادر التي يحتاج إليها أحد هذه الإجراءات لإتمام تنفيذها لا يتجاوز العدد الكلي المتاح من المصدر. نقول في هذه الحالة أن النظام في حالة آمنة. وبالعكس، يكون النظام في حالة غير آمنة إذا كان المجموع السابق أكبر تمامًا من العدد الكلي المتاح من المصدر. من الممكن لكن ليس من الضروري أن تؤدي الحالات الغير آمنة إلى منع متباين، لكن من المؤكد أن الحالات الآمنة لن تؤدي أبداً لحدوث منع متباين. لذلك تضمن الخوارزمية بقاء النظام في حالة آمنة وتضمن بالتالي عدم حدوث المنع المتباين.

لناخذ على سبيل المثال نظاماً مكونًا من ثلاثة إجراءات A و B و C ومصدر من نوع واحد (كلة ذارة مثلاً). إذا علمنا أن الإجراء A لن يحتاج لحجز أكثر من 6 مصادر في نفس الوقت ولن يحتاج B لأكثر من 5 ولن يحتاج A بأكثر من 7. يحتاج النظام لبناء جدول مشابه للجدول التالي في كل لحظة لنتبع المصادر المتاحة والمطلوبة في تلك اللحظة.

<table>
<thead>
<tr>
<th>العدد الذي من الممكن أن يحتاج إليه</th>
<th>العدد المصادر المحجوزة حالياً</th>
<th>العدد الأعظمي المطلوب</th>
<th>الإجراء</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>C</td>
</tr>
<tr>
<td>عدد الكلي المتاح حالياً</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

عند كل طلب حجز للمصادر، يجب أن يكون نظام التشغيل هذا الجدول للتلك من أن النظام ما زال في وضع آمن. فيما يلي مثال على وضع آمن للنظام:
الإجراء
العدد الذي من الممكن أن يحتاج إليه
عدد المصادر المحجوزة حالياً
العدد الأقصى المطلوب
الإجراء
أ
6
2
6
A
B
5
3
5
B
C
7
1
7
C

العدد الكلي المتاح حالياً
4

هذه الحالة آمنة لأن العدد الكلي المتاح حالياً (وهو 4) أكبر أو يساوي العدد الذي من الممكن أن يحتاج إليه أحد الإجراءات لإتمام تنفيذه (مثل الإجراء B على سبيل المثال، وكذلك الإجراء A).

فيما يلي مثال على حالة غير آمنة لأن العدد الكلي المتاح لا يكفي أي إجراء، ومن الممكن إذاً أن يحدث منع متبادل:

<table>
<thead>
<tr>
<th>العدد الذي من الممكن أن يحتاج إليه</th>
<th>عدد المصادر المحجوزة حالياً</th>
<th>العدد الأقصى المطلوب</th>
<th>الإجراء</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>7</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

العدد الكلي المتاح حالياً
1

priority inversion -3- انعكاس الأولوية

من أهم نتائج استعمال تقنيات تنظيم الإجراءات كالعالجات وغيرها في نظام الزمن الحقيقي حدوث انعكاس لأولوية إجراءات الزمن الحقيقي، حيث يُفقد الإجراء ذو الأولوية المنخفضة عوضاً عن الإجراء ذو الأولوية المرتفعة فقط لأنه يحجز مصدره يحتاج إليه ذلك الإجراء. هذا الأمر غير مربح فيه نهائياً لأنه يمكن أن يؤدي لأن يتعدى إجراء حساس جداً حده الزمني بسبب إجراء ذي أولوية منخفضة. نبين هذه الحالة في المثال التالي.

لدينا نظام يحتوي على ثلاثة إجراءات كالعالجات وعمليات أخرى:

- أولوية 
  - أولوية 
  - أولوية 
  - أولوية 
  - أولوية 
  - أولوية 

ولكن هناك مصدر مشترك يحتوي على ثلاث إجراءات 

- إجراء A
- إجراء B
- إجراء C

إلى المصدر المشترك. لا يحتاج أي إجراء إلى الوصول إلى المصدر. نفترض أنه حدث تنفيذ الإجراءات حسب المخطط الزمني التالي (القسم الحرج ملتزمن باللصول المتزامن).
ليس فقط، بل لأنه يريد العمل مع المصدر المشترك. يحصل الإجراء في اللحظة $t_2$ يقفل العلام لأنه لا يوجد آخرين يقفلونه، في اللحظة $t_3$ يقفل العلام لأنه يقفل أيضاً العلام مع المصدر المشترك، لكنه يحتاج في حالة انتظار (يصبح مجعداً) لأن العلام محجوز سابقًا من قبل العلام $t_2$.

عندما $t_3$ يقوم بتقفيز العلام، يعود التنفيذ للإجراء $t_4$ لتجنب قسمه الحرج. لكن في اللحظة $t_5$ يقفل العلام لأنه أولوية أعلى من $t_4$ في اللحظة $t_6$ وتحرر $t_7$ لبدء التنفيذ للإجراء $t_5$ في اللحظة $t_7$، يعود التنفيذ للإجراء $t_6$ في اللحظة $t_8$ حتى ينتهي تنفيذه في اللحظة $t_8$.

توصّل لوجود هكذا عدد أكبر من الإجراءات المشابهة للإجراء $t_2$ لأنها أولوية أعلى من $t_3$ وأقل من $t_4$. من الممكن في هذه الحالة أن تزيد هذه الإجراءات من زمن انتظار $t_2$ لفترة طويلة رغم أولويتهم المرتفعة.

نقول بشكل عام أنه حصلت حالة انعكاس أولوية خلال فترة زمنية ما إذا مُنع إجراء ذو أولوية مرتفعة من التنفيذ من قبل إجراءات لها أولوية أقل. حصل هذا الأمر في المثال السابق في المجال الزمني $[t_3, t_6]$ من التنفيذ من قبل $t_2$ و $t_3$.

من الواضح أنه من غير المرغوب به أن يحصل انعكاس لأولوية في نظام الزمن الحقيقي. لذلك، ستتعامل في هذه الحالة خوارزميات مختلفة عن الخوارزميات التي أتيناها سابقاً بهدف التقليل قدر الإمكان من هذه الظاهرة. ستناقش فيما يلي خوارزميتين شهيرتين في هذا المجال هما "وراثة الأولوية" و "قفز الأولوية".
التغير أولويات الإجراءات في هذه الخوارزمية ديناميكياً بحيث يأخذ الإجراء الذي يعمل ضمن القسم الحرج الأولوية العظمى للإجراءات التي تنظف خروجه من القسم الحرج. بكمات أخرى، عندما يجد الإجراء ما \( \tau_1 \) إجراءات أخرى ذات أولوية أعلى، يرث مؤقتاً أعلى أولوية من أولويات هذه الإجراءات إلى أن يخرج من القسم الحرج حيث يستعيد أولويته الأصلية. فيما يلي الخطوات الأساسية للخوارزمية:

- سيدخل أي إجراء مهما كانت أولويته في حالة انتظار إذا حاول حجز علاء محجوز من قبل إجراء آخر.

- إذا لم يجد إجراء آخر من قبل إجراء آخر، يرث إجراء أولوية السائدة، وكانت أولوية أولوية \( \tau_1 \) أعلى من أولوية أولوية \( \tau_2 \) طالما بقي محظوراً. وبعدها ينخفض أولويته لأولوية إجراء آخر.

- وراثة الأولوية متعددة. أي إذا كان إجراءاً أكبر من أولوية إجراءة أخرى أولوية، عندما يرث أولوية إجراءة أخرى أولوية.

وفي المثال الذي ناقشناه في الفقرة السابقة، إذا استعملنا فكرة وراثة الأولوية، ستزداد أولوية إجراءة \( \tau_3 \) عندما يرث إجراءة أولوية إجراءة آخر من أولوية الإجراءة \( \tau_3 \). عندنها، لن ينتج من شفع إجراءة أولوية إجراءة آخر من أولوية إجراءة يتبع مثل المخطط الزمني الناتج عن تطبيق فكرة وراثة الأولوية على المثال السابق.

تنفذ طبيعياً

\[ \tau_1 \]

محمدر

تاقل

وراثة الأولوية

الرجوع للإجراة الأساسية

القسم الحرج

 الزمن

\\( t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \)\\

توجد الإشارة أخيراً إلى أن خوارزمية وراثة الأولوية لا تمنع حدوث المنع المتبادل على عكس خوارزمية "سقف الأولوية" التي ستناقشها في الفقرة التالية.
**خوارزمية سقف الأولوية**

هذه الخوارزمية هي تحسين لخوارزمية وراثة الأولوية، بحيث يُمنع أي إجراء من الدخول إلى القسم الحرج إذا كان هذا سيؤدي إلى تجاوز مهم. يُعطى لكل مصدر أولوية (تُسمى سقف الأولوية) تساوي أعلى أولوية لإجراء سيستعمل هذا المصدر.

خطوات خوارزمية سقف الأولوية هي نفسها خطوات خوارزمية وراثة الأولوية، مع فرق واحد هو أنه يمكن لإجراء $\tau_i$ أن يُمنع من دخول القسم الحرج إذا وجد علامً محسوز من قبل إجراء آخر. فسوف أولوية أكبر أو يساوي أولوية $\tau_i$ سيبين هذا الفرق في المثال التالي. لنحن لدينا نظام مصفوفة من ثلاثة إجراءات $\tau_1$ و $\tau_2$ و $\tau_3$ ويوجد فيه المصدر التالية:

<table>
<thead>
<tr>
<th>المصدر الأولوية</th>
<th>الإجراءات التي تصل إليه</th>
<th>القسم الحرج</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\tau_1)$</td>
<td>$\tau_1, \tau_2$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$P(\tau_1)$</td>
<td>$\tau_1, \tau_2, \tau_3$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>$P(\tau_3)$</td>
<td>$\tau_3$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$P(\tau_2)$</td>
<td>$\tau_2, \tau_3$</td>
<td>$S_4$</td>
</tr>
</tbody>
</table>

إذا فرضنا أن $\tau_2$ يحضر حالياً المصدر $S_2$، وبدأ الإجراء $\tau_1$ بالعمل وطلب حجز المصدر $S_1$، عندئذٍ سيجعل $\tau_1$ وحجز المصدر $S_1$ رغم أنه حرج. إذا كان المصدر $S_1$ أولوية المحرز المجموع $\tau_1$ أولوية أكبر من ألوانه $\tau_2$ و $\tau_3$ و $\tau_2$ و $\tau_1$ و $\tau_2$ و $\tau_3$ أولا بالنسبة $\tau_2$ أكبر من أولا بالنسبة $\tau_3$ وأولوية $\tau_3$ أكبر من الأولوية $\tau_1$ تؤمن هذه الإجراءات بالعمليات التالية:

- $S_1$: حجز $\tau_1$
- $S_1$: حجز $\tau_1$, تحريج $S_1$, تحريج $S_2$
- $S_2$: حجز $\tau_2$
- $S_2$: حجز $\tau_2$, تحريج $S_2$, تحريج $S_3$
- $S_3$: حجز $\tau_3$
- $S_3$: حجز $\tau_3$, تحريج $S_3$, تحريج $S_4$

حسب قاعدة تحديد قيمة سقف الأولوية، تكون قيمة سقف أولوية $P(\tau_1)$ تساوي $S_1$ أو قيمة سقف أولوية $P(\tau_2)$ تساوي $S_2$ أو قيمة سقف أولوية $P(\tau_3)$ تساوي $S_3$.

الحصول على $S_1$
نفترض أن $\tau_3$ يبدأ التنفيذ أولاً ويحجز المصدر في اللحظة $t_1$. ويدخل في القسم الحرج، $S_2$ في اللحظة $t_2$. ويقوم بشغف $\tau_3$ لأن أولويته أعلى يحاول بعد ذلك حجز المصدر الذي يحجزه $\tau_3$ في هذه اللحظة، يُجدَّد $\tau_2$ لأن أولويته ليست أعلى من سقف أولوية $S_2$ الذي يحجزه الإجراء $\tau_3$، ويربط $\tau_3$ بطريقته في اللحظة $t_4$، يصل الإجراء $\tau_1$ ويتتابع.$\tau_3$ يؤدي بالتنفيذ حتى اللحظة $t_5$ حيث يحاول حجز المصدر $S_1$. يُعطى المصدر $S_1$ للإجراء $\tau_1$ لأن أولويته أعلى من أسقف أولويات جميع المصادر المحجوزة في هذه اللحظة (وهي $\tau_3$ فقط المصدر $S_2$ الذي يحجزه $\tau_3$). ينتهي تنفيذ الإجراء $\tau_1$ في اللحظة $t_6$، يعود التنفيذ إلى الإجراء $S_2$ الذي يتابع تنفيذه حتى ينتهي في اللحظة $t_7$، عندما يُسمح للإجراء $\tau_2$ حجز $S_1$ ثم $S_2$ ومتابعة التنفيذ حتى ينتهي في اللحظة $t_8$.

 pantalla 6-3 نتائج خوارزميات التحكم بالوصول للمصادر

يمكن لإجراء $\tau_i$ أن يُجدَّد من قبل إجراء ذي أولوية أقل لمرة واحدة فقط ولفترة لا تزيد عن طول قسم $\tau_i$ حرج واحد. الشرط التالي كاف لاختيار إمكانية جدولة $n$ إجراءات لحجز $S_1$ حسب خوارزمية RM:

$$\sum_{i=1}^{n} \left( \frac{e_i}{p_i} \right) + \max\left( \frac{B_1}{p_1}, \ldots, \frac{B_{n-1}}{p_{n-1}} \right) \leq n\left(2^{\frac{1}{n}} - 1\right)$$

حيث $B_i$ هو زمن التجديد الأعظمي الذي يمكن للإجراء ذي أولوية منخفضة $\tau_i$ هو دور الإجراء $\tau_i$ هو دور الإجراء $\tau_i$. $p_i$ هو دور الإجراء $\tau_i$.
الفصل الثالث: تصميم برامج نظم الزمن الحقيقي

1- هندسة الاحتياجات

إن هندسة الاحتياجات هي جزء من علم هندسة البرمجيات. وهو يهدف إلى تحديد أهداف ومهام وقيود النظام المفترضة على النظام البرمجية، وتمثل هذه المعلومات بأشكال مناسبة للنمذجة والتحليل. هدفها الأساسي إذاً هو إنشاء توصيف كامل وصحيح ومفهوم لكل من الزبون والمو дор لاحتياجات النظام البرمجي. هذا الأمر يحتوي على تناقض بشكل من الأشكال، إذ يجب على هندسة الاحتياجات تقديم توصيف سهل الفهم للزبون لكي يتأكد من أن النظام قيد التطوير يلائم تماماً احتياجاته وتوقعاته، ويجب كذلك أن تحتوي على توصيف كامل للمهام والقيود المفروضة على النظام لكون أساساً يعتمد عليه المطورون. وفي حالة نظم الزمن الحقيقي، يضاف إلى كل ما سبق التقييمات الناتجة عن الحاجة لتمثيل قيد الأداء والزمن.

بين الشكل التالي المخطط العام لتدفق العمل في مرحلة هندسة الاحتياجات من مراحل بناء نظام الزمن الحقيقي. تُتم الفعاليات على شكل مستطيلات ذات حواف ناعمة، بينما تُتم الوثائق الناتجة عن هذه الفعاليات على شكل مستطيلات ذات رمادية.

يدأ إجراء هندسة الاحتياجات بمراحل الدراسة الأولية، وتتضمن القيام ببحث حول دوافع المشروع وطبيعة المشكلة والقيود المفروضة عليها وتحديد نطاق العمل وحدوديات المستقبلية، بالإضافة إلى تحليل مبكر للفوائد الزمنية المفروضة على النظام إذا كان النظام يعمل في الزمن الحقيقي. الوثيقة الأساسية الناتجة عن هذه المرحلة هو تقرير الجدول الذي من الممكن أن يقترح عدم حدود المتابعة في تطوير هذا النظام، وهو أمر لا يحدث في غالبية الأحيان.
ينتقل الإجراء بعدها لمرحلة استخلاص الاحتياجات، وهي تتضمن عمليات جمع الاحتياجات والمتطلبات باستخدام أساليب متعددة مثل مقابلة الزبون وطرح الأسئلة المفصلة عليه وإجراء النماذج الأولية للمشروع. يمكن التعبير عن الاحتياجات المستخلصة من هذه المرحلة باستخدام أساليب مثل مقابلة الزبون وطرح الأسئلة المفصلة عليه وإجراء النماذج الأولية للمشروع. لكن، عادةً ما يُعتبر أنها باستعمال نموذج النطاق، وهو نموذج يتضمن مخططات عديدة مثل مخطط حالات الاستعمال، والمخططات العلاقانية ومخططات السياق.

المتطلب الأولية للمشروع هو استخلاص الاحتياجات، وتتراوح بين النص العادي المكتوب بلغة إنسانية والتوصيف الرياضي المجرد. يُستخدم نموذج النطاق، وهو نموذج يُتضمن مخططات عديدة مثل مخطط حالات الاستعمال، والمخططات العلاقانية ومخططات السياق.

المرحلة التالية هي مرحلة تعريف الاحتياجات. فمن الضروري تعريف كل الاحتياجات المستخلصة بدقة بحيث يمكن تحليلها لاحقاً لتحقيق من صلاحيتها وكفايتها وعدم تناقضها. خرج هذه المرحلة هو وثيقة متطلبات تحتوي على مواصفات الحاجات البرمجية التي تصف مزايا وتصرف النظام النهائي.'

بالإضافة إلى القواعد المفروضة عليه.

2 - أنواع الاحتياجات

على الرغم من وجود العديد من التصنيفات للاحتياجات، فإن التصنيف المتبع عادةً في نظام الزمن الحقيقي هو التصنيف IEEE830 المعرّف من قبل المنظمة IEEE. يُعرف هذا التصنيف أنواع الاحتياجات التالية:

1. احتياجات وظيفية (functional)
2. الواجهات الخارجية (wages)
3. الأداء (context)
4. قاعدة المعطيات المنطقية (data)
5. قيود التصميم (design)
   • مدى التوافق مع المعايير
   • وصفات النظام البرمجي
6. وصفات النظام البرمجي (Reliability)
   • الوثوقية
   • الإنجابية (Reliability)
   • الأمان (Availability)
   • قابلية الصيانة (Security)
   • النقلية (Maintainability)
   • Portability

تسمى الاحتياجات من 2 إلى 6 بالاحتياجات غير وظيفية.
تشمل الاحتياجات الوظيفية توصيفًا لجميع مداخل النظام، بالإضافة إلى سلسلة العمليات الواجب تنفيذها والخروج الموافق لكل قيمة دخل ممكنة، سواء كانت هذه القيمة طبيعية أو غير طبيعية. يمكن أن تشمل الحالات الغير طبيعية توصيفًا لكيفية معالجة الأخطاء ومبادئ حالات تعدي إجراءات الزمن الحقيقي لحدودها الزمنية. بكلمات أخرى، تصف الاحتياجات الوظيفية التصرف الحتمي الكامل لنظام.

أما احتياجات الواجهات الخارجية، فهي توصف جميع مداخل ومخارج النظام وتشمل:

- الاسم
- الهدف
- مصدر الدخل أو وجهة الخرج
- المجالات المقبولة والدقة والسماحية
- وحدة القياس
- التوقيت
- العلاقات مع المداخل والمخارج الأخرى
- أنماط المعطيات
- أنماط الأوامر

تشمل احتياجات الأداء الاحتياجات الديناميكية والساكنة المفروضة على النظام البرمجي أو تفاعل المستخدم مع النظام البرمجي ككل. يمكن أن تشمل الامتيازات الساكنة في نظام الزمن الحقيقي عدد المستخدمين الذين يمكنهم استخدام النظام في نفس الوقت، بينما يمكن أن تشمل الاحتياجات الديناميكية عدد المناقلات وحجم المعطيات التي يمكن معالجتها في فترة زمنية محددة في الحالات العادية وحالات العقبات الزائد المحتمل على النظام.

أما احتياجات قواعد المعطيات المتغيرة فهي تتضمن تعريف أنماط المعطيات المستعملة في الوظائف المختلفة للنظام ومعدل استعمالها وسماحيات الوصول وكائنات المعطيات والعلاقات بينها وشروط تكامل المعطيات.

أخيراً، تتعلق احتياجات قيود التصميم بمدى المطابقة مع المعايير وحدود البيئة المادي المستعمل، بينما تتعلق احتياجات واسعات النظم البرمجية بالوثوقية والإنتاجية والأمان وقابلية الصيانة والقابلية.

3- توصيف احتياجات نظام الزمن الحقيقي

يبدو أنه لا يوجد أسئلة محددة لتوصيف احتياجات نظام الزمن الحقيقي. يستعمل مهندس نظم الزمن الحقيقي عادة أحد الطرق التالية أو خليط منها:

- التحليل من الأعلى للأسفل (top-down analysis)
- structured
يمكن تصنيف هذه الأساليب بشكل عام إلى ثلاثة أصناف: أساليب صورية وشبه صورية وغير صورية.

**الأساليب الصورية**

تعتمد الأساليب الصورية مثل مخطط الحالات المنتهية ، ولغة Z ، على أساس رياضي متين ودقيق، بينما لا يمكن تصميم الأساليب الغير صورية مثل مخططات التدفق بشكل رياضي متين ولا يمكن بالتالي دراستها تحليلياً. كل ما يمكن عمله باستخدام الأساليب غير صورية هو إيجاد مثال معاكس يبين أن أن الشكل من الشكل في تحقيق احتمالاتها أو أن حدث تعارض في النظام. هذه الطرق ليست مناسبة في نظم الزمن الحقيقي حيث تحتاج لتصنيف دقيق وواضح لاحتياجات أداء النظام.

**الأساليب الصورية**

تتمى الأساليب التي لا يمكن تصنيفها على أنها صورية تماماً أو غير صورية تماماً بالأساليب شبه الصورية. يمكن أن تستدعي بعض هذه الأساليب على تصميم رياضي، رغم أنها تبدو كأنها ليست كذلك. على سبيل المثال، يعتبر البعض أن لغة النمذجة الموحدة (UML) هي شبه صورية لأن مخطط الحالة فيها هو صوري لكنه في نمذجة الموحدة يونايتد موديلينج (Unified Modeling Language - UML) يستعمل تقانات نمذجة لها أساس شبه رياضي، بينما يعتبر البعض الآخر أن UML هي ليست حتى شبه صورية لأن توصيفها فيه نمذجات خطيرة وعدم تجانس. لذلك جرى اقتراح تعديل على اسم UML إلى UML2.0 في حيا مكونات لها توصيف صوري نظامي. ونهاك محاولات أخرى لجعلها أكثر صورية.

**الأساليب الصورية**

4- **الطرق الصورية لتصنيف الاحتياجات البرمجية**

تحاول الطرق الصورية تحقيق القدرة التعبيرية للاحتياجات تمديد وتعديل نطق رياضي معرفي مثل منطق الافتراضيات وحساب الإسناديات ونظرية المجموعات ونظرية المجموعات. وهي جدابة ومفيدة لأنها تقدم طريقة علمية ممتعة لتصنيف الاحتياجات، ويؤدي عادة استعمالها لتصنيف الاحتياجات إلى اكتشاف الأخطاء التصميمية مند المراحل الأولى لدورة تطوير البرمجيات. حيث يمكن تصحيحها بسرعة وبأقل كلفة ممكنة.

لكن تتصف الطرق الصورية مع ذلك بأنها صعبة القراءة والاستعمال حتى من قبل الخبراء المدربين. لذلك يتفادى الكثيرون استعمال هذه الطرق لتصنيف الاحتياجات أنظمتهم ويستعملون عوضاً عنها طرقاً أخرى غير صورية أو شبه صورية.

هناك العديد من الطرق الصورية لتصنيف الاحتياجات منها:

- **propositional logic** ومنطق الفرضيات
- **predicate calculus** حساب الإسناديات
سنستعرض فيما يلي نموذجاً "آلة الحالات المنتهية" و "شبكات بتري" الأكثر استعمالاً في تصميم نظم الزمن الحقيقي.

4-1 آلة الحالات المنتهية

وتسمى أيضاً "آلة حالات المنتهية" (FSA) و "مخطط انتقال الحالة" (STD)، وهو نموذج رياضي صوري يستعمل لتوصيف وتصميم مجال واسع من النظم. يمكن التعبير ببساطة عن هذه الآلة بأنها تعتمد على حقيقة أنه يمكن تمثيل معظم النظم باستعمال عدد محدد من الحالات المتميزة. يمكن للنظام أن ينتقل من حالة لأخرى بمرور الزمن أو بعد حدوث حدث معين.

يمكن التعبير عن FSM على شكل مخطط أو جدول. لنفترض على سبيل المثال أننا نريد نمذجة نظام تحكم حاسوبي بطائرات مقاتلة. يحتوي النظام على وحدة قياس للعطالة، ويمكن لنظام التحكم أن يعمل في أحد الحالات الخمس التالية: الإقلاع TAK والملاحة NAV والإياب NAE والهروب/هجوم NAA والهبوط/طيارات NAA. لنفترض أن النظام يتفاعل مع إشارات الواردات من أجزاء أخرى من الطائرة وهي: إسناد المهمة MA وقفل العدو LO واتخاذ الهدف TD وإتمام المهمة MC وهروب العدو EE وتدمر العدو MC. الحالة البدنية للنظام هي TAK والملاحمة المتميزة هي LAN. يبين الشكل التالي مخطط انتقال النظام من حالة لأخرى حسب الإشارات الواردة إليه.

languages
Finite State Machine
Statecharts
Petri nets

Finite State Machine (FSM)
يمكن كذلك التعبير عن نفس المخطط السابق على شكل جدول انتقال حالة، أسطره هي الحالة الحالية، أعمدته هي إشارة الدخل، والقيم المخزنة فيه هي الحالة التالية. بين الجدول التالي تمثل نفس جدول الحالات:

<table>
<thead>
<tr>
<th></th>
<th>MA</th>
<th>LO</th>
<th>TD</th>
<th>MC</th>
<th>EE</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAK</td>
<td>NAV</td>
<td>TAK</td>
<td>TAK</td>
<td>TAK</td>
<td>TAK</td>
<td>TAK</td>
</tr>
<tr>
<td>NAV</td>
<td>NAV</td>
<td>NAE</td>
<td>NAA</td>
<td>LAN</td>
<td>NAV</td>
<td>NAV</td>
</tr>
<tr>
<td>NAE</td>
<td>NAE</td>
<td>NAE</td>
<td>NAE</td>
<td>NAE</td>
<td>NAA</td>
<td>NAE</td>
</tr>
<tr>
<td>NAA</td>
<td>NAA</td>
<td>NAE</td>
<td>NAA</td>
<td>NAA</td>
<td>NAA</td>
<td>NAV</td>
</tr>
<tr>
<td>LAN</td>
<td>LAN</td>
<td>LAN</td>
<td>LAN</td>
<td>LAN</td>
<td>LAN</td>
<td>LAN</td>
</tr>
</tbody>
</table>

كمثال آخر على آلية الحالات المنتهية، نفترض أن أي إجراء في نظام تشغيل الزمن الحقيقي يمكن أن يكون في أحد الحالات الثلاث التالية: مجمد suspended وجاهز ready وقيد التنفيذ executing. يمكن بسهولة استنتاج الأحداث التي تجعل الإجراء ينتقل من حالة لأخرى من المخطط التالي:

 بشمال المصدر متاح

 المحرز

 تجميد

 إختراء المجدول

 مقاطعة

 يصبح المصدر متاح

 المجتمد

 المحرز

 قيد التنفيذ

 تمام

 اختيار المجدول

 المصدر

 FSM

 تتصف بأنها سهلة التطوير ويمكن تحقيقها بسهولة باستعمال جداول الانتقال. وهي كذلك غير غامضة وقدرة على التعبير عن مفاهيم مهمة بأسلوب سهل. كذلك تتتوفر الكثير من الدراسات والخوارزميات التي تمكن المصمم من اختزال لأقصى حد. لكن يمكن أحياناً أن يثير عدد الحالات لقدر كبير يجعل من الصعب قراءتها والتعبير عنها بوضوح.
الشبكات بتري

هي طريقة صورية أخرى تستعمل لتوصيف العمليات التي يجب تأديتها في بيئة متعددة المعالجات أو متعددة الإجراءات. يمكن التعبير عنها بأسلوب رياضي أو بأسلوب بياني. تحتوي شبكة بتري على مجموعة من الدوائر تسمى "الأماكن" وتمثل الإجراءات أو مخازن المعطيات. تمثل المستطيلات العمليات أو الانتقالات. يكتب بجانب كل إجراء عدد المعطيات الموجودة فيه، وابن الجهة كل انتقالات المتعددة المواقيت. يصل بين الدوائر والمستطيلات أضلاع وحيدة الاتجاه.

يُعبر عن الحالة الابتدائية للشبكة بجميع m0 من القيم تمثل أعداد المعطيات الابتدائية في كل إجراء. ينتج عن الحالة m0 حالات أخرى ... m1, m2, ... حالات أخري ... An تؤدي إلى الانتقال إلى الحالات الجديدة. لا تغير شبكة علامات الاتجاه مع مرور الوقت، بل يتغير فقط قيم أعداد المعطيات في كل إجراء. يبين الشكل التالي مثالاً لشبكة بتري بسيطة، ويبيين الجدول الذي يليه جدول القدح المواقيت لها:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before firing</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>After firing</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

يبيين الشكل التالي والجدول الذي يليه مثالاً آخر لشبكة بتري أعقد من السابقة:
يمكن لشبكات بتري أن تستخدم لنموذج النظم وتحليل القيود الزمنية وحالات التسوب race، كما يمكنها نمذجة البنى المعروفة لمخططات التدفق flowcharts كما هو مبين conditions في الأشكال التالية. يبين الشكل (a) كيفية الحصول على التسلسل sequence باستعمال شبكات بتري. أما الشكل (b) فيبين كيفية الحصول على التفرع الشرطي ويعبر الشكل (c) عن حلقة while.
تفيد شبكات بتري كثيراً عند الحاجة لتمثيل النظم المتعددة المعالجات أو المتعددة الإجراءات. هناك أيضاً عدد من التمثيلات التي يمكن تطبيقها على هذا النموذج للحصول على نماذج أكثر قوة مثل شبكات بتري الموقوطة وشبكات بتري الملونة وشبكات بتري الملونة والموقوطة.
إن شبكة بتري هي أداة قوية يمكن استخدامها أثناء تحليل أو تصميم النظام لكشف حالات القفل المتبادل (dead-lock) وحالات التسابق. لنفرض على سبيل المثال أن جزءاً من شبكة بتري لمنظمة ما هو المبين في الشكل التالي:

من الواضح أنه في هذه الحالة لا يمكن معرفة أي الانتقالات سترد. يمكن كذلك استخدام شبكات بتري للكشف الدورات في النظام التي يمكن أن تؤدي للفصل المتبادل كما هو المبين في الشكل التالي:

من الواضح أيضاً أن هذه الحالة هي حالة قفل متبادل. يمكن أخيراً استخدام شبكات بتري للكشف وجود حلقات أكبر وأقل وضوحاً من هذا المثال وتكتم عددًا كبيراً من الإجراءات.
الفصل الرابع: لغات برمجة نظام الزمن الحقيقي

1- مقدمة

يستثمر سوء استعمال لغة البرمجة المستعملة لبرمجة نظام الزمن الحقيقي أثراً كبيراً على أداء النظام كلما، ويسبب بالتالي تعديل العديد من الإجراءات لحدودها الزمنية. كذلك فإن الاستعمال المتزايد لـ Java و Object-Oriented Programming Languages للغات الغرضية التوجه مثل C++ و ADA تجعل المشكلة أكثر حرجاً وصعوبة. لكن لا يمكن إنكار أن لهذه اللغات العالية المستوى مزايا عديدة جعلت مبرمجي نظام الزمن الحقيقي يستعملونها بكثرة عوضاً عن اللغات المنخفضة المستوى مثل C ولغة التجميع .assembly language

إن البرمجة هي أقرب إلى "الحرف" و "الفن" منها إلى الإنتاج الكمي. ومثل أي حرفية، تحتاج البرمجة إلى الأدوات الضرورية والملائمة ذات الجودة العالية، وتحتاج لأن يتقن المبرمج العمل عليها. الأداة الأساسية التي يستعملها المبرمج لتطوير البرمجيات هي المترجم compiler. استعملت العديد من لغات البرمجة لبناء نظم الزمن الحقيقي مثل C و C++ و C# و Java و Forth و Pascal و Fortran وباسكال و ADA95 وADA و Prolog و Lisp و Visual Basic. ولغة التجميع، وحتى ADA95 وباسكال، وبعض هذه اللغات غرضية التوجه مثل C++ و C#, وبعضها إجرائية procedural مثل BASCOM و FORTRAN. تمتلك لغة ADA95 عنصر من الطرفين ويمكن استعمالها كلغة إجرائية أو غرضية التوجه حسب رغبة المبرمج. استعمال بعض اللغات الخاصة مثل Lisp و Prolog في برمجة نظام الزمن الحقيقي غير شائع، وإن كان موجوداً.

السؤال المهم هنا هو: "كيف يمكن معرفة مدى ملاءمة لغة ما لبرمجة نظام الزمن الحقيقي، وما هو المعيار الذي يمكن استعماله لقياس ذلك؟" يمكن الإجابة على هذا السؤال بشكل تقريبي عبر المعايير التجريبية التالية المسمى "معايير كارديلي" Cardelli’s Criteria:

- الاقتصاد في زمن التنفيذ: ما هي السرعة التي يعمل بها البرنامج الناتج؟
- الاقتصاد في زمن الترجمة: ما هو الزمن المقابل لتحويل البرنامج التنفيذي انطلاقاً من البرنامج المصدر؟
- الاقتصاد في التطوير على نطاق صغير: مقدار الجهد الذي يجب على مبرمج وحيد أن يبذل لتطوير النظام?
- الاقتصاد في التطوير على نطاق واسع: مقدار الجهد الذي يجب على مجموعة من المبرمجين أن يبذلوه لتطوير النظام?
- الاصطلاح من ناحية ميزات اللغة: ما هو مقدار صعوبة تعلم ميزات لغة البرمجة؟
لكل لغة برمجة نافذة قوة وضعف عندما يُراد استخدامها لبرمجة نظم الزمن الحقيقي، ويمكن استعمال المعايير السابقة لمقارنتها مع بعضها البعض واختيار الأنسب منها حسب التطبيق المطلوب والمهارات البرمجية المتاحة لدى فريق التطوير.

ستقدم في هذا الفصل عرضاً للغات البرمجة التي يمكن استعمالها لبرمجة نظم الزمن الحقيقي، بالإضافة إلى الأساليب الواجب اتباعها لتحسين البرنامج الناتج قدر الإمكان بحيث يصبح أمثلاً وقابلاً للتنبيز من ناحية الأداء والتصرف.

**Assembly**

على الرغم من انتفاخ مستوى هذه اللغة وعدم احتوائها على العديد من المزايا التي تقدمها اللغات عالية المستوى، تمتلك لغة التجميع بعض المزايا التي تجعلها ملائمة في بعض الأحيان لبرمجة نظم الزمن الحقيقي. فهي تسهم للمبرمج بالتحكم المباشر والمطلق بالبنى صلب للنظام، ويمكن أن يكون البرنامج المكتوب مباشرة بلغة التجميع أكثر أمثلية من البرنامج الناتج عن مترجمات اللغات عالية المستوى. لكن التطور الكبير الذي حصل في مجال تصميم وكتابة المترجمات جعل برامج التجميع الناتج عنها أمثلًا إلى حد كبير بحيث يصعب على معظم المبرمجين كتابة برامج أفضل منه بلغة التجميع مباشرة.

تعاني أيضاً لغة التجميع من أنها غير مهيكلة وذات تجريد ضعيف جدًا، وهي تختلف كثيراً من معالج لآخر والبرنامج الناتج عنها ذو ناقلية صعبة ومجهدة ومليئة بالأخطاء البرمجية. لذلك لا ينصح باستعمالها على الإطلاق. يقتصر استعمالها عادة في الحالات التي لا يقدم فيها مترجم اللغة عالية المستوى بعض التعديلات أو البنية الضرورية لعمل النظام، أو عندما تكون القيود الزمنية المفروضة على النظام حرة جداً بحيث يجب أن يكتب بلغة الآلة ليكون أمثلًا ومثبتًا به تماماً.

كحل وسط، يمكن كتابة الأجزاء الحساسة فقط من النظام بلغة التجميع وكتابة باقي بلغة عالية المستوى. تسمح بعض اللغات عالية المستوى مثل ADA95 وباسكال بحشر أجزاء مكتوبة بلغة التجميع ضمن البرنامج.

إذا أردنا تقييم لغة التجميع حسب معايير كارديلي، يمكن القول أنها اقتصادية جداً في زمن التنفيذ والترجمة (إلا أنها غير مترجمة أصلًا)، لكنها غير اقتصادية في التطور على نطاق ضيق أو واسع، وفقرة من ناحية ميزات اللغة.

**procedural languages**

واللغات التي تُعرف فيما عملها البرامج على شكل لانحة من التعليمات التسلسلية تُتُفق على النتائج. من الأمثلة عليها نذكر: C، بيسكال، فورتран، BASIC، ADA95، Modula-2، BASIC، ولامب. يمكن في جميع هذه اللغات تجميع التعليمات في وحدات منطقية لها مهام محددة تسمى التوابع.
هناك العديد من الميزات التي من المهم توفرها في هذا النوع من اللغات لجعلها مناسبة أكثر لبرمجة نظام الزمن الحقيقي نذكر منها:

- تتبع أساليب تمرير المعاملات
- تسهيلات الحجز الدينيميكية للذاكرة
- التبسيط القوي للمتحولات
- أنماط المعطيات المجردة
- معالجة الاستثناءات (أخطاء زمن التنفيذ)
- modularity

وهي تقابل التوابع في اللغات الإجرائية (تُسمى الطرائق في بعض النواحي العامة، ويسهل إعادة الاستخدام) وتوفر الوحدات البرمجية لتي تُستخدم في توزيع واجهات البرمجية في اللغة إلى إمكانية ترجمة كل وحدة على حدة، مما يجعل اللغات الإجرائية اقتصادية أيضًا في زمن الترجمة ويسهل عملية ترجمة النظم الكبيرة وتوفر أن واجهات interfaces الوحدات ثابتة إلى حد ما.

سنناقش الآن معايير كاردنلي للغات الإجرائية. يؤدي التبسيط القوي للمتحولات إلى تسهيل عملية توأمة برنامج التجميع النتائج، مما يجعل اللغات الإجرائية اقتصادية جداً من ناحية زمن التنفيذ (بفرض أن المتجمد جيد). كذلك يؤدي توفر الوحدات البرمجية في اللغة إلى إمكانية ترجمة كل وحدة على حدة، مما يجعل اللغات الإجرائية اقتصادية أيضًا في زمن الترجمة ويسهل عملية ترجمة النظم الكبيرة.

تتغص هذه اللغات أيضًا بأنها اقتصادية في التطوير على نطاق صغير، لأن عملية التحقق من الأنماط الموجودة في مترجماتها تكشف معظم أخطاء البرمجة وتخفف من جهد الاختبار والتنقيح. كذلك فإن تجريد المعطيات وتوفر الوحدات البرمجية لهما تأثير كبير في تسهيل التطوير على نطاق واسع. يمكن لفريق كبير من المبرمجين الاتصال على واجهات بين الوحدات البرمجية، ثم تطوير كل منها واختبارها بشكل مستقل، شرط تقليل البنية والعمليات العامة على بعضها البعض ما أمكن.

أخيراً، تتصف اللغات الإجرائية بالإعداد من ناحية ميزات اللغة، حيث لا تتصف بالتعقيد ويسهل عادة تعلمها والإلمام بميزاتها المختلفة.

4- اللغات عرضية التوجه

هذه اللغات مزأيا عديدة معروفة ومفيدة جداً مثل الوثوقية وزيادة فعالية المبرمج reliability. إمكانية إعادة الاستعمال reusability. يمكن تعرف هذه اللغات بأنها لغات البرمجة عالية المستوى التي تدعم تجريد المعطيات و sonrasıية inheritance وتوفر الفضيلة للأشكال messaging وإرسال الرسائل POLYMORPHISM. و Support الطريقة Eiffel و ADA95 و C# و Java و C++.

إن الأعراض والمتطلبات وكبسيلتها encapsulation وهي تقابل التوابع في اللغات الإجرائية) في كتابة واحدة سهلة
باستخدام. عادةً ما يكون استعمال الأسلوب الغراسي النوجه لتوصيف نظام ما سهل وطبيعي أكثر من الأساليب الأخرى.

تعدم لغات البرمجة غرسي النوجه الحديثة العديد من التقنيات المهمة في نظام الزمن الحقيقي وكذلك synchronization والمزامنة threads والسلسلة serialization للتحدي في زمن التنفيذ، تتصف هذه اللغات بأنها أسواق من اللغات الإجرائية بطبيعتها وذلك لعدة أسباب أهمها بطء استدعاء الطرائق بسبب مستوى الوصول غير مباشر الإضافي (يحوى كل غرض على جدول indirection) ببعاونين طرائقه يجب الرجوع إليه عند كل استدعاء لأحدها).

فيما يتعلق بالاقتصاد في زمن الترجمة، فإن اللغات غرسي النوجه أسواق من اللغات الإجرائية من هذه الناحية أيضاً. إذ أن مترجماتها معقدة وعادة ما يكون من الصعب تكييف النظم الكبيرة إلى وحدات محددة تماماً عن بعضها البعض، على سبيل المثال، إذا أدخل المبرمج تعديل على صف ما، يجب عندنا إعادة ترجمة كل الصفوف الموروثة منه. لذلك فإنه من الممكن أن يزيد class الزمن اللازم لترجمة النظام بمعدل كبير عندما يصبح حجمه كبيرًا.

بالمقابل فإن اللغات غرسي النوجه أفضل من اللغات الإجرائية من ناحية الاقتصاد في التطوير على نطاق صغير لأنه يمكن للمبرمجين المنفرد خصوصاً من المكتبات الصفوف والبيانات العمل المتوفرة بكثرة لتخفيف جهد وزمن الترجمة إلى حد كبير. مع ذلك، يحتاج المبرمجون في كثير من الأحيان إلى الدخول في تفاصيل مكتبات الصفوف لفهم عملها واستعمالها بفعالية، وهو أمر صعب عادة من الدخول في تفاصيل الوحدات البرمجية الإجرائية.

أما من ناحية الاقتصاد في التطوير على نطاق واسع، فتتفوق فيها اللغات الإجرائية على اللغات غرسي النوجه على الرغم تميز الأخيرة بصفة "إعادة الاستعمال". يعود السبب في ذلك إلى افتقارها للإجتزائية modularity بسبب صعوبة تعديل الصفوف واستعمال الوراثة inheritance على سبيل المثال، يمكن أن يقوم المبرمج عن طريق الخطأ بإعادة تعريف بعض التوابع الموروثة التي يجب عدم إعادة تعريفها لكي يعمل الصفة جيداً. لذلك يضطر المبرمج في كثير من الأحيان إلى الدخول في تفاصيل الصف الإصلي لكي يتمكن من تعديله واتخاذ الوراثة.

أخيراً، تиндف اللغات غرسي النوجه بضغوط الاقتصاد في ميزات اللغة. على سبيل المثال، اعتمد Simula تصميم لغة C++ على نموذج بسيط نسبياً حيوي على لغة، لكنها مليئة بالميزات التي يصعب تعليمها واستعمالها بشكل صحيح. حتى اللغات الحديثة مثل Java، تمت LLC تحديتها جيدة. وما يظل كثير من الناس. لسوء الحظ، تبين أن مبدأ "كل شيء هو غرض object الذي بدأ اقتصادياً"
للمحة عامة عن لغات البرمجة الزمن الحقيقي

ستستعرض فيما يلي عدةً من اللغات المستعملة عادةً لبرمجة نظام الزمن الحقيقي بالترتيب الأبجدي وأسماً. حذفنا من هذا العرض اللغات التالية: ML و Lisp مثل functional script واستعمالها في نظام الزمن الحقيقي، وليس لقلة أهميتها. حذفنا كذلك بعض اللغات الخطاطية مثل Python و Ruby وذلك لأن غير ملائمة عادةً لبرمجة النظام.

ADA95 - لغة 1-5

كان الهدف الأساسي لتطوير لغة ADA هو أن تصبح اللغة الإجابية لتطوير كافة مشاريع إدارة الدفاع الأمريكية والتي تتضمن الكثير من النظم المضافة. لكن النسخة القياسية الأولى التي ظهرت عام 1983 احترمت على الكثير من المشاكل التي أصبحت معروفة للمبرمجين منذ ظهور أول مترجم لها. لذلك قام مطوروها بحل هذه المشاكل لتنظير النسخة الثانية بالاسم 95 والتي تعتبر أول لغة غرضية التوجه قياسية عالمية. لكن إدارة الدفاع وافقت على استعمال لغات أخرى غير مناسبة لتلبية الهدفADA في العديد من المشاريع، مما جعل المطورون يدركون أن لغة ADA الأصلية مناسبة جداً.

ADA95 صُممت لغة ADA خصيصًا لتطبيقات نظام الزمن الحقيقي المضمنة. لكن المطورون وجدوا أنها لغة ثقيلة وغير فعالة لتمثيل هذه التطبيقات، بالإضافة إلى احترامها على العديد من المشاكل عند تحقيق النظم المتعددة المهام باستخدام الأدوات القليلة التي تتيحها اللغة.

أخيراً، على الرغم من أن عدد المطورين الذين يستعملون لغة ADA قليل، توفر نسخة مفتوحة المصدر من لغة ADA على نظام لينوكس أعطاءها دفعًا صغيرًا مؤخرًا.

C - لغة 2-5

أُخْرِجَت لغة C عام 1971 وهي لغة مناسبة للتطبيقات "المنخفضة المستوى". يعود السبب في ذلك إلى أنها مشتقة من لغة BCPL التي كان النمط الوحيد الذي تدعمه هو "كلمة الآلة". تدعم لغة C الأتمام الأساسية منخفضة المستوى مثل البايتات والتبتات والمحارف والعناوين التي يمكن استعمالها للتعامل مع متحدثات المقااطعات وسلسلات المعالج والتجهيزات الأخرى الضرورية في نظم الزمن الحقيقي. لذلك controllers غالباً ما يستعمل لغة C ككلمة تجميع عالية المستوى مستقلة عن البنية المادية.

لغة C تحتوي على العديد من أنماط المتحولات التي تسمح بالتحكم بتوليد الترميز النهائي من مستوي عال مثل compiler و constant و static و volatile و register. في هذا البديل، إذا عُرِّف register منحول بالنماذج.register، مما يجعلها نسخة من سجلات المعالج لهذا المتحول مما يجعل نتائج الترمجة أسرع وأصغر حجماً.
هكذا، تخبر الصفة المترجم بأن لا يحاول أمثلة volatile المتحولات المعرّفة بهذه الصفة. هذا الأمر مفيد على سبيل المثال للتعامل مع الدخل/خراج المعنون بالذاكرة.

يشكل عام، تعتبر لغة C مناسبة للبرمجة المضمنة لأنها لغة مهيئة ومرنة وخالية من القيود.

3- لغة C++

لغة C++ هي لغة برمجة متوجة هجينة (أي أنها لغة إجرائية أيضا) بدأ كطائفة ماكروية مبنية C على لغة C، لكنها أصبحت اليوم لغة مستقلة لها متجهاتها الخاصة المتواقة مع C. تمتلك جميع مزايا اللغات غرضية متوجة وتؤمن كبسولة وجريداً أفضل من C.

يتتفق معظم المبرمجون أن إساءة استخدام المؤشرات هو المصدر الأساسي للأخطاء البرمجية في الأوقات عمليات حسابية معقدة على المؤشرات لإنشاء C/C++ فقد استعمل مبرمجو ++ للأعمال كمية معقدة على المؤشرات لإنشاء ++. لذلك، كانوا يفسرون وقنا طويلاً لاصطياد أخطاء برمجية معقدة ناتجة عن معالجة بسيطة لسلسل المحارف. حالياً، تحتوي Standard Template Leben على العديد من المكتبات النظامية لبئرة المعطيات الديناميكية مثل ++، والتي تحتوي على الأنماط ++ والخ. هذه火热 البرامج من عناية لغة ++ التعامل مع سلاسل المحارف.

يتزايد كل يوم عدد النظم المضمنة المبرمجة باستعمال ++ ويسأل المبرمجون أنفسهم كل يوم: هل استعمل C أم C++ لتحقيق نظامي مضمن؟ الجواب هو: الأمر يعتمد على ما يريد المبرمج. اختيار C بدلاً من ++ يعني الحصول على برنامج سريع ويسهل التنبيز بتصرفاته، لكنه صعب الصيانة. وبالعكس، فإن البرامج المكتوبة بلغة ++ أبطأ ويصعب التنبيز بتصرفاتها، لكن صيانتها أسهل بكثير من البرامج المكتوبة بلغة C.

4- لغة C#

Java Virtual Machine (JVM) وهي لغة شبيهة بلغة Java ولها بيئة تشغيل تشبه كثيراً آلة جافا الافتراضية التي تحتوي على عكس جافا، تسمى "منصة عمل آمنة". لكنها لا تثقافة "غير "unsafe code " أي يمكن فيها التعامل مع المؤشرات مباشرة وإجراء العمليات عليها كما هو للغة ++.﻿

5- لغة C
garbage collector أسباب، منها زمن التنفيذ الغير محدود لجامع النفايات حتمية. من الممكن أن بيئة التشغيل، وعدم توفير البيئة اللازمة لجعل جدولة التناسب كالتي نحتاجها لاستدعاء خدمات نظام threads # تعطي نظم الزمن الحقيقي الرخوة والlaşma وذلك لاستدعاء خدمات نظام التشغيل, بالإضافة operating system API التشغيل إلى أن أداء عمليات الفاصلة العائمة فيها يقارب أداء لغة C.
تُعتبر لغة فورتران من أقدم اللغات عالية المستوى المستعملة لبرمجة نظم الزمن الحقيقي. وسبب افتقد النسخ الأولى من فورتران للعديد النظم المكتبية بها على أجزاء كبيرة مكتوبة بلغة التجميع لمعالجة المقاطعات وجدولة الإجراءات. أما الاتصال مع الأجهزة الخارجية، فكان يجري باستعمال DMA والدخل/خرج المعنون بالذاكرة.

حيث أن النسخ الحديثة من فورتران على دعم Memory-mapped I/O للعديد. رغم ذلك، مازالت نظم الزمن الحقيقي المكتبية بها تحتوي على أجزاء مكتوبة بلغة التجميع.

بطورت فورتران في وقت كان من الضروري فيه الحصول على أداء أمثل على أجهزة حاسوب صغيرة الحجم وبطيئة. ونتيجة لذلك، كانت البنية الأساسية لغة مفتقة عدة هو الأداء، وهو weakly-typed ما كان محققا بالفعل في أغلب الأحيان. تتفوق معاها بأنها ضعيفة التنميط وتفتق لتغطية أخطاء زمن التنفيذ وأنماط المعطيات المجردة. لكنها ما زالت تُستخدم حتى الآن في العديد من نظم الزمن الحقيقي الحديثة.

لغة جافا

جافا هي لغة مفسرة. يحول مترجم جافا البرنامج المصدر إلى لغة آلة وسيلة مستقلة عن أي مترجم Java Virtual Machine JVM اسمها byte code، ويقوم مفسر خاص هو آلة جافا الافتراضية 

بتنفيذ هذه التعليمات في بيئة مدارة managed بمتلئ هذه التعليمات في بيئة مدارة JVM التعليمات الناتجة عن الترجمة

مصدر مفسر آلة افتراضية حاسوب

ميزة هذا الأسلوب هو الناقلية portability حيث يمكن لبرنامج جافا المترجم أن يعمل على آية آلة تتوفر عليها آلة جافا الافتراضية. هناك العديد من التطبيقات الهامة لهذا الأسلوب في النظام المصمم والمحمول، مثل الهواتف الخلوية والبطاقات الذكية والويب.

يتوفر أيضًا متترجمات حقيقية للغة جافا ينتج عنها ترميز ثنائي يعمل على نوع خاص من المعالجات كغيرها من اللغات المترجمة الأخرى مثل C++. يتوفر كذلك معالجات قادرة على تنفيذ تعليمات مباشرة byte code.
لأولية للبرنامج وير نسخة من جافا مناسبة لتطبيقات الزمن في نتائج سطحية وحجزها
ديناميكيا، وتقع مسؤولية تحريرها على كاهال جزء خاص من المفسر هو جامع النفايات.
كذلك فإن المفسر يخبر معظم الحالات التي يمكن أن تولد أخطاءً في زمن التنفيذ مثل خروج دليل مصفوفة خارج المجال الصحيح أو التقسيم على صفر. كل هذا يجعل
أداء لغة جافا أسوأً من أداء اللغات المترجمة الأخرى مثل ++C.

**real-time java** لـ "لغة

مشكلة لغة جافا الأصلية هو الآداء الذي لا يمكن التنويه به للبرنامج بسبب جامع النفايات لأن
المواصفات الأصلية للغة تحدد خطوطاً عريضة فقط للجدولة. على سبيل المثال، عند تنافس عدة
نياسب الوصول إلى مصادر مشتركة، تُثني نسبة النياسب ذات الألوان العالية أخرى على حساب
نياسب ذات الألوان المنخفضة. لكن هذا لا يضمن أن تبقى النياسب ذات الألوان العالية في
حالة تنفيذ، كما لا يمكن الاعتماد على أولويات النياسب لتحقيق المنع المتبادل بشكل موثوق. كل هذا
جعل المطورين يعلمون أن لغة جافا القياسية غير مناسبة لتطبيقات الزمن الحقيقي.

National Institut of Standards and Technology (NIST) للتغلب على هذه المشاكل، كُلف المعهد الوطني للمعايير والتكنولوجيا
بتطوير نسخة من جافا مناسبة لتطبيقات الزمن الحقيقى والنظم المضمنة. نشر تقرير مجموعة العمل المكلفة بهذا الأمر في شهر أيلول عام 1999 وعرف تسمية احتياجات أساسية لمواصفات نسخة الزمن الحقيقي من لغة جافا -

وهي: Time Specification for Java (RTSJ)

**profilers** يجب أن تتضمن المواصفات إطار عمل للاكتشاف والبحث عن أية محلات
متاحة.

1. يجب أن يكون جامع النفايات ذو تأخير شفيعي محدود.
2. يجب أن تكون جامع النفايات ذات النسخة الحقيقية بنفس مستوى التفصيل
المتاح حالياً في وثائق المعايير الأخرى.
3. يجب أن تتضمن المواصفات واجبات API لاتصال ومزامنة الإجراءات المكتوبة بلغة
جافا مع الإجراءات المكتوبة باللغات الأخرى.
4. يجب أن تتضمن المواصفات أساليب معاينة الأحداث الداخلية والخارجية الغير متزامنة.
5. يجب أن تتضمن المواصفات إمكانية إنهاء الغير متزامن للنيابة.
6. يجب أن تتضمن المواصفات أساليب لتحقيق المنع المتبادل دون تجميد.
7. يجب أن تتضمن المواصفات أساليب توحيد النسخة المرادفة من معرفة هنالك يجري تنفيذها
في نسبي زمن حقيقي أم في نسبي عادي.
8. يجب أن تضمن المواصفات العلاقات بين نسبي الزمن الحقيقي والنيابة العادية.
9. يجب أن تعزّز المواصفات العلاقات بين نسبي الزمن الحقيقي والنيابة العادية.
تعزّز المواصفات RT، صفاً RTSJ، بجدول الزمن الحقيقي، ويقوم مجدول مقيم خاص الوصول إلى الأغراض في الكومة، لذلك هي عرضة للتأخيرات الزمنية التي تسببها جامع النفايات.

فيما يتعلق بجميع النفايات، تُعرّف RTSJ نموذج ذاكرة موسع يدير الذكاء بطرق تأثير على إمكانية نيايب الزمن الحقيقي لتحقيق التنفيذ الحتمي. يسمح هذا النموذج لكل من الأغراض ذات الأعمار القصيرة والطويلة بالتواجد خارج منطقة الكومة التي تتحكم بها جامع النفايات. وهو من رؤية كيفية بحيث يسمح بحلول معروفة لإدارة الذاكرة بفعالية مثل حوض الأغراض pre-allocated object pool.

يقوم النظام بتخزين جميع النفايات التي تنتظر إتمال مصدر ما في رئل أولوتي (أي مرتبة في الرئل حسب أولويتاتها). تشمل هذه المصادر المعالج ومناطق الذاكرة المشتركة. يقدم للبرمج صفح يمكنه من الوصول المباشر للذاكرة الفيزيائية. الأول هو RawMemoryAccess الفيزيائية، والوصول إليها على مستوى word و byte. الصف الثاني هو PhysicalMemoryArea الذي يسمح بالوصول على أغراض من الأغراض في PhysicalMemory التي يمكن للنظام أن يحجز الأغراض فيها.

Communicating

تعتبر لغة Occam 2 على نموذج "الإجراءات التسلسليه المتخصصة". الوحدة الأساسية فيها هي "الإجراء" الذي يوجد منه أربعة أنواع أساسية: الإسقاط، الدخول، الخروج والانتظار. تُبنى الإجراءات الأعدادي بتركيب تسلسلي أو تفريعي من هذه الإجراءات. البنية التحتية الأساسية هي IF التي تحوي قائمة من Shroid وإجراءات مواجهة لها.

ستُستخدم لغة Occam 2 لأتمّك إجراءات التفريعي على الأجهزة التفريعي من نوع transputers، لكن يوجد لها أيضاً مترجمات للبنية الأخرى. أخيراً، استُعملت هذه اللغة في العديد من التطبيقات في بريطانيا خصوصاً.

لغات الزمن الحقيقي المتخصصة

ظهرت واختفت العديد من لغات الزمن الحقيقي المتخصصة على مدى الثلاثين سنة الماضية. تقدم فيما يلي بعضها منها:
لغة PEARL: طُورت هذه اللغة من قبل مجموعة من الباحثين الألمان في بدايات السبعينيات، ولها تطبيقات واسعة خصوصاً في مجال الأتمتة الصناعية في ألمانيا.
لغة Real-Time Euclid: وهي لغة تجريبية تتمتع بكونها من اللغات القليلة المناسبة تماماً لتحليل قابلية الجدولة scheduability analysis. للأسف، لم تجد هذه اللغة طريقة لتحقيق تطبيقات عملية في مجال نظام الزمن الحقيقي.
لغة Real-Time C: وهو اسم عام للعديد من التمديدات الماكروية للغة C التي تقدم بها لангة C الخاصة للتوقيت والتحكم غير متوفرة في لغة C الأصلية.
لغة Real-Time C++: وهو اسم عام للعديد من المكتبات التي تقدم امكانيات متقدمة للتقوية والتحكم. للغة C++, وهي تقدم امكانيات متقدمة للتوقيت والتحكم.