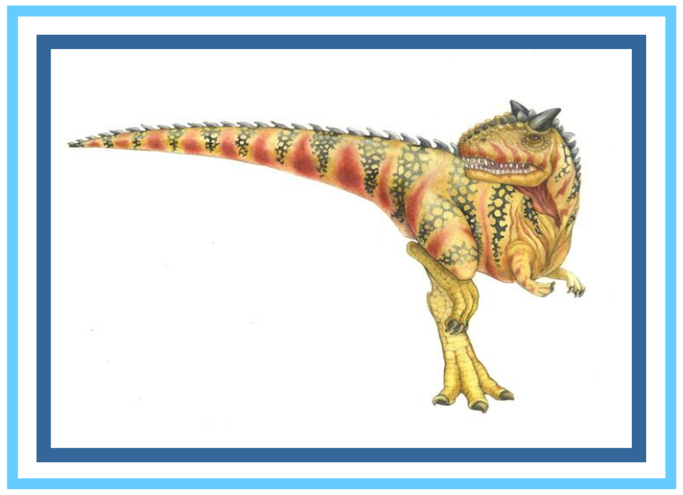
عملية الجدولة :6الفصل



عملية الجدولة :6الفصل

- خلفية
- والحرجة القسم مشكلة
- الحل بيترسون
- تزامن الأجهزة
- أقفال مزامنة
- الإشارات
- مشاكل كلاسيكية من التزامن
- ويرصد
- أمثلة تزامن
- مقاربات بديلة

أهداف

- للتعريف المشكلة مقطع حرج، التي يمكن استخدامها الحلول لضمان اتساق البيانات المشتركة
- أن يقدم كل من الحلول البرمجية والأجهزة من مشكلة القسم الحرجة
- لدراسة العديد من المشاكل العملية تزامن الكلاسيكية
- لاستكشاف العديد من الأدوات التي تستخدم في حل مشاكل التزامن عملية

خلفية

- ويمكن لعمليات تنفيذ في وقت واحد
  - قد ينقطع في أي وقت، واستكمال تنفيذ جزئيا
- الوصول المتزامن إلى البيانات المشتركة قد يؤدي إلى عدم تناسق البيانات
- الحفاظ على تناسق البيانات يتطلب آليات لضمان التنفيذ المنظم لعمليات التعاون
- التوضيح لهذه المشكلة:
  يمكننا أن نفعل ذلك من خلال وجود عدد صحيح **عداد** .لنفترض أننا نريد تقديم حل للمشكلة بين المستهلك والمنتج الذي يملأ **جميع** المخازن المؤقتة
  يتزايد من قبل المنتج بعد أن تنتج منطقة عازلة جديدة .0في البداية، **عداد** ومن المقرر أن .أن يحتفظ من عدد من مخازن كاملة
  وdecremented من قبل المستهلكين بعد أن تستهلك العازلة.

منتج

```
} (صحيح)بينما
/أنتجت المقبل *إنتاج عنصر في * /

; (حجم المخزن المؤقت ==العداد )بينما
/ *لا تفعل شيئا * /
.أنتجت المقبل = [في]عازلة
.% حجم المخزن المؤقت(1 +في ) =في
++مكافحة ;
}
```

مستهلك

```
} (صحيح)بينما
    (0 ==العداد )بينما
    / *لا تفعل شيئا  * / .
    .[من]عازلة  =تستهلك المقبل
    ؛--عداد .% حجم المخزن المؤقت(1 +من ) =من
    *تستهلك هذا البند في القادم المستهلكة  * /
/
}
```

حالة السباق

- يمكن تنفيذها في **++مكافحة**

```
register1 = عداد
register1 = register1 + 1
مكافحة    = register1
```

- يمكن تنفيذها في **عداد--**

```
REGISTER2 = عداد
REGISTER2 = REGISTER2 - 1
مكافحة    = REGISTER2
```

- في البداية "5 =العد" النظر في هذا التنفيذ التداخل مع:

S0: منتج تنفيذ **register1 = عداد** {register1 = 5}
S1: منتج تنفيذ **register1 = register1 + 1** {register1 = 6}
S2: تنفيذ المستهلك **REGISTER2 = عداد** {REGISTER2 = 5}
S3: تنفيذ المستهلك **REGISTER2 = REGISTER2 - 1** {REGISTER2 = 4}
S4: منتج تنفيذ **مكافحة= register1** { مكافحة= 6}
S5: تنفيذ المستهلك **مكافحة= REGISTER2** { مكافحة= 4}

مشكلة القسم حرجة

- النظر في نظام *ن* عمليات {*ع*$_1$، *ع*$_2$، ... *ع*$_ن$}
- لكل عملية **جزء حرج** جزء من التعليمات البرمجية
  - قد تكون عملية تغيير المتغيرات المشتركة، واستكمال الجدول، وكتابة الملف، الخ
  - عندما عملية واحدة في مقطع حرج، قد يكون هناك البعض في قسمها حرجة
- ***القسم مشكلة حرجة*** هو تصميم بروتوكول لحل هذه
- كل عملية يجب أن استأذن لدخول مقطع حرج في **القسم الدخول**، قد اتبع مقطع حرج مع **القسم الخروج**، ثم **القسم الباقي**

جزء حرج

- الهيكل العام للعملية *ص*ن هو

```
do {
        entry section
                critical section
        exit section
                remainder section
} while (true);
```

حل لمشكلة الحرجة القسم

1. **استبعاد متبادل** - إذا عملية ***P*** أنا تنفذ في قسمها حرج، فلا عمليات أخرى يمكن أن تنفذ في أقسام الحرجة
2. **تقدم** - إذا لم يكن هناك عملية المنفذة في قسمها النقدي وهناك توجد بعض العمليات التي يرغبون في دخول قسم انتقاداتهم، ثم اختيار العمليات التي ستدخل مقطع حرج المقبل لا يمكن تأجيلها إلى أجل غير مسمى
3. **يحدها الانتظار** - يجب أن يكون موجودا وملزمة على عدد المرات التي يسمح العمليات الأخرى للدخول أقسامها الحاسمة بعد إجراء عملية طلب لدخول قسمها النقدي وقبل منح هذا الطلب
   - نفترض أن كل عملية ينفذ بسرعة غير صفرية
   - لا الافتراض المتعلق **السرعة النسبية** ل *ن* العمليات

n نهجين اعتمادا على إذا النواة وقائية أو غير وقائية
- **وقائي** - يسمح الاستباقية من عملية عند التشغيل في وضع kernel
- **غير وقائي** - يستمر حتى وضع مخارج النواة، وبنات، وغلة أو طوعا وحدة المعالجة المركزية
  4 مجانا أساسا من شروط السباق في وضع النواة

الحل بيترسون

- وصف حسابي جيدة لحل المشكلة
- حل عملية اثنين
- وهذا هو، لا يمكن وقفها .نفترض أن **حمل** و **متجر**التعليمات هي ذرية
- تشترك العمليتين اثنين من المتغيرات:
  - **الباحث بدوره ؛**
  - **[2] علامة منطقية**
- المتغير **منعطف** يشير عليه الدور هو إدخال مقطع حرج
- أنا جاهز**P** يعني أن عملية **صحيح** = **[b] العلم** .ال **علم** يستخدم مجموعة لبيان ما إذا كان عملية مستعدة للدخول في مقطع حرج !

أنا Pالخوارزمية لعملية

```
فعل{
    العلم[ط] = صحيح.
    تحويل= ي.
    في حين( علم[ي] && تتحول== ي) ;
            جزء حرج
    العلم[ط] = كاذبة.
            القسم الباقي
} بينما (صحيح) ;
```

- يمكن اثباتها أن
1. يتم الاحتفاظ الإقصاء المتبادل
2. شرط التقدم اقتنعت
3. الانتظار-تم استيفاء متطلبات يحدها

تزامن الأجهزة

- توفر العديد من أنظمة دعم الأجهزة لمقطع التعليمات البرمجية حرجة
- كل الحلول القائمة أدناه على فكرة **قفل**
  - حماية المناطق الحرجة عن طريق أقفال
- Uniprocessors - يمكن تعطيل المقاطعات
  - قيد التشغيل حاليا لن كود تنفيذ دون الشفعة
  - عموما فعالة جدا على أنظمة متعددة المعالجات
    - أنظمة التشغيل باستخدام هذا ليس تحجيم نطاق واسع
- توفر الآلات الحديثة تعليمات الأجهزة الذرية الخاصة
  - **الذري**= عدم عدم انقطاع،
  - إما كلمة اختبار الذاكرة وقيمة المجموعة
  - أو محتويات مبادلة كلمتين الذاكرة

حل لقسم الحرجة المشكلة عن طريق أقفال

```
}فعل
    الحصول على القفل
        جزء حرج
    تأمين الافراج
        القسم الباقي
} في حين أن (TRUE) ؛
```

التعليمات test_and_set

- تعريف:

```
test_and_set منطقية( منطقي *الهدف)
 {
 رف منطقية                = * ؛المستهدفة
                  * =الهدف صحيح.
العودة رف                  :
 }
```

test_and_set () الحل باستخدام

- المشتركة قفل متغير منطقية، تهيئة إلى FALSE
- حل:

```
فعل{
    بينما (test_and_set (وقفل))
      . / * لا تفعل شيئا * /
  /* جزء حرج */
قفل = كاذبة؛
   / * القسم الباقي * /
} ؛ (صحيح) بينما
```

تعليماتcompare_and_swap

- تعريف:

```
قيمة، كثافة العمليات المتوقع، قيمة جديدة  *كثافة ) الباحث مقارنة وتبادل
  } (المترجم
القيمة؛  * =  الباحث درجة الحرارة
(المتوقع ==قيمة  *)  إذا
    * قيمة جديدة =القيمة.
العودة مؤقت  .
}
```

الحل باستخدامcompare_and_swap

- كل عملية لديها مفتاح متغير منطقي المحلي .FALSEالمشتركة قفل متغير منطقي تهيئة إلى
- حل:

```
فعل{
    بينما  ( مقارنة وتبادل( قفل، 0وقفل، 1)! = 0)
      . / * لا تفعل شيئا  * /
      /*جزء حرج */
قفل  = 0 ؛
      / *القسم الباقي * /
} ؛ (صحيح)بينما
```

test_and_setيحدها المنتظر الاستبعاد المتبادل مع

```
فعل{
انتظار   [ط] = صحيح.
مفتاح   = صحيح.
بينما  ( انتظار [أنا]في && مفتاح)
مفتاح      = test_and_set (وقفل) ؛
انتظار   [ط] =  كاذبة؛
  /*  جزء حرج */
ي   = ( ط+ 1)ن %.
بينما   ((ي! = ط) &&! الانتظار[ي])
ي       = ( ي+ 1)ن %.
إذا    ( ي==ط)
قفل      =  كاذبة؛
آخر
انتظار        [ي] = كاذبة.
  / *القسم الباقي * /
} بينما (صحيح)؛
```

أقفال مزامنة

- الحلول السابقة معقدة ويصعب الوصول إليها عادة للمبرمجين تطبيق
- مصممي نظام التشغيل بناء أدوات البرمجيات لحل مشكلة حرجة القسم
- أبسط هذه الطرق هي قفل مزامنة
- هذا **()** لقفل ثم **إطلاق سراح ()** المنتج المناطق الحرجة معها من قبل لأول مرة **اكتساب**
  - متغير منطقية تشير إلى إذا القفل هو متاح أم لا

- يجب أن يكون نوويا **()** و **إطلاق سراح ()** يدعو ل **اكتساب**
  - تنفذ عادة عن طريق تعليمات الذرية الأجهزة

- لكن هذا الحل يتطلب **الانتظار مشغول**
  - ولهذا القفل يسمى **spinlock**

()والافراج عن ()الحصول على

```
} () اكتساب
 (متوفرة !)   في حين
      . / * الانتظار مشغول * /
 متاح   = كاذبة ;;
}
{ () إطلاق سراح
 متاح   = صحيح.
}
```

```
{فعل
الحصول على القفل
جزء حرج
 تأمين الافراج
 القسم الباقي
} بينما (صحيح) ;
```

الملوحة جهاز

- أداة المزامنة التي لا تتطلب الانتظار مشغول
- متغير عدد صحيح - **S**الملوحة جهاز
- **()** و **إشارة ()** **انتظر :S**عمليتين القياسية تعديل
  - **()** و **الخامس ()** ودعا في الاصل **ف**
- أقل تعقيدا
- (ذرية)يمكن الوصول إليها إلا عن طريق عمليتين للتجزئة

```
الانتظار (S) {
 بينما     (S <= 0)
        . // الانتظار مشغول
    S--.
}
 إشارة (S) {
    S ++;
}
```

إشارة الاستخدام

- يمكن أن قيمة عددية تتراوح على مجال دون قيود - **عد إشارة**
- يمكن أن تتراوح قيمة عددية فقط بين 0 و1 - **إشارة الثنائية**
  - ثم **قفل مزامنة**
- كما إشارة الثنائية ***S*** يمكن تنفيذ إشارة العد
- يمكن أن تحل مشاكل التزامن المختلفة
- يعتبر P_1 و P_2 التي تتطلب S_1 قبل أن يحدث ذلك S_2

```
P1:
    S1.
 إشارة    (التوافق)؛
P2:
 الانتظار    (التوافق).
    S2.
```

تنفيذ إشارة

- على نفس الإشارة في نفس الوقت **()** و **إشارة** **()** يجب أن تضمن أن لا العمليتين يمكن تنفيذ **الانتظار**
- وهكذا، يصبح تنفيذ الجزء مشكلة حرجة حيث يتم وضع الانتظار ورمز إشارة في مقطع حرج
  - يمكن الآن لدينا **الانتظار مشغول** في تنفيذ مقطع حرج
    - لكن تنفيذ التعليمات البرمجية قصيرة
    - الانتظار قليلا مشغولة إذا مقطع حرج نادرا ما احتلت
- لاحظ أن التطبيقات قد تنفق الكثير من الوقت في مقاطع الهامة، وبالتالي هذا ليس حلا جيدا

تنفيذ إشارة
مع لا الانتظار مشغول

- مع كل إشارة هناك طابور الانتظار المرتبطة
- كل دخول في طابور الانتظار اثنين من عناصر البيانات:
  - قيمة (من نوع عدد صحيح)
  - المؤشر إلى السجل التالي في القائمة

- عمليتين:
  - مكان عملية استدعاء العملية على قائمة انتظار الانتظار المناسب - **منع**
  - إزالة واحدة من العمليات في طابور الانتظار ووضعه في قائمة انتظار جاهزة - **استيقظ**

تنفيذ إشارة مع (يتبع)لا تنتظر مشغول

```
typedef  والبنية{
القيمة الصحيحة    .
 البنية القائمة العملية    * ؛
} إشارة.
 الانتظار( إشارة* S) {
   S-> value--.
 إذا    (S-> قيمة<0) {
 إضافة هذه العملية إلى      S-> قائمة.
منع       () ؛
   }
}
 إشارة( إشارة* S) {
   S->  قيمة++ ؛
 إذا    (S-> قيمة<= 0) {
 إزالة عملية      P  منS-> قائمة.
استيقظ       ) ؛
   }
}
```

الجمود والمجاعة

- **مأزق** - اثنين أو أكثر من العمليات ينتظرون إلى ما لا نهاية لهذا الحدث يمكن أن يكون سبب واحد فقط من عمليات الانتظار
- سمح ***S*** يكون اثنين من الإشارات تهيئة إلى ***س*** و1

P_0 P_1

```
انتظار (S)  ؛ انتظار (Q) ؛
انتظار (Q)  ؛ انتظار (S) ؛
.
.
.
إشارة (S)  ؛ إشارة (س) ؛
إشارة (س)  ؛ إشارة (S) ؛
```

- **مجاعة** - **حجب لأجل غير مسمى**
  - قد لا يمكن إزالتها عملية من قائمة الانتظار عمود الاشارة التي يتم تعليقها
- **عكس الأولوية** - مشكلة الجدولة عندما يحمل عملية أولوية أقل قفل التي تحتاجها عملية العالي ذات الأولوية
  - تحل عبر **بروتوكول الأولوية الميراث**

مشاكل الكلاسيكية من التزامن

- المشاكل التقليدية المستخدمة لاختبار مخططات تزامن المقترحة حديثا
  - العازلة مشكلة-يحدها
  - القراء والكتاب مشكلة
  - تناول الطعام الفلاسفة مشكلة

العازلة مشكلة-يحدها

- ***ن*** مخازن، كل يمكن ان تحمل عنصر واحد
- 1إلى القيمة nitialized الملوحة جهاز **مـزامـنـة** أنا
- 0الملوحة جهاز **كـامـل** تهيئة إلى القيمة
- الملوحة جهاز **فـارغـة** تهيئة لن قيمة

(يتبع)يحدها العازلة مشكلة

- هيكل عملية منتج

```
فعل{
        ...
        / * إنتاج عنصر في next_produced * /
        ...
    انتظار (فارغ) ؛
    انتظار (مزامنة) ؛
        ...
        / * إضافة المنتجة بجانب المخزن المؤقت * /
        ...
    إشارة (مزامنة) ؛
    إشارة (كاملة) ؛
} بينما (صحيح) ؛
```

(يتبع)يحدها العازلة مشكلة

- هيكل عملية المستهلك

```
فعل{
    انتظار (كامل) ؛
    انتظار (مزامنة) ؛
        ...
        / * إزالة عنصر من عازلة لnext_consumed * /
        ...
    إشارة (مزامنة) ؛
    إشارة (فارغة) ؛
        ...
        / *تستهلك هذا البند في القادم المستهلكة * /
        ...
  } بينما (صحيح) ؛
```

قراء الكتاب مشكلة

- ويشارك مجموعة البيانات بين عدد من العمليات المتزامنة
  - القراء -قراءة فقط مجموعة البيانات. يفعلون ***ليس*** القيام بأي تحديثات
  - الكتاب -على حد سواء يمكن القراءة والكتابة
- تسمح للقراء متعددة لقراءة في نفس الوقت -مشكلة
  - كاتب واحد فقط يمكن الوصول إلى البيانات المشتركة في نفس الوقت
- كل تتضمن أولويات -العديد من الاختلافات في كيفية تعامل القراء والكتاب

- البيانات المشتركة
  - مجموعة البيانات
  - تهيئة ل **`rw_mutex`** الملوحة جهاز 1
  - تهيئة ل **`مزامنة`** الملوحة جهاز 1
  - تهيئة إلى **`read_count`** عدد صحيح 0

(يتبع)قراء الكتاب مشكلة

- هيكل عملية الكاتب

```
فعل{
    الانتظار    (مزامنة RW) ؛
        ...
        / *يتم تنفيذ الكتابة * /
        ...
 إشارة    (مزامنة RW) ؛
} بينما (صحيح) ؛
```

(يتبع)قراء الكتاب مشكلة

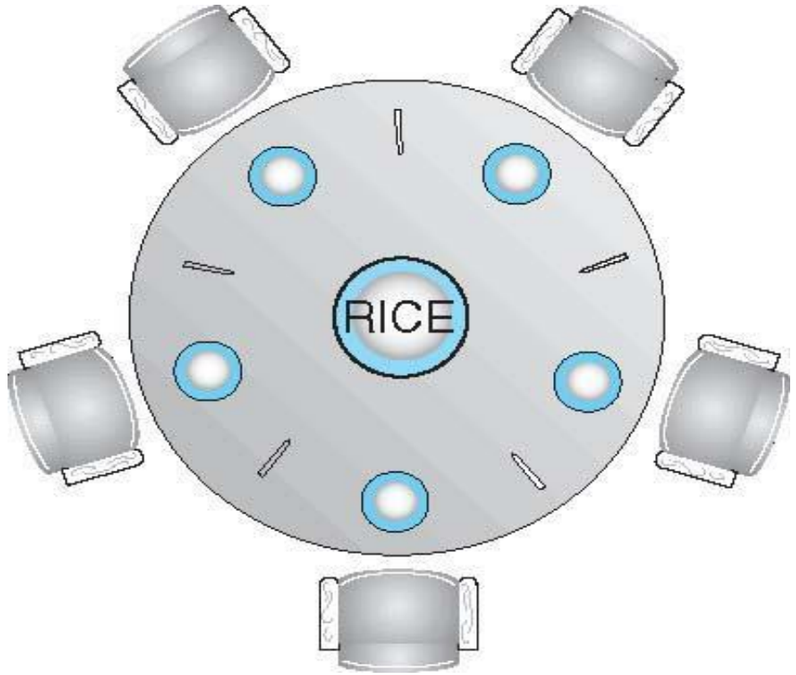
- هيكل عملية القارئ

- فعل{
  الانتظار(مزامنة)؛
  قراءة عدد++؛
  إذا(قراءة عدد== 1)
- الانتظار(RW مزامنة) إشارة (مزامنة)؛
- ...
  / * يتم تنفيذ القراءة */
- ... الانتظار(مزامنة)؛
  قراءةcount--.
  إذا(قراءة عدد== 0)
- إشارة(RW مزامنة) إشارة (مزامنة)؛
- } بينما (صحيح)؛

قراء الكتاب الاختلافات مشكلة

- لا القارئ ينتظرون إلا الكاتب لديه الإذن لاستخدام الكائن المشترك -*الأول* الاختلاف
- مرة واحدة الكاتب جاهز، فإنه يؤدي إرسال اسرع وقت ممكن -*ثانيا* الاختلاف
- قد يكون كل من الجوع مما يؤدي إلى المزيد من الاختلافات
- يتم حل المشكلة على بعض الأنظمة التي كتبها نواة توفير أقفال قارئ الكاتب

تناول الطعام الفلاسفة مشكلة



- الفلاسفة تنفق حياتهم التفكير والأكل
- لتناول الطعام من وعاء (واحد في وقت واحد)عيدان 2لا تتفاعل مع جيرانهم، وأحيانا محاولة لالتقاط
  - تحتاج كل من لتناول الطعام، ثم حرر كلا عند القيام به
- الفلاسفة 5في حالة
  - البيانات المشتركة
    - (مجموعة البيانات)عاء من الأرز 4
    - 1تهيئة ل [5]الملوحة جهاز عود 4

تناول الطعام الفلاسفة مشكلة خوارزمية

n هيكل الفيلسوف *أنا*:

```
}فعل
الانتظار        ( عود[أنا]؛(
                الانتظار( عود )[(ط + 1) ٪5]؛(

                // أكل

                إشارة( عود )[أنا]؛(
                إشارة( عود )[(ط + 1) ٪5]؛(

                // فكر في

} في حين أن (TRUE)؛(
```

n ما هي المشكلة مع هذه الخوارزمية؟

مشاكل مع أعمدة الإشارة

- الاستخدام غير الصحيح لعمليات الإشارة :
  - (مزامنة)الانتظار .... (مزامنة)إشارة
  - (مزامنة)انتظر ... (مزامنة) الانتظار
  - (أو كليهما) (مزامنة)أو إشارة (مزامنة) حذف من الانتظار
- الجمود والموت جوعا

ويرصد

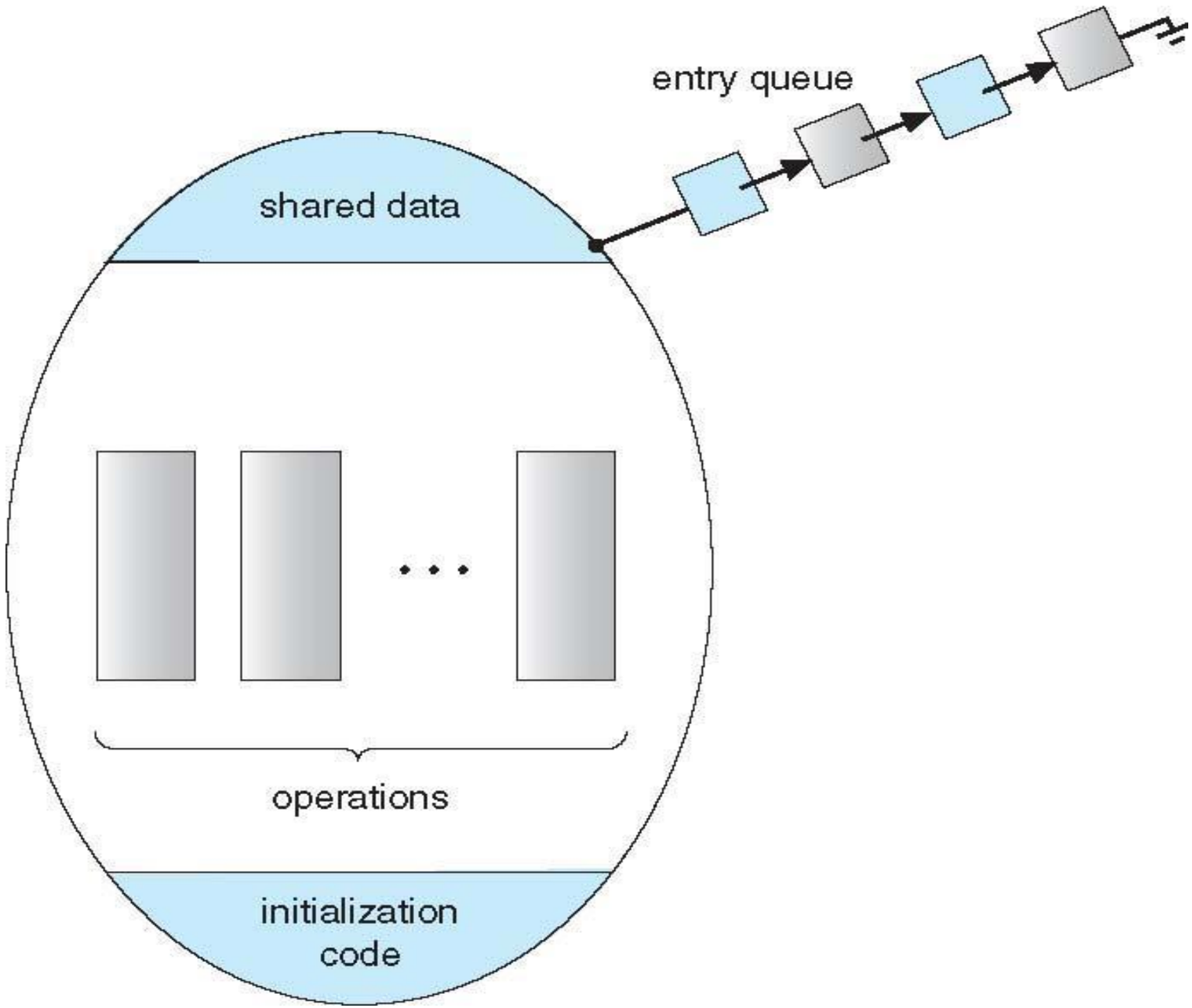
- والتجريد رفيع المستوى يوفر آلية مناسبة وفعالة لتزامن عملية
- *نوع البيانات* مجردة، والمتغيرات الداخلية يمكن الوصول إليها إلا عن طريق رمز ضمن الإجراء
- قد تكون عملية واحدة فقط نشطة داخل الشاشة في وقت واحد
- ولكن ليس بالقوة الكافية لنمذجة بعض المخططات تزامن

```
رصد مراقب اسم
{
   // تعريفات المتغير المشتركة
    الإجراءP1 (...) {.... }

    الإجراءPN (...) {.....}

 رمز التهيئة    (...) {...}
   }
}
```

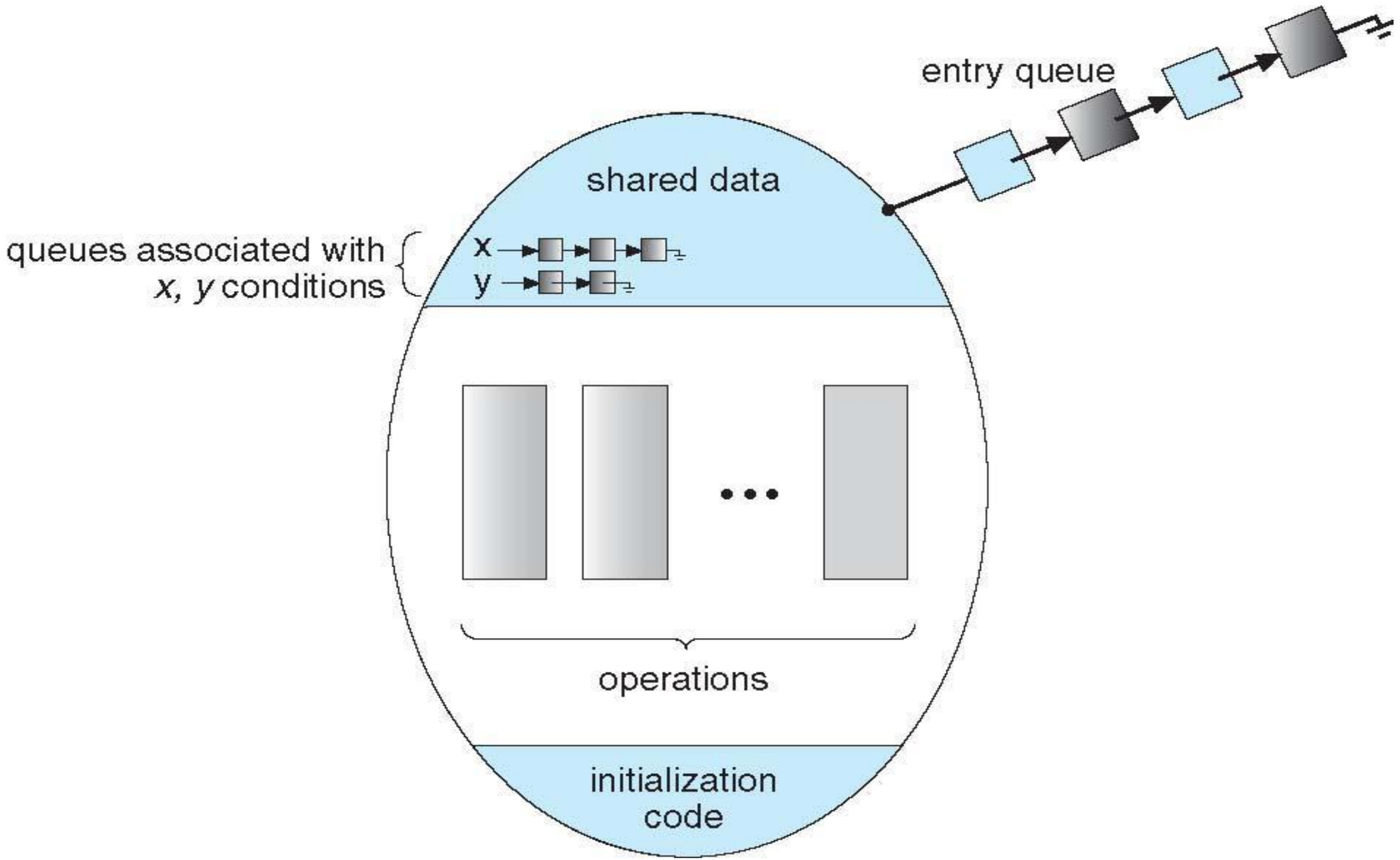
تخطيطيا للمراقبة



حالة المتغيرات

- حالة س، ص.
- عمليتين على متغير حالة:
  - x.wait () - يتم تعليق عملية استدعاء العملية حتى x.signal ()
  - x.signal () - تستأنف واحدة من العمليات (لو اي) أن التذرع x.wait ()
    - على المتغير، ثم انه ليس لديها تأثير على المتغير x.wait () إذا لم يكن هناك

مع رصد الحالة المتغيرات



خيارات حالة المتغيرات

- الدولة، ما يجب أن يحدث بعد ذلك؟ x.wait ()، مع س في x.signal () Pإذا يتضرع عملية
  - يجب الانتظار Pإذا استأنفت س، ثم

- وتشمل الخيارات
  - **إشارة والانتظار**- ف ينتظر حتى يترك س الشاشة أو ينتظر شرط آخر
  - **إشارة والاستمرار**- ينتظر س حتى ف يترك الشاشة أو ينتظر شرط آخر

  - المنفذ لغة يمكن أن تقرر -كلاهما له إيجابيات وسلبيات
  - ويرصد تنفيذها في تسوية باسكال المتزامن
    - 4 إشارة تنفيذ يترك فورا الشاشة، يتم استئناف س P
  - ، جافا# Cنفذت في لغات أخرى بما في ذلك ميسا،

حل لالفلاسفة الطعام

```
DiningPhilosophers رصد
  {
      التعداد{التفكير. الأكل (جائعا،  دولة[5].
      حالة النفس[5].

      بيك اب باطل(ط كثافة العمليات) {
            الدولة[ط] = الجوعى.
            اختبار(ط)؛
            إذا( ولاية[ط] = الأكل!)  الذاتي[أنا] .wait.
      }

 putdown  باطل(ط كثافة العمليات) {
            الدولة[ط] = التفكير.
                  // اختبار اليسار واليمين الجيران
            اختبار((ط+ 4) %5)؛
            اختبار((I + 1) %5)؛
       }
```

(يتبع)حل لالفلاسفة الطعام

```
} (ط كثافة العمليات)اختبار باطل
        إذا((دولة [ط +4) ٪5] != الأكل) &&
        (ولاية [أنا] == الجوعى) &&
        ((ولاية [ط +1) ٪5] != الأكل) {
            الدولة[ط] = الأكل.
            النفس[أنا] .signal ();
        }
    }

    initialization_code () {
        ل( كثافة العمليات ط =0 ط؛ ط<5؛ ط++)
            الدولة[ط] = التفكير.
    }
}
```

(يتبع)حل لالفلاسفة الطعام

- استدعاء عمليات امسك() و ضع ارضا() كل فيلسوف *أنا* في التسلسل التالي:

DiningPhilosophers.pickup (ط)؛

أكل

DiningPhilosophers.putdown (ط)؛

- لا الجمود، ولكن الجوع ممكن

مراقبة تنفيذ عن طريق أعمدة الإشارة

- المتغيرات
  - إشارة مزامنة. // (= في البداية 1)
  - إشارة القادمة؛ // (= في البداية 0)
  - NEXT_COUNT = 0؛ الباحث
- سيحل محله ***F*** كل إجراء

```
الانتظار(مزامنة)؛
...
F.
...
إذا(NEXT_COUNT> 0)
    إشارة(المقبل)
آخر
    إشارة(مزامنة)؛
```

جثة

- ومن المؤكد أن الإقصاء المتبادل داخل شاشة

الحالة المتغيرات -مراقبة تنفيذ

- لكل متغير حالة*س*، لدينا:

```
إشارةx_sem. // (= في البداية0)
الباحثx_count = 0؛
```

- العمليةx.wait يمكن تنفيذها على النحو التالي:

```
س العد++؛
إذا(NEXT_COUNT> 0)
        إشارة(التالي).
آخر
        إشارة(مزامنة)؛
الانتظار(x_sem)؛
سcount--.
```

(يتبع)مراقبة تنفيذ

- العمليةx.signal يمكن تنفيذها على النحو التالي:

```
إذا(خ العد> 0) {
        NEXT_COUNT ++؛
         إشارة(x_sem)؛
         الانتظار(التالي).
        next_count--.
}
```

استئناف العمليات ضمن مراقب

- المنفذة، والتي ينبغي أن يتم استئناف؟ x.signal () إذا اصطفت العديد من العمليات في حالة س، و
- في كثير من الأحيان لا يكفي FCFS
- x.wait (ج) بناء النموذج **الانتظار** **-المشروط**
  - حيث c **عدد أولوية**
  - المقبل (أولوية قصوى)ومن المقرر العملية مع أقل عدد

ومراقبة تخصيص مورد واحد

```
مراقبةResourceAllocator
{
        قيمة المنطقية مشغول.
        حالة س.
        اكتساب باطل(الوقت كثافة العمليات) {
            إذا(مشغول)
                x.wait (الوقت).
            مشغول= صحيح.
        }
        الإفراج باطل() {
            مشغول= FALSE.
            x.signal () ؛
        }
    رمز التهيئة() {
        مشغول= FALSE.
        }
}
```

أمثلة تزامن

- سولاريس
- ويندوز إكس بي
- لينكس
- بثريدس

سولاريس التزامن

- ، ومتعدد المعالجة(بما في ذلك المواضيع في الوقت الحقيقي)تنفذ مجموعة متنوعة من الأقفال لدعم تعدد المهام، وخاصية تعدد
- الاستخدامات **كائنات المزامنة على التكيف** لكفاءة عندما حماية البيانات من أجزاء التعليمات البرمجية قصيرة
  - يبدأ بمثابة إشارة القياسية تدور قفل
  - إذا قفل عقد، وبخيط رفيع يعمل على وحدة المعالجة المركزية آخر، يدور
  - إذا قفل التي عقدها موضوع غير تديرها الدولة، ومنع النوم في انتظار إشارة من قفل يطلق سراحه
- الاستخدامات **متغيرات حالة**
- الاستخدامات **قراء الكتاب** أقفال عندما تحتاج أقسام أطول من رمز الوصول إلى البيانات
- الاستخدامات **البوابات** أن تأمر قائمة المواضيع الانتظار للحصول على إما مزامنة التكيف أو قفل قارئ الكاتب
  - البوابات هي لكل قفل عقد الخيط، وليس في وجوه
- الأولوية الميراث لكل دوار يعطي تشغيل الخيط أعلى من أولويات المواضيع في الباب الدوار ل

ويندوز إكس بي التزامن

- يستخدم أقنعة المقاطعة لحماية الوصول إلى الموارد العالمية على أنظمة أحادي المعالجة
- على أنظمة متعددة المعالجات **spinlocks** الاستخدامات
  - الخيط Spinlocking لن استبقت
- يوفر أيضا **كائنات المرسل** المستخدم الأراضي التي قد تعمل كائنات المزامنة، الإشارات، والأحداث، وتوقيت
  - **أحداث**
    - يعمل حدث مثل الكثير من متغير حالة 4
    - مؤقتات يخطر موضوع واحد أو أكثر عند وقت مضى
    - (موضوع سوف تسد)أو **غير أشارت الدولة** (كائن متوفرة)الأشياء مرسل إما **أشار للدولة**

لينكس التزامن

- لينكس:
  - ، تعطيل المقاطعات لتنفيذ مقاطع الهامة قصيرة2.6قبل نواة النسخة
  - والإصدارات الأحدث، استباقية تماما 2.6النسخة

- لينكس يوفر:
  - الإشارات
  - spinlocks
  - إصدارات القارئ الكاتب على حد سواء

- استبداله تمكين وتعطيل نواة الشفعة spinlocksعلى نظام وحدة المعالجة المركزية واحدة،

بثريدس التزامن

- بثريدس هو السراج المستقلة API
- أنه يوفر:
  - أقفال مزامنة
  - متغيرات حالة
- وتشمل ملحقات غير المحمولة:
  - للقراءة والكتابة الأقفال
  - spinlocks

المعاملات الذرية

- النظام النموذجي
- -استعادة مقرها تسجيل
- نقاط التفتيش
- المعاملات الذرية المتزامنة

النظام النموذجي

- ويؤكد أن العمليات تحدث كوحدة منطقية واحدة من العمل، في مجملها، أو لا على الاطلاق
- متعلقة بمجال نظم قواعد البيانات
- على الرغم من فشل نظام الكمبيوتر atomicityالتحدي هو مطمئن
- مجموعة من التعليمات أو العمليات التي يؤدي وظيفة منطقية واحدة -عملية تجارية
  - القرص -هنا نشعر بالقلق مع تغييرات في تخزين ثابت
  - الصفقة هي سلسلة من اقرأ و اكتب عمليات
  - عملية (الصفقة فشلت)أو إجهاض (صفقة ناجحة)أنهت يلتزم
  - يجب أن تكون المعاملة أجهضت تراجع التراجع عن أي التغييرات التي أجريت

أنواع وسائط التخزين

- المعلومات المخزنة هنا لا ينجو تعطل النظام -تخزين متقلبة
  - الذاكرة الرئيسية، مخبأ :على سبيل المثال
- معلومات عادة على قيد الحياة تحطم -غير قلق التخزين
  - الشريط والقرص :على سبيل المثال
- معلومات لم يفقد أبدا -تخزين ثابت
  - إلى الأجهزة مع وسائط فشل مستقلة RAIDليس من الممكن في الواقع، لذلك يقترب عبر النسخ المتماثل أو

الصفقة حيث تسبب الفشل فقدان المعلومات على تخزين atomicityالهدف من ذلك هو ضمان متقلبة

استعادة تسجيل القائم على

- سجل لتخزين المعلومات مستقرة حول كل التعديلات التي صفقة
- الأكثر شيوعا هو تسجيل الكتابة الاخضر
  - تسجيل الدخول تخزين ثابت، ويصف كل سجل سجل تشغيل واحد الكتابة المعاملة، بما في ذلك
    - اسم المعاملة
    - اسم العنصر البيانات
    - القيمة القديمة
    - قيمة جديدة
  - <Tأنا يبدأ> كتب لتسجيل عندما الصفقة أنا يبدأT
  - <Tأنا يرتكب> كتب عندما أنا يرتكبT
- تسجيل الدخول يجب أن تصل تخزين مستقرة قبل حدوث العملية على البيانات

خوارزمية استرداد تسجيل القائم على

- باستخدام السجل، يمكن نظام معالجة أي أخطاء الذاكرة المضطربة
  - تحديثها $T_{أنا}$ يعيد قيمة جميع البيانات عن طريق التراجع ($T_{أنا}$)
  - $T_{أنا}$ إلى قيم جديدة يحدد القيم من كافة البيانات في المعاملة إعادة ($T_{أنا}$)
- التراجع ($T_{أنا}$) وإعادة ($T_{أنا}$) لا بد وأن idempotent
  - يجب أن يكون الإعدام متعددة نفس النتيجة مثل إعدام واحدة
- إذا فشل نظام استعادة الدولة من جميع البيانات المحدثة عن طريق السجل
  - إذا احتوى السجل <$T_{أنا}$ يبدأ> بدون <$T_{أنا}$ يرتكب> التراجع ($T_{أنا}$)،
  - إذا احتوى السجل <$T_{أنا}$ يبدأ> و <$T_{أنا}$ يرتكب> إعادة ($T_{أنا}$)،

نقاط التفتيش

- سجل يمكن أن تصبح طويلة، والانتعاش قد تستغرق وقتا طويلا
- نقاط التفتيش تقصير سجل والانتعاش الوقت.
- مخطط نقاط التفتيش:
  1. الناتج عن تسجيل بيانات حاليا في تخزين متقلبة إلى تخزين ثابت
  2. انتاج جميع البيانات المعدلة من متقلبة إلى تخزين ثابت
  3. إلى سجل على تخزين ثابت <تفتيش>إخراج سجل سجل
- الآن الانتعاش يشمل فقط تي، مثل أن تي التي تنفذ قبل نقطة التفتيش الأخير، وجميع المعاملات بعد تي جميع المعاملات الأخرى بالفعل على تخزين ثابت

المعاملات المتزامنة

- يجب أن يكون معادلا لتنفيذ المسلسل- serializability
- يمكن أداء جميع المعاملات في مقطع حرج
  - غير فعال، مقيدة جدا
- توفير خوارزميات التحكم التزامن serializability

Serializability

- النظر في بندين البيانات ألف وباء
- النظر في المعاملات T_0 ت و $_1$ ت
- تنفيذ T_0 ت، $_1$ ت بالذرة
- تسلسل تنفيذ يسمى جدول
- أجل عملية منفذة بالذرة يسمى جدول تسلسلي
- جداول مسلسل صالحة N!، هناك Nوبالنسبة للمعاملات

الجدول1: T_0 ثم T_1

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

جدولNonserial

- يسمح المتراكبة تنفيذ جدولNonserial
  - مما أدى التنفيذ لا غير صحيح بالضرورة
- النظر في الجدول الزمني Sعمليات ،Oيا أنا
  - صراع إذا البند الوصول نفس البيانات، مع الكتابة واحد على الأقل
- ويا أنا لا تتعارضO &إذا يا أناياي العمليات المتتالية ومختلف المعاملات
  - Sأنا أي ما يعادل Oمع النظام تبادلت يا ي ' Sثم
- إذاS يمكنS ' عمليات عبر مبادلة nonconflicting
  - S هو الصراع للتسلسل

المتزامن الجدول الزمني للتسلسل :2الجدول

T_0	T_1
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

قفل بروتوكول

- من خلال ربط قفل مع كل بند من بنود البيانات serializabilityضمان
  - اتبع بروتوكول تأمين مراقبة الدخول
- أقفال
  - على البند س، رأنا يمكن قراءة س ولكن لم يكتب س (S)تأنا يوجد قفل وضع مشترك -المشتركة
  - على س، رأنا يمكن القراءة والكتابة س (X)تي لديها قفل الحصري وضع -حصرية
- تتطلب كل معاملة على البند س اكتساب قفل المناسب
- إذا قفل عقدت بالفعل، قد يكون طلب جديد إلى الانتظار
  - مماثلة لقراء الكتاب خوارزمية

بروتوكول قفل مرحلتين

- الصراع serializabilityيضمن عموما
- كل القضايا الصفقة لقفل وفتح طلبات على مرحلتين
  - أقفال الحصول على -متزايد
  - أقفال الإفراج -تقلص
- لا يمنع الجمود

البروتوكولات القائمة على الطابع الزمني

- الطابع الزمني يأمر -حدد النظام بين المعاملات مقدما
- $T_{أنا}$ يبدأ$T_{أنا}$قبل (الطابع الزمني $TS_{أنا}$ المرتبطة $T_{أنا}$عملية
  - إذا دخلت تي $T_{ي}$ النظام قبل $TS(T_{أنا}) < TS(T_{ي})$
  - يمكن أن تتولد من ساعة النظام أو مكافحة كما منطقية متزايد في كل دخول الصفقة TS
- الطوابع الزمنية تحديد ترتيبserializability
  - إذا $TS(T_{أنا}) < TS(T_{ي})$ $T_{ي}$ $T_{أنا}$ يظهر قبل T، ونظام يجب أن تضمن إنتاج جدول يعادل جدول تسلسلي حيث

بروتوكول تنفيذي على أساس زمني

- البند البيانات س يحصل على اثنين الطوابع
  - W- الطابع الزمني (Q) - بنجاح (س)أكبر زمني من أي صفقة التي نفذت الكتابة
  - R- الطابع الزمني (Q) - (س)أكبر الطابع الزمني للقراءة الناجحة
  - المنفذة (س)أو الكتابة (س)تحديث كلما قرأت
- بروتوكول الطابع الزمني يأمر يؤكد أي متضاربة اقرأ و اكتب أعدم في ترتيب زمني
- (س)لنفترض أن ينفذ تي قراءة
  - إذا TS ($T_{أنا}$) <W- الطابع الزمني (س)التي تم تجاوزها بالفعل ، تي تحتاج إلى قراءة قيمة س
    - $T_{أنا}$ تراجع اقرأ عملية رفض و
  - إذا TS ($T_{أنا}$) ≥ W- الطابع الزمني (س)
    - (TS ($T_{أنا}$)، R- الطابع الزمني (س)لتعيين الحد الأقصى (س)الطابع الزمني -R اقرأ أعدم،

بروتوكول الطابع الزمني يأمر

- لنفترض تي ينفذ الكتابة(س)
  - إذا TS (T$_{أنا}$) <R- الطابع الزمني (س)من قبل و، قيمة س التي تنتجها T$_{أنا}$ هناك حاجة يفترض أنه لن يتم إنتاجها T$_{أنا}$
    - تراجع T$_{أنا}$ كتابة عملية رفض،
  - إذا TS (T$_{أنا}$) <W- الطابع الزمني (س)من س محاولة كتابة قيمة المتقادمة من ، تي$_{أنا}$
    - تراجع T$_{أنا}$ كتابة عملية رفض و
  - وإلا، اكتب أعدم
- يتم تعيين الطابع الزمني الجديد وإعادة تشغيل T$_{أنا}$ أي التراجع المعاملة
- الصراع والتحرر من الجمود serializability الخوارزمية يضمن

جدولة ممكن بموجب بروتوكول الطابع الزمني

T_2	T_3
read(B)	
	read(B)
	write(B)
read(A)	
	read(A)
	write(A)