Advanced Operating System

Lecture Notes

Dr. Professor, J.M. Khalifeh

قسم المعلوماتية

الوحدة الثالثة

Unit-3

Deadlock

ملخص

قد تتنافس عدة مسالك في بيئة متعددة البرمجة على عدد محدود من الموارد. إذا لم تكن الموارد متاحة حين يطلبها المسلك فإنه يدخل في حالة انتظار. وهنا قد لا يستطيع المسلك في بعض الأحيان، تغيير حالته مرة أخرى أبدًا، لأن الموارد التي طلبها يتم الاحتفاظ بها من قبل مسالك انتظار أخرى. وهذا يسمى الجمود Deadlock. تطرقنا إلى هذه المشكلة سابقا بإيجاز سابقا كشكل من أشكال فشل النظام. هناك، وحينها عرفنا الجمود بأنه موقف تنتظر فيه كل عملية في مجموعة من العمليات حدثًا لا يمكن أن يتسبب فيه إلا عملية أخرى في المجموعة.

نصف في هذا الفصل، الأساليب التي يمكن لمطوري التطبيقات وكذلك مبرمجي أنظمة التشغيل استخدامها لمنع الجمود أو التعامل معه. على الرغم من أن بعض التطبيقات يمكنها التعرف على البرامج التي قد تصل إلى حالة الجمود، فإن أنظمة التشغيل لا توفر عادةً وسائل منع حدوثه، ويظل من مسؤولية المبرمجين التأكد من تصميم برامج خالية منه. وقد أصبحت مشاكل الجمود أكثر تحديًا مع استمرار الطلب على زيادة التزامن والتوازي في أنظمة متعددة النواة.

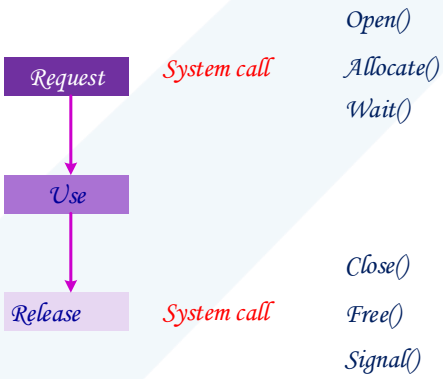
أهداف الوحدة

- إيضاح كيفية حدوث الجمود عند استخدام أقفال المزامنة المتبادلة.
- معرفة الشروط الأربعة الضرورية التي تميز الجمود.
- تقييم الطرق الأربعة المختلفة لمنع الجمود.
- تطبيق خوارزمية المصرفي لتجنب الجمود.
- تطبيق خوارزمية اكتشاف الجمود.
- تقييم الطرق للتعافي من الجمود.

Introduction of Deadlock in Operating System مدخل إلى الجمود في نظم التشغيل

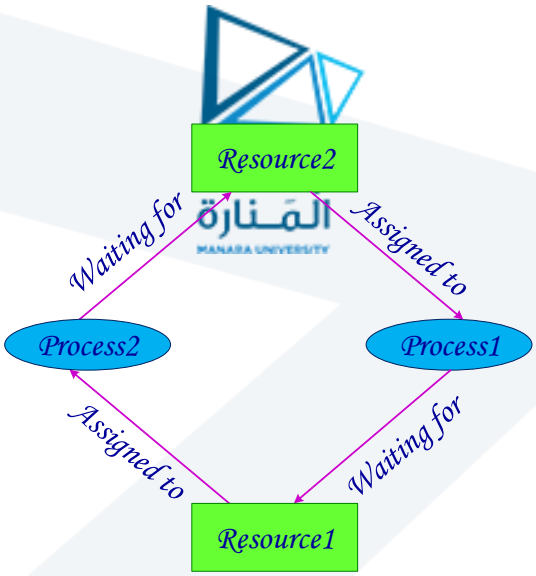
قبل الخوض في التفاصيل حول كيفية حدوث الجمود في نظام التشغيل، دعنا أولاً نناقش كيف يستخدم نظام التشغيل الموارد الموجودة. تستخدم العملية في نظام التشغيل الموارد بالطريقة التالية في الشكل 1.

- تطلب العملية موردًا ما
- تستخدم المورد
- تحرر المورد



الشكل1: كيفية استخدام الموارد

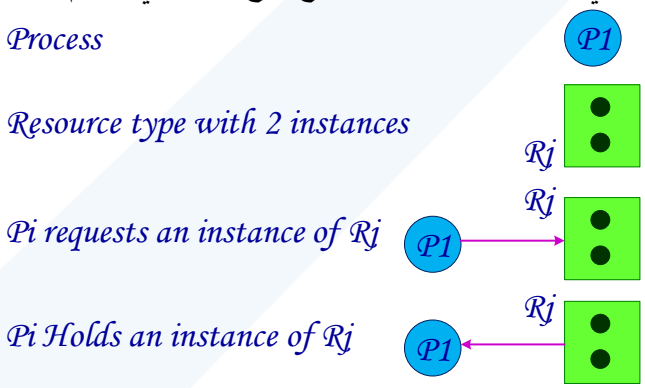
يحدث موقف في أنظمة التشغيل عندما يكون هناك عمليتان أو أكثر تحتفظان ببعض الموارد وتنتظران الموارد التي تحتفظ بها عمليات أخرى. على سبيل المثال، في الشكل 2، تحتفظ العملية 1 بالمورد 1 وتنتظر المورد 2 الذي حصلت عليه العملية 2، وتنتظر العملية 2 المورد 1.



الشكل2: حدوث الجمود

الرسم البياني لتخصيص الموارد Resource-Allocation Graph

يمكن وصف حالات الجمود بدقة أكبر من حيث الرسم البياني الموجه المسمى الرسم البياني لتخصيص موارد النظام. يتكون هذا الرسم البياني كما في الشكل 3، من مجموعة من الرؤوس V ومجموعة من الحواف E. يتم تقسيم مجموعة الرؤوس V إلى نوعين مختلفين من العقد: T = {T1, T2, …, Tn}، وهي المجموعة المكونة من جميع المسالك النشطة في النظام، و{R = {R1, R2, …, Rm، وهي المجموعة المكونة من جميع أنواع الموارد في النظام.

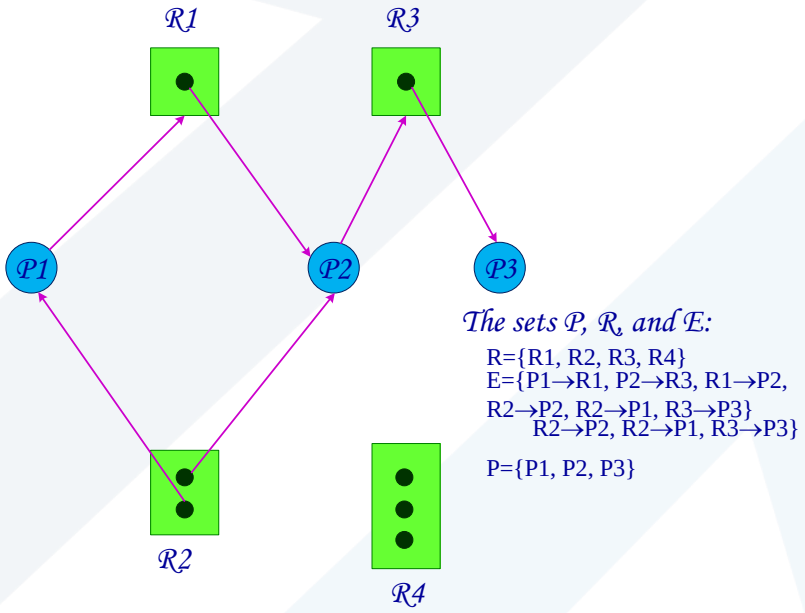


الشكل3 : الرموز المستخدمة في الرسم البياني

يتم الإشارة إلى الحافة الموجهة من المسلك Ti إلى نوع المورد Rj بواسطة Ti → Rj؛ وهذا يدل على أن المسلك Ti قد طلب مثيلًا من نوع المورد Rj وينتظر حاليًا هذا المورد. يتم الإشارة إلى الحافة الموجهة من نوع المورد Rj إلى المسلك Ti بواسطة Rj → Ti؛ وهذا يدل على أنه تم تخصيص مثيل من نوع المورد R j للمسلك Ti. تسمى الحافة الموجهة Ti → Rj حافة الطلب؛ تسمى الحافة الموجهة Rj → Ti حافة التعيين. على سبيل المثال، نمثل كل مسلك Ti على هيئة دائرة وكل نوع مورد R j على هيئة مستطيل. وكمثال بسيط، يوضح الرسم البياني لتخصيص الموارد الموضح في الشكل 4، حالة الجمود. ونظرًا لأن نوع المورد Rj قد يكون له أكثر من مثيل، فإننا نمثل كل مثيل من هذا القبيل على هيئة نقطة داخل المستطيل. لاحظ أن حافة الطلب تشير فقط إلى المستطيل Rj، في حين يجب أن تشير حافة التعيين أيضًا إلى إحدى النقاط في المستطيل. عندما يطلب المسلك Ti مثيلًا من نوع المورد Rj، يتم إدراج حافة الطلب في الرسم البياني لتخصيص الموارد. وعندما يمكن تلبية هذا

الطلب، يتم تحويل حافة الطلب على الفور إلى حافة تعيين. وعندما لا يحتاج المسلك بعد الآن إلى الوصول إلى المورد، فإنه يطلق سراح المورد. ونتيجة لذلك، يتم حذف حافة التعيين.

يصور الرسم البياني لتخصيص الموارد الموضح في الشكل 4 الموقف التالي.



الشكل 4:وجود حلقة مع الجمود

المجموعات **T** *و* **R** *و* **E** *هي:*

- $T = \{T_1, T_2, T_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{T_1 \rightarrow R_1, T_2 \rightarrow R_3, R_1 \rightarrow T_2, R_2 \rightarrow T_2, R_2 \rightarrow T_1, R_3 \rightarrow T_3\}$

مثيلات الموارد هي:

- مثيل واحد من المورد R1
- مثيلان من المورد R2
- مثيل من المورد R3
- ثلاثة من المورد R4

حالات المسالك:

- المسلك T1 يمسك مثيلا من المورد R2 وينتظر مثيلا من المورد R1
- المسلك T2 يمسك مثيلا من المورد R2 ومثيلا من المورد R1 وينتظر مثيلا من المورد R2
- المسلك T3 يمسك مثيلا من المورد R3, ولا ينتظر أي مثيل

بالنظر إلى تعريف الرسم البياني لتخصيص الموارد، يمكن إظهار أنه:

- إذا لم يحتوي الرسم البياني على حلقات، فلا يوجد أي مسلك في النظام في حالة جمود.
- إذا كان الرسم البياني يحتوي على حلقة، فقد يوجد جمود. في هذه الحالة:

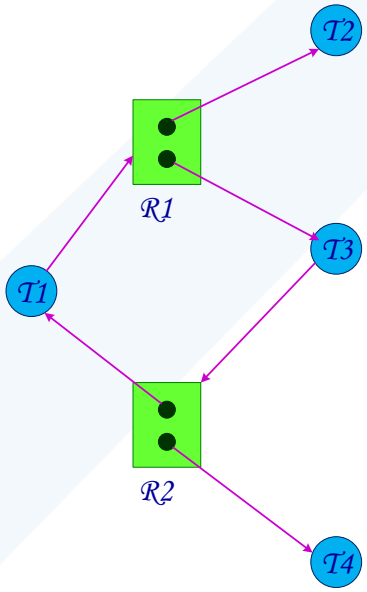
1. إذا كانت الحلقة تحوي فقط مجموعة من أنواع الموارد، ولكل منها مثيل واحد فقط، فإن الجمود قد حدث. كل مسلك مشارك في الحلقة في حالة جمود. في هذه الحالة، تكون الحلقة في الرسم البياني شرطًا ضروريًا وكافًا لوجود الجمود.
2. إذا كان لكل نوع من الموارد عدة مثيلات، فإن الحلقة لا تعني بالضرورة حدوث جمود. في هذه الحالة، تكون الحلقة في الرسم البياني شرطًا ضروريًا ولكنه ليس كافيًا لوجود الجمود.

لتوضيح هذا المفهوم، نعود إلى الرسم البياني لتخصيص الموارد الموضح في الشكل4. افترض أن المسلك T3 يطلب مثيلًا من نوع المورد R2 . نظرًا لعدم توفر مثيل مورد حاليًا، نضيف حافة طلب T3 → R2 إلى الرسم البياني الشكل 4. في هذه المرحلة، توجد دورتان في الحد الأدنى في النظام:

$$T_1 \rightarrow R_1 \rightarrow T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$$
$$T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2 \rightarrow T_2$$

المسالك T1 و T2وT3في حالة جمود. ينتظر المسلك T2 المورد R3 ، الذي يحتفظ به المسلك T3. ينتظر المسلك T3إما المسلك T1 أو المسلك T2 لتحرير المورد R2 . بالإضافة إلى ذلك، ينتظر المسلك T1 المسلك T2 لتحرير المورد R1. الآن، بالنظر إلى الرسم البياني لتخصيص الموارد في الشكل 5. في هذا المثال، لدينا أيضًا حلقة:

$$T_1 \rightarrow R_1 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$$



الشكل 5: وجود حلقة دون جمود

ومع ذلك، لا يوجد جمود. لاحظ أن المسلك T4 قد يحرر مثيله من نوع المورد R2. يمكن بعد ذلك تخصيص هذا المورد إلى T3، مما يؤدي إلى كسر الحلقة. باختصار، إذا لم يكن للرسم البياني لتخصيص الموارد حلقة، فهذا يعني أن النظام ليس في حالة جمود. إذا كانت هناك حلقة، فقد يكون النظام في حالة جمود أو لا يكون كذلك. وهذه الملاحظة مهمة عندما نتعامل مع مشكلة الجمود.

طرق معالجة Methods for Handling Deadlocks

بشكل عام، يمكننا التعامل مع مشكلة الجمود بإحدى الطرق الثلاث التالية:

- يمكننا تجاهل المشكلة تمامًا والتظاهر بأن الجمود لا يحدث أبدًا في النظام.

- يمكننا استخدام بروتوكول لمنع أو تجنب الجمود، مما يضمن عدم دخول النظام في حالة جمود أبدًا.
- يمكننا السماح للنظام بالدخول في حالة جمود واكتشافها والتعافي منها.

الحل الأول هو الحل الذي تستخدمه معظم أنظمة التشغيل، بما في ذلك Linux وWindows. ثم يعود الأمر إلى مطوري النواة والتطبيقات لكتابة برامج تتعامل مع الجمود، باستخدام الأساليب الموضحة في الحل الثاني عادةً. تتبنى بعض الأنظمة – مثل قواعد البيانات – الحل الثالث، مما يسمح بحدوث الجمود ثم إدارة التعافي.

لضمان عدم حدوث الجمود مطلقًا، يمكن للنظام استخدام إما مخطط منع الجمود أو مخطط إبطال الجمود.

منع حدوث الجمود Deadlock Prevention

هناك أربعة شروط ضرورية لكي يحدث الجمود. من خلال التأكد من عدم تحقق شرط واحد على الأقل من هذه الشروط، يمكننا منع حدوث الجمود. سنتناول هذا النهج بالتفصيل من خلال فحص كل من الشروط الأربعة الضرورية بشكل منفصل.

الاستبعاد المتبادل Mutual Exclusion

يجب أن يتحقق شرط الاستبعاد المتبادل. أي أنه يجب أن يكون هناك مورد واحد على الأقل غير قابل للمشاركة. لا تتطلب الموارد القابلة للمشاركة وصولاً متبادلاً حصريًا وبالتالي لا يمكن إشراكها في حالة الجمود . تعد الملفات للقراءة فقط مثالاً جيدًا على الموارد القابلة للمشاركة. إذا حاولت عدة مسالك فتح ملف للقراءة فقط في نفس الوقت، فيمكن منحها وصولاً متزامنًا إلى الملف. لا يحتاج المسلك أبدًا إلى انتظار مورد قابل للمشاركة. ومع ذلك، بشكل عام، لا يمكننا منع حالات الجمود عن طريق رفض شرط الاستبعاد المتبادل فقط، لأن بعض الموارد غير قابلة للمشاركة بطبيعتها. على سبيل المثال، لا يمكن مشاركة قفل mutex في وقت واحد بواسطة عدة مسالك.

الاحتفاظ والانتظار Hold and Wait

لضمان عدم حدوث حالة الاحتفاظ والانتظار في النظام، يجب أن نضمن أنه عندما يطلب مسلك موردًا، فإنه لا يحتفظ بأي موارد أخرى. يتطلب أحد البروتوكولات التي يمكننا استخدامها أن يطلب كل مسلك ويخصص له جميع موارده قبل أن يبدأ التنفيذ. وهذا بالطبع غير عملي بالنسبة لمعظم التطبيقات بسبب الطبيعة الديناميكية لطلب الموارد.

يسمح بروتوكول آخر للمسلك بطلب الموارد فقط عندما لا يكون لديه أي موارد. قد يطلب المسلك بعض الموارد ويستخدمها. ولكن قبل أن يتمكن من طلب أي موارد إضافية، يجب عليه تحرير جميع الموارد المخصصة له حاليًا.

كل من هذين البروتوكولين له عيبان رئيسيان.

أولاً، قد يكون استخدام الموارد منخفضًا، حيث قد يتم تخصيص الموارد ولكن لا يتم استخدامها لفترة طويلة. على سبيل المثال، قد يتم تخصيص قفل mutex لمسلك لتنفيذه بالكامل، ومع ذلك يتطلبه لفترة قصيرة فقط.

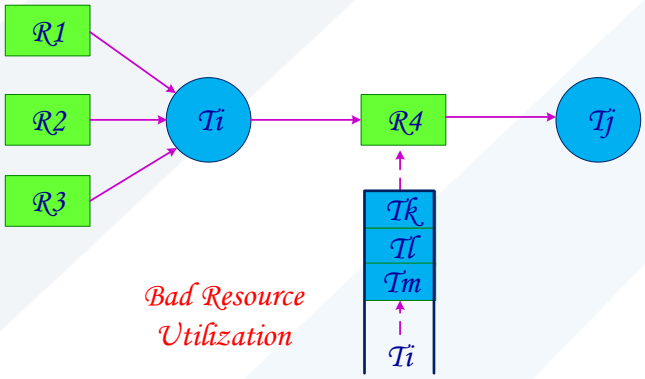
ثانيًا، إمكانية حدوث التجويع. قد يتعين على المسلك الذي يحتاج إلى العديد من الموارد الشائعة الانتظار إلى أجل غير مسمى، لأنه يتم تخصيص مورد واحد على الأقل يحتاجه دائمًا لمسلك آخر.

عدم وجود الاستيلاء No Preemption

الشرط الثالث الضروري لحدوث الجمود هو عدم وجود استيلاء للموارد التي تم تخصيصها بالفعل. للتأكد من عدم حدوث هذا الشرط، يمكننا استخدام البروتوكول التالي. إذا كان أحد المسالك يحتفظ ببعض الموارد ويطلب موردًا آخر، في الشكل 6 لدينا المسلك Ti يطلب المورد R4 ولا يمكن تخصيصه له على الفور أي يجب على المسلك الانتظار لأنه مخصص للمسلك

Tj، (كما يمكن أن يكون هناك عدة مسالك تنتظر تحرير المورد R4 لتستخدمه. من الأفضل أن يتم إلغاء تخصيص جميع الموارد التي يحتفظ بها المسلك Ti حاليًا حيث أنها قد تكون مطلوبة من مسالك أخرى قيد الانتظار. بعبارة أخرى، يتم تحرير هذه الموارد ضمناً. تتم إضافة الموارد التي تم إلغاء الأولوية لها إلى قائمة الموارد التي ينتظرها المسلك. سيتم إعادة تشغيل المسلك فقط عندما يتمكن من استعادة موارده القديمة، بالإضافة إلى الموارد الجديدة التي يطلبها.

بدلاً من ذلك، إذا طلب المسلك بعض الموارد، فإننا نتحقق أولاً مما إذا كانت متاحة. إذا كانت متاحة، فإننا نخصصها. إذا لم تكن متاحة، فإننا نتحقق مما إذا كانت مخصصة لمسلك آخر ينتظر موارد إضافية. إذا كان الأمر كذلك، فإننا نلغي الموارد المطلوبة من المسلك المنتظر ونخصصها للمسلك الطالب. إذا لم تكن الموارد متاحة أو محتفظ بها بواسطة مسلك منتظر، فيجب على المسلك الطالب الانتظار. أثناء الانتظار، قد يتم قطع بعض موارده، ولكن فقط إذا طلبها مسلك آخر. لا يمكن إعادة تشغيل المسلك إلا عندما يتم تخصيص الموارد الجديدة التي يطلبها واستعادة أي موارد تم قطعها أثناء الانتظار.

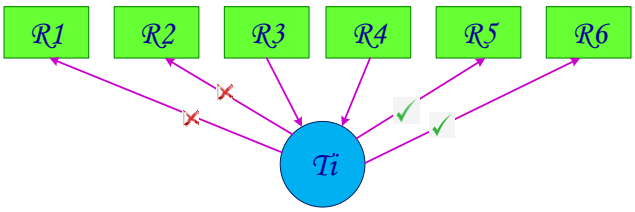


الشكل 6: الاستخدام السيء للموارد

غالبًا ما يتم تطبيق هذا البروتوكول على الموارد التي يمكن حفظ حالتها بسهولة واستعادتها لاحقًا، مثل سجلات وحدة المعالجة المركزية ومعاملات قاعدة البيانات. لا يمكن تطبيقه بشكل عام على موارد مثل أقفال المزامنة المتبادلة والسيمافورات، وهو نوع الموارد الذي يحدث فيه الجمود بشكل شائع.

الانتظار الدائري Circular Waiting:

تنتظر مجموعة من العمليات بعضها البعض بطريقة دائرية. على سبيل المثال، لنفترض أن هناك مجموعة من المسالك {T0، T1، T2، T3} بحيث تعتمد T0 على T1، وتعتمد T1 على T2، وتعتمد T2 على T3، وتعتمد T3 على T0 وهذا ينشئ علاقة دائرية بين كل هذه العمليات ويجب عليها الانتظار إلى الأبد حتى يتم تنفيذها. لمنع حدوث الانتظار الدائري يتم ترتيب الموارد بشكل تصاعدي مثلاً، بحيث أنه إذا كان أحد المسالك يحتفظ بمورد ما فإنه يستطيع أن يطلب الموارد ذات الترقيم الأعلى من الذي يحتفظ به ولا يمكنه طلب الموارد الأدنى قبل أن يحرر جميع موارده، كما في الشكل 7.



الشكل 7: عدم السماح بحصول الانتظار الدائري

الفكرة العامة وراء تجنب الجمود هي منع حدوث الجمود من خلال منع أحد الشروط المذكورة أعلاه على الأقل.

يتطلب هذا مزيدًا من المعلومات حول كل عملية، ويميل إلى التسبب في انخفاض استخدام الجهاز. (أي أنه نهج محافظ.)

في بعض الخوارزميات، يحتاج المجدول فقط إلى معرفة العدد الأقصى لكل مورد قد تستخدمه العملية. في الخوارزميات الأكثر تعقيدًا، يمكن للمجدول أيضًا الاستفادة من جدول الموارد التي قد تكون مطلوبة بالضبط وبأي ترتيب.

عندما يرى المجدول أن بدء عملية أو منح طلبات موارد قد يؤدي إلى جمود مستقبلي، فهذا يعني أن هذه العملية لن تبدأ أو أن الطلب لن يتم منحه.

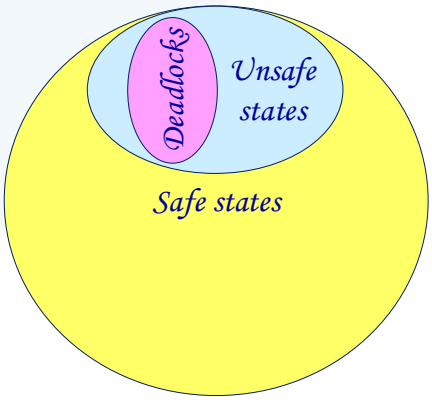
يتم تعريف حالة تخصيص الموارد بعدد الموارد المتاحة والمخصصة، والحد الأقصى لمتطلبات جميع العمليات في النظام.

الحالة الآمنة

تكون الحالة آمنة إذا كان النظام قادرًا على تخصيص جميع الموارد المطلوبة من قبل جميع العمليات أو المسالك (حتى الحد الأقصى المعلن) دون الدخول في حالة جمود.

بشكل آخر، تكون الحالة آمنة إذا كان هناك تسلسل آمن من العمليات {T0, T1, T2, ...Tn} بحيث يمكن منح جميع طلبات الموارد لـ Ti باستخدام الموارد المخصصة حاليًا لـ Ti وجميع المسالك Tj حيث (j < i. أي إذا انتهت جميع العمليات السابقة لـ Ti وحررت مواردها، فسيكون Ti قادرًا على الانتهاء أيضًا، باستخدام الموارد التي حررت.)

إذا لم يكن هناك تسلسل آمن، فسيكون النظام في حالة غير آمنة، مما قد يؤدي إلى إمكانية حدوث الجمود. (جميع الحالات الآمنة خالية من إمكانية حدوث الجمود، كما في الشكل 8 ولكن ليست كل الحالات غير الآمنة تؤدي إلى إمكانية حدوث الجمود.)

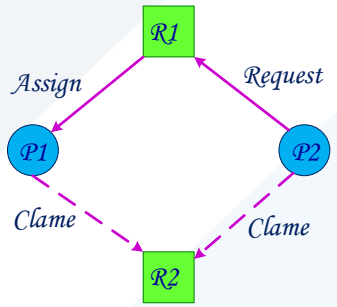


الشكل 8: حالات النظام

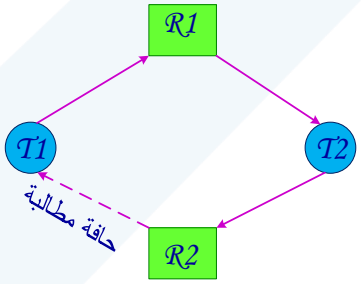
خوارزمية الرسم البياني لتخصيص الموارد Resource-Allocation Graph Algorithm

- إذا كانت فئات الموارد تحتوي على مثيلات فردية فقط من مواردها، فيمكن اكتشاف حالات الجمود من خلال الحلقات في الرسوم البيانية لتخصيص الموارد.

- في هذه الحالة، يمكن التعرف على الحالات غير الآمنة وتجنبها من خلال *تزويد* الرسم البياني لتخصيص الموارد بحواف المطالبة، التي يتم الإشارة إليها بخطوط متقطعة، والتي *تبدأ* من عملية *وتنتهي إلى* مورد قد تطلبه في المستقبل.
- لكي تعمل هذه التقنية، يجب إضافة جميع حواف المطالبة إلى الرسم البياني لأي عملية معينة قبل السماح لهذه العملية بطلب أي موارد. (*في الحالة البديلة*، لا يجوز للعمليات تقديم طلبات إلا للموارد التي أنشأت لها بالفعل حواف مطالبة، ولا يمكن إضافة حواف المطالبة إلى أي عملية تحتفظ بموارد حاليًا.)
- عندما تقدم عملية طلبًا، يتم تحويل حافة المطالبة Pi->Rj إلى حافة طلب. وبالمثل، عندما يتم تحرير مورد، تعود المهمة إلى حافة مطالبة.
- يعمل هذا النهج عن طريق رفض الطلبات التي من شأنها أن تنتج حلقات في الرسم البياني لتخصيص الموارد، مع الأخذ في الاعتبار حواف المطالبة.
- على سبيل المثال، فكر فيما يحدث عندما تطلب العملية P2 المورد R2:



الشكل : الرسم البياني لتخصيص الموارد عند تجنب الجمود



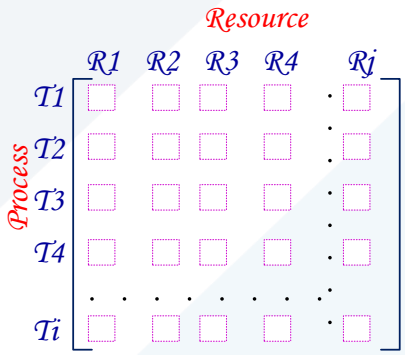
الشكل : حالة غير آمنة

- سيكون الرسم البياني لتخصيص الموارد الناتج عبارة عن حلقة، وبالتالي لا يمكن الموافقة على الطلب.

خوارزمية *المصرفي* Banker's Algorithm

- بالنسبة لفئات الموارد التي تحتوي على أكثر من مثيل، لا تعمل طريقة الرسم البياني لتخصيص الموارد، ويجب اختيار طرق أكثر تعقيدًا .
- حصلت خوارزمية المصرفي على اسمها لأنها طريقة يمكن للمصرفيين استخدامها للتأكد من أنه عندما يقرضون الموارد، سيظلون قادرين على إرضاء جميع عملائهم. (لن يقرض المصرفي القليل من المال لبدء بناء منزل ما لم يكن متأكدًا من أنه سيكون قادرًا لاحقًا على إقراض بقية المال لإكمال المنزل.)
- عندما تبدأ عملية ما، يجب أن تحدد مسبقًا الحد الأقصى لتخصيص الموارد التي قد تطلبها، حتى المبلغ المتاح على النظام.

- عند تقديم طلب، يحدد المجدول ما إذا كان منح الطلب سيترك النظام في حالة آمنة. إذا لم يكن الأمر كذلك، فيجب أن تنتظر العملية حتى يمكن منح الطلب بأمان.
- تعتمد خوارزمية المصرفي على العديد من هياكل البيانات الرئيسية: (حيث n هو عدد العمليات و m هو عدد فئات الموارد.)
- يشير Available[m] إلى عدد الموارد المتاحة حاليًا من كل نوع.
- يشير Max[n][m] إلى الحد الأقصى للطلب لكل عملية من كل مورد.
- يشير Allocation[n][m] إلى عدد كل فئة من الموارد المخصصة لكل عملية.
- يشير Need[n][m] إلى الموارد المتبقية المطلوبة من كل نوع لكل عملية. (لاحظ أن Need[i][j] = Max[i][j] - Allocation[i][j] لجميع (i,j)



- تبسيط المناقشات، نقوم بالملاحظات/الملاحظات التالية:
- يمكن التعامل مع صف واحد من متجه الحاجة، Need[i]، كمتجه يتوافق مع احتياجات العملية i، وعلى نحو مماثل بالنسبة للتخصيص والحد الأقصى.

خوارزمية الأمان Safety Algorithm

- لتطبيق خوارزمية المصرفي، نحتاج أولاً إلى طريقة لتحديد ما إذا كانت حالة آمنة أم لا.
- تحدد هذه الخوارزمية ما إذا كانت الحالة الحالية للنظام آمنة، وفقًا للخطوات التالية:

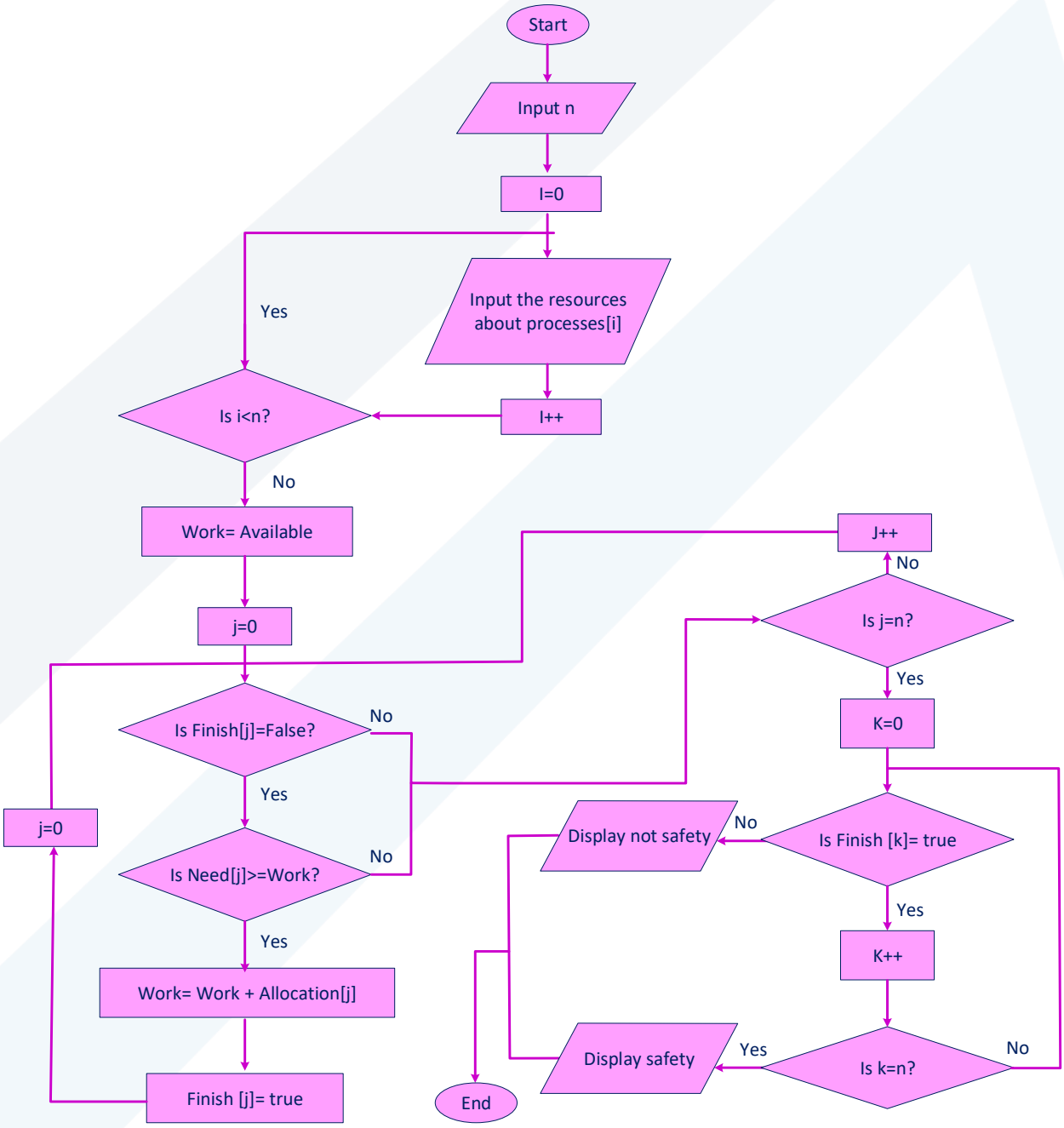
1. ليكن Work وFinish متجهان بطول m وn على التوالي.

- Work هو نسخة عاملة من الموارد المتاحة، والتي سيتم تعديلها أثناء التحليل.
- Finish هو متجه من القيم المنطقية التي تشير إلى ما إذا كانت عملية معينة يمكن أن تنتهي. (أو انتهت حتى الآن في التحليل.)
- قم بتهيئة Work إلى Available، وFinish إلى false لجميع العناصر.

2. ابحث عن i بحيث يكون كل من (A) Finish[i] == false، و (B) Need[i] < Work. لم تنته هذه العملية، ولكن يمكن أن تنتهي بمجموعة العمل المتاحة المعطاة. إذا لم يكن هناك مثل هذه i، فانتقل إلى الخطوة 4.
3. اضبط Work = Work + Allocation[i]، واضبط Finish[i] على true. يتوافق هذا مع انتهاء العملية i وإطلاق مواردها مرة أخرى في مجموعة العمل. ثم عد إلى الخطوة 2.
4. إذا كانت finish[i] == true لجميع i، فإن الحالة تكون حالة آمنة، لأنه تم العثور على تسلسل آمن.

- تعديل JTB:

1. في الخطوة 1. بدلاً من جعل Finish عبارة عن مجموعة من القيم المنطقية المهيأة إلى false، اجعلها مجموعة من الأعداد الصحيحة المهيأة إلى 0. قم أيضًا بتهيئة int s = 0 كعداد خطوة.
2. في الخطوة 2، ابحث عن Finish[i] == 0.
3. في الخطوة 3، اضبط Finish[i] على s++ حيث S تحسب عدد العمليات المكتملة.
4. بالنسبة للخطوة 4، يمكن أن يكون الاختبار إما Finish[i] > 0 لجميع i، أو s >= n تكمن فائدة هذه الطريقة في أنه إذا كانت هناك حالة آمنة، فإن Finish[] يشير إلى تسلسل آمن واحد (من بين العديد من التسلسلات المحتملة.)



خوارزمية طلب المورد Resource-Request Algorithm

- الآن بعد أن أصبح لدينا أداة لتحديد ما إذا كانت حالة معينة آمنة أم لا، فنحن الآن على استعداد لإلقاء نظرة على خوارزمية المصرفي نفسها.

- تحدد هذه الخوارزمية ما إذا كان الطلب الجديد آمنًا، وتمنحه فقط إذا كان من الآمن القيام بذلك.
- عند وجود طلب (لا يتجاوز الموارد المتاحة حاليًا)، يتم التعامل مع الأمر وكأنه قد تم تخصيص هذا المورد وفقا للطلب المذكور، ثم يتم التحقق مما إذا كانت الحالة الناتجة آمنة. إذا كان الأمر كذلك، فيتم تخصيص المورد بشكل نهائي، وإذا لم يكن كذلك، يتم رفض الطلب، على النحو التالي:

1. ليكن Request[n][m] عدد الموارد المطلوبة حاليًا من كل نوع بواسطة المسالك. إذا كان Request[i] > Need[i] لأي مسلك i، يتم الإعلان عن حالة خطأ.
2. إذا كان Request[i] > Available لأي مسلك i، فيجب أن ينتظر هذا المسلك حتى تصبح الموارد متاحة. وإلا يمكن للمسلك الاستمرار إلى الخطوة 3.
3. تحقق لمعرفة ما إذا كان يمكن تحقيق الطلب بأمان، من خلال التظاهر بأنه قد تم تحقيقه ثم معرفة ما إذا كانت الحالة الناتجة آمنة. إذا كان الأمر كذلك، يتم تحقيق الطلب، وإذا لم يكن كذلك، فيجب أن ينتظر المسلك حتى يمكن تحقيق الطلب بأمان. الإجراء الخاص بتحقيق الطلب (أو التظاهر بذلك لأغراض الاختبار) هو:

- Available = Available - Request
- Allocation = Allocation + Request
- Need = Need – Request

مثال توضيحي

لتوضيح استخدام خوارزمية المصرفي، ضع في اعتبارك نظامًا به خمسة مسالك من T0 إلى T4 وثلاثة أنواع من الموارد A وB و C. يحتوي نوع المورد A على عشر حالات، ويحتوي نوع المورد B على خمس حالات، ويحتوي نوع المورد C على سبع حالات. افترض أن الصورة التالية تمثل الحالة الحالية للنظام:

	Allocation	Max	Need	Available
	A B C	A B C	A B C	A B C
T0	0 1 0	7 5 3	7 4 3	3 3 2
T1	2 0 0	3 2 2	1 2 2	
T2	3 0 2	9 0 2	6 0 0	
T3	2 1 1	2 2 2	0 1 1	
T4	0 0 2	4 3 3	4 3 1	

لدينا الآن ثلاث مثيلات من المورد A وثلاث مثيلات من المورد B ومثيلان من المورد C نلاحظ أن 332<743 أي أن حاجة المسلك T0 لا يمكن تلبيتها من كل الموارد. بينما 332>122 أي يمكن تلبية حاجة المسلك T1 بشكل يمكن معه انهاء تنفيذه حتى النهاية ثم إعادة جميع الموارد التي لديه إلى النظام. كذلك الأمر بالنسبة للمسلك T3. أي يمكننا السماح للمسلك T1 أو المسلك T3 ببدء التنفيذ. بفرض أننا سمحنا للمسلك T1 فإنه حين اكتمال تنفيذه سيقوم بإعادة جميع موارده إلى النظام ليصبح لدينا 532 أي 5 مثيلات من المورد A وثلاث من المورد B وثلاث من المورد C. وفي هذه الحالة سيمكننا متابعة التنفيذ للمسلك T3 أو T4. بفرض أننا سمحنا لـ T3 فستقوم بالتنفيذ حتى النهاية ثم ستعيد بعد ذلك ماتملكه من الموارد إلى النظام ليصبح لدينا 743 أي 7 موارد من A وأربع موارد من B و ثلاث موارد من C ويمكننا في هذه الحالة تخصيصها لأي من المسالك المتبقية . وهكذا نرى أنه لدينا أكثر من تسلسل تنفيذ يؤمن لدينا التنفيذ في الحالة الآمنة أي دون الخوف من حدوث الجمود بسبب ذلك.

- والآن فكر فيما يحدث إذا طلبت العملية P1 مثيلًا واحدًا من A ومثيلين من C. (Request[1] = (1, 0, 2))

	Allocation	Need	Available
	A B C	A B C	A B C
T0	0 1 0	7 4 3	2 3 0
T1	3 0 2	0 2 0	
T2	3 0 2	6 0 0	
T3	2 1 1	0 1 1	
T4	0 0 2	4 3 1	

- ماذا عن طلبات (3,0 ,3) المقدمة من T4؟ أو (0, 2, 0) المقدمة من T0؟ هل يمكن قبولها بأمان؟ لماذا أو لماذا لا؟

Deadlock Detection كشف الجمود

- إذا لم يتم تجنب حالات الجمود، فهناك نهج آخر يتمثل في اكتشاف وقت حدوثها والتعافي منها بطريقة ما.
- بالإضافة إلى التأثير السلبي على الأداء الناتج عن التحقق المستمر من حالات الجمود، يجب وضع سياسة/خوارزمية للتعافي من حالات الجمود، وهناك احتمال لفقدان العمل عندما يتعين إلغاء العمليات أو استباق مواردها.

Single Instance of Each Resource Type حالة موارد كل منها بمثيل واحد

- إذا كانت كل فئة من الموارد تحتوي على مثيل واحد، فيمكننا استخدام تنويعة من مخطط تخصيص الموارد المعروف باسم مخطط الانتظار.
- يمكن إنشاء مخطط الانتظار من مخطط تخصيص الموارد عن طريق إزالة الموارد وتقليص الحواف المرتبطة بها، كما هو موضح في الشكل أدناه.
- يشير القوس arc من Ti إلى Tj في مخطط الانتظار إلى أن العملية Ti تنتظر موردًا تحتفظ به العملية Tj حاليًا.

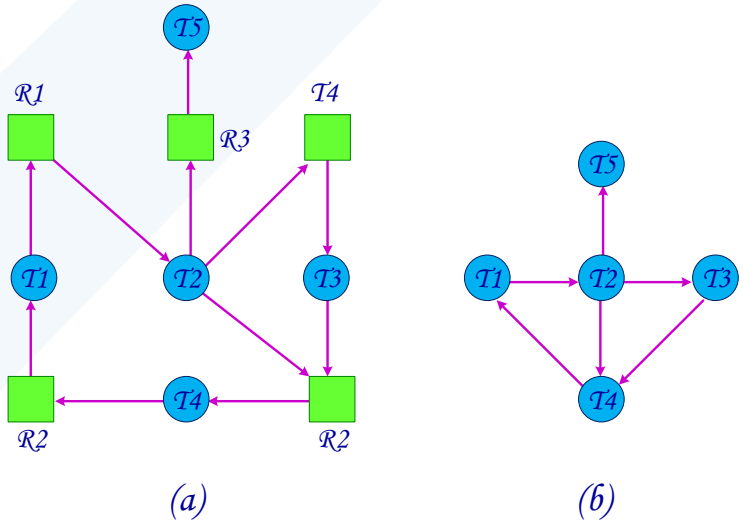


Figure 7.9 - (a) Resource allocation graph. (b) Corresponding wait-for graph

- كما في السابق، تشير الحلقات في الرسم البياني للانتظار إلى حالة الجمود .
- يجب أن تحافظ هذه الخوارزمية على الرسم البياني للانتظار، وتبحث فيه بشكل دوري عن الحلقات.

Several Instances of Each Resource Type حالة موارد كل منها متعدد المثيلات

- خوارزمية الكشف الموضحة هنا هي في الأساس نفس خوارزمية Banker، مع اختلافين دقيقين:
  - في الخطوة 1، تقوم خوارزمية Banker بتعيين Finish[i] على false لجميع .تقوم الخوارزمية المقدمة هنا بتعيين Finish[i] على false فقط إذا لم يكن Allocation[i] صفرًا. إذا كانت الموارد المخصصة حاليًا لهذه العملية صفرًا، تقوم الخوارزمية بتعيين Finish[i] على true. هذا يفترض في الأساس أنه إذا كان بإمكان جميع العمليات الأخرى الانتهاء، فيمكن لهذه العملية أيضًا الانتهاء. علاوة على ذلك، تبحث هذه الخوارزمية على وجه التحديد عن العمليات التي تشارك في حالة الجمود، ولا يمكن أن تشارك العملية التي ليس لديها أي موارد مخصصة في حالة الجمود ، وبالتالي يمكن إزالتها من أي اعتبار آخر.
  - الخطوتان 2 و3 لم تتغيرا
  - في الخطوة 4، تقول خوارزمية Banker الأساسية أنه إذا كان Finish[i] == true لجميع i، فلا يوجد حالة الجمود . هذه الخوارزمية أكثر تحديدًا، من خلال تحديد أنه إذا كانت Finish[i] == false لأي عملية Ti، فإن هذه المسلك متورطة بشكل خاص في الجمود الذي تم اكتشافه.
- (ملاحظة: تم تقديم طريقة بديلة أعلاه، حيث احتفظت Finish بالأعداد الصحيحة بدلاً من القيم المنطقية. سيتم تهيئة هذا المتجه إلى جميع الأصفار، ثم يتم ملؤه بأعداد صحيحة متزايدة مع اكتشاف العمليات التي يمكن الانتهاء منها. إذا تم ترك أي عمليات عند الصفر عند اكتمال الخوارزمية، فهناك جمود، وإذا لم يكن الأمر كذلك، فإن الأعداد الصحيحة في finish تصف تسلسلًا آمنًا. لتعديل هذه الخوارزمية لتتوافق مع هذا القسم من النص، يمكن ملء العمليات ذات التخصيص = صفر بـ N، N - 1، N - 2، وما إلى ذلك في الخطوة 1، وأي عمليات متبقية مع Finish = 0 في الخطوة 4 هي العمليات التي وصلت إلى حالة الجمود .)
- ضع في اعتبارك، على سبيل المثال، الحالة التالية، وحدد ما إذا كانت في حالة جمود حاليًا:

	Allocation	Request	Available
	A B C	A B C	A B C
T0	0 1 0	0 0 0	0 0 0
T1	2 0 0	2 0 2	
T2	3 0 3	0 0 0	
T3	2 1 1	1 0 0	
T4	0 0 2	0 0 2	

- لنفترض الآن أن المسلك T2 يقدم طلبًا للحصول على مثيل إضافي من النوع C، مما يؤدي إلى الحالة الموضحة أدناه. هل أصبح النظام الآن في حالة جمود؟

Detection-Algorithm Usage استخدام خوارزمية الكشف

- متى يجب إجراء اكتشاف الجمود؟ بشكل متكرر أم بشكل غير متكرر؟

- قد تعتمد الإجابة على عدد مرات حدوث الجمود المتوقعة، بالإضافة إلى العواقب المحتملة لعدم اكتشافها على الفور. (إذا لم تتم إزالة الجمود فورًا عند حدوثه، فقد "تتراكم" المزيد والمزيد من العمليات خلف الجمود، مما يجعل المهمة النهائية المتمثلة في إلغاء حظر النظام أكثر صعوبة وربما يتسبب في إلحاق الضرر بمزيد من العمليات.)
- هناك طريقتان واضحتان، ولكل منهما مقايضات:

3. إجراء اكتشاف الجمود بعد كل تخصيص موارد لا يمكن منحه على الفور. يتميز هذا بميزة اكتشاف الجمود على الفور، في حين يشارك الحد الأدنى من العمليات في الجمود. (قد يعتبر المرء أن العملية التي تسبب طلبها في حدوث حالة الجمود هي "سبب" الجمود، ولكن من الناحية الواقعية فإن جميع العمليات في الحلقة مسؤولة بالتساوي عن الجمود الناتج.) الجانب السلبي لهذا النهج هو التكلفة العامة الشاملة والضرر الذي يلحق بالأداء بسبب التحقق من الجمود بشكل متكرر.
4. لا تقم باكتشاف الجمود إلا عندما يكون هناك دليل على حدوث الجمود، مثل عندما ينخفض استخدام وحدة المعالجة المركزية إلى 40% أو أي رقم سحري آخر. والميزة هنا هي أن اكتشاف الجمود يتم بشكل أقل تكرارًا، ولكن الجانب السلبي هو أنه يصبح من المستحيل اكتشاف العمليات المشاركة في الجمود الأصلي، وبالتالي فإن استعادة الجمود قد تكون أكثر تعقيدًا وتسبب ضررًا لمزيد من العمليات.

التعافي من حالة الجمود Recovery From Deadlock

- هناك ثلاثة طرق أساسية للتعافي من الجمود:

1. إبلاغ مشغل النظام، والسماح له/لها بالتدخل اليدوي.
2. إنهاء مسلك أو أكثر متورطة في الجمود
3. استباق الموارد.

انهاء العملية والمسالك Process and Threads Termination

- طريقتان أساسيتان، يستعيد كل منهما الموارد المخصصة للعمليات المنتهية:
  - إنهاء جميع العمليات المتورطة في الجمود. هذا يحل الجمود بالتأكيد، ولكن على حساب إنهاء عدد من العمليات أكثر مما هو ضروري تمامًا.
  - إنهاء العمليات واحدة تلو الأخرى حتى يتم كسر الجمود. هذا أكثر تحفظًا، لكنه يتطلب إجراء اكتشاف الجمود بعد كل خطوة.
- في الحالة الأخيرة، هناك العديد من العوامل التي يمكن أن تدخل في تحديد العمليات التي سيتم إنهاؤها بعد ذلك: أولويات العملية.

1. كم من الوقت تعمل العملية، ومدى قربها من الانتهاء.
2. كم عدد الموارد ونوعها التي تحتفظ بها العملية. (هل من السهل استباقها واستعادتها؟)
3. كم عدد الموارد الإضافية التي تحتاجها العملية لإكمالها.
4. كم عدد العمليات التي ستحتاج إلى إنهائها
5. ما إذا كانت العملية تفاعلية أم دفعية.
6. (ما إذا كانت العملية قد أجرت تغييرات غير قابلة للاستعادة على أي مورد أم لا.)

- عند استباق الموارد لتخفيف الجمود، هناك ثلاث قضايا مهمة يجب معالجتها:

1. اختيار الضحية – يتضمن تحديد الموارد التي سيتم استباقها من أي عملية العديد من معايير القرار نفسها الموضحة أعلاه.
2. التراجع – من الناحية المثالية، يرغب المرء في استعادة عملية استباق إلى حالة آمنة قبل النقطة التي تم فيها تخصيص هذا المورد في الأصل للعملية. لسوء الحظ، قد يكون من الصعب أو المستحيل تحديد ما هي هذه الحالة الآمنة، وبالتالي فإن التراجع الآمن الوحيد هو العودة إلى البداية. (أي إلغاء العملية وجعلها تبدأ من جديد.)
3. التجويع – كيف تضمن عدم تجويع العملية لأن مواردها يتم استباقها باستمرار؟ قد يكون أحد الخيارات هو استخدام نظام الأولوية وزيادة أولوية العملية في كل مرة يتم فيها استباق مواردها. في النهاية، يجب أن تحصل على أولوية عالية بما يكفي بحيث لا يتم استباقها بعد الآن.