تزامن الإجراءات

العنوان	الصفحة
1. الإجراءات المتعاونة	4
2. مشكلة المنتج والمستهلك (producer consumer)	4
3. مشكلة المقطع الحرج (critical section)	5
4. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 1)	6
5. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 2)	7
6. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 3)	8
7. حلول في حالة عدة إجراءات (خوارزمية المخبز)	9
8. تعليمات التزامن على مستوى العتاديات	10
9. سيمافور	13
10. التحقيق البرمجي للسيمافور	13
11. الإقفال المتبادل	15
12. مشكلات التزامن–مشكلة المنتج والمستهلك (producer consumer)	16
13. مشكلة القراء والكتاب (readers writers)	17
14. مشكلة الفلاسفة المفكرين والطاعمين (dining philosophers)	19
15. المراقب	20
16. نشاط مسألة الحلاق النائم (sleeping barber)	21
17. نشاط مسألة مدخنات السجائر (cigarette smokers)	22
18. التمارين	23

الكلمات المفتاحية:

الإجراء المتعاون، المنتج والمستهلك، مقطع حرج، استبعاد متبادل، مقطع الدخول، مقطع الخروج، خوارزمية المخبز، سيمافور، انتظر وأرسل، انتظار مشغول، الإقفال المتبادل، القرّاء والكتّاب، الفلاسفة الطاعمين، المراقب.

ملخص:

يركز هذا الفصل على مفهوم التعاون بين الإجراءات، والمشكلات الناتجة عن هذا التعاون، بالإضافة إلى حلول هذه المشكلات.

أهداف تعليمية:

يهدف هذا الفصل إلى:

- التعرف على الإجراءات المتعاونة.
- مشكلة المنتج والمستهلك.
- مشكلة المقطع الحرج وحلولها (الخوارزمية1، الخوارزمية2، الخوارزمية3، خوارزمية المخبز).
- التعليمات الخاصة للتزامن على مستوى العتاديات.
- السيمافور واستخدامها لحل مشكلة المقطع الحرج.
- مشكلات التزامن (مشكلة المنتج والمستهلك، مشكلة القراء والكتاب، مشكلة الفلاسفة الطاعمين).

المخطط:

1. الإجراءات المتعاونة.
2. مشكلة المنتج والمستهلك (producer consumer).
3. مشكلة المقطع الحرج (critical section).
4. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 1).
5. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 2).
6. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 3).
7. حلول في حالة عدة إجراءات (خوارزمية المخبز).
8. تعليمات التزامن على مستوى العتاديات.
9. سيمافور.
10. التحقيق البرمجي للسيمافور.
11. الإقفال المتبادل.
12. مشكلات التزامن–مشكلة المنتج والمستهلك (producer consumer).
13. مشكلة القراء والكتاب (readers writers).
14. مشكلة الفلاسفة المفكرين والطاعمين (dining philosophers).
15. المراقب.
16. نشاط مسألة الحلاق النائم (sleeping barber).
17. نشاط مسألة مدخنات السجائر (cigarette smokers).
18. التمارين.

1. الإجراءات المتعاونة

إن الإجراء المتعاون هو إجراء يؤثر ويتأثر بالإجراءات الأخرى الموجودة في النظام.
يمكن أن تتشارك الإجراءات المتعاونة مباشرةً في فضاء العنونة المنطقية (الرماز والمعطيات)، ويتم هذا التشارك من خلال استخدام الإجراءات الخفيفة أو مسالك التنفيذ (threads)، كما يمكن أن يُسمح للإجراءات المتعاونة بالتشارك في المعطيات عن طريق الملفات.
يمكن أن يؤدي نفاذ مجموعة من الإجراءات على نحو متوازي إلى المعطيات المشتركة إلى عدم اتساق المعطيات، لذلك نحتاج إلى آليات لضمان التنفيذ بترتيب محدد لإجراءات متعاونة تتشارك في فضاء العناوين المنطقية.

2. مشكلة المنتج والمستهلك (producer consumer)

- لدينا إجراء للمنتج يقوم بتوليد عناصر، ووضعها في صِوان، حتى يتم أخذها من قبل المستهلك، حيث يمثل المتحول counter (عدد صحيح) عدد العناصر الحالي في الصِوان، ويبدأ بالصفر وتتم إضافة واحد إليه مع كل إضافة لعنصر جديد على الصِوان)، وفق الخوارزمية التالية:

```
while (1) {
    /* produce an item in nextProduced */
    while (counter == BUFFER_SIZE)
          ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

- تكون مساحة الصِوان الخاص بالتخزين محدودة (BUFFER_SIZE)، ولذلك يقوم المنتج بإنتاج عناصر، طالما أن الصِوان يحوي مساحة فارغة، وإلا فإنه يتوقف عن العمل بانتظار فراغ أحد عناصر المخزن على الأقل.
- بينما يقوم إجراء المستهلك بأخذ العناصر من الصِوان طالما أنه يحتوي على عناصر، وإلا فإنه يتوقف عن العمل بانتظار قدوم عنصر واحد على الأقل إلى الصِوان، حيث يتم في الخوارزمية انقاص قيمة المتحول counter بمقدار واحد مع كل عملية أخذ عنصر من الصِوان، وفق الخوارزمية التالية:

```
while (1) {
    while (counter == 0)
          ; // do nothing
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
    /* consume the item in nextConsumed */
}
```

على الرغم من أن إجرائي المنتج والمستهلك صحيحان كلاً على حدة، إلا أنهما قد لا يقومان بسلوك صحيح إذا نفذا على التوازي. سنوضح ذلك بالمثال التالي:

لنفرض أن قيمة المتحول count تساوي 5، وأن إجرائي المنتج والمستهلك تنفذان العبارتين "counter++" و "counter--" على التوازي، تبعاً لذلك يمكن أن يأخذ المتغير counter إحدى القيم 4، 5، 6، والقيمة الوحيدة الصحيحة هي counter = 5، التي تنتج عندما تُنفذ كلٌ من التعليمتين على حده.

3. مشكلة المقطع الحرج (critical section)

ليكن لدينا نظام يحوي n إجراء $\{P_0, P_1, \dots, P_{n-1}\}$، لكل إجراء مقطع رماز يسمى المقطع الحرج، ويمكن للإجراء أن يغير متحولات مشتركة (جدول، ملف،...).

عندما يجري تنفيذ إجراء ضمن المقطع الحرج الخاص به، لا يُسمح لأي إجراء آخر أن يُنفِذ ضمن المقطع الحرج الذي يخصه، وبالتالي يجري تنفيذ المقاطع الحرجة بوساطة الإجراءات باستخدام "الاستبعاد المتبادل"، وبذلك يجري حلّ مشكلة المقطع الحرج خلال تصميم بروتوكول خاص، يُستخدم لوضع آلية تعاون بين الإجراءات عندما تعمل في مقاطعها الحرجة، حيث يتوجب على كل إجراء أن يطلب السماح له بدخول مقطعه الحرج من خلال (مقطع الدخول)، بينما يجري الخروج من المقطع الحرج من خلال (مقطع الخروج).

```
Do {
```

entry section

critical section

exit section

remainder section

```
} while (1);
```

يجري حلّ مشكلة المقطع الحرج عبر توفير المتطلبات التالية:

- الاستبعاد المتبادل: إذا كان إجراء Pi ينفذ ضمن المقطع الحرج الخاص به، فلا يمكن لأي إجراء آخر أن يكون ضمن المقطع الحرج الخاص به.

- التقدم: إذا لم يكن هنالك أي إجراء ينفذ ضمن المقطع الحرج الخاص به، وصدف وجود مجموعة إجراءات أخرى تريد دخول مقاطعها الحرجة، عندئذ يمكن للإجراءات التي توجد خارج مقاطعها الحرجة فقط، أن تشارك في تقرير من هو الإجراء التالي للدخول إلى المقطع الحرج.
- الانتظار المحدود: عند طلب إجراء ما الدخول إلى مقطعه الحرج، فهنالك حد لعدد المرات التي يسمح فيه لإجراءات أخرى بدخول مقاطعها الحرجة قبل تلبية هذا الطلب.
-

4. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 1)

لنفرض وجود إجراءين P_0 و P_1 (أو نرمز إليهم بـ p_i و p_j).

في الخوارزمية 1، لنترك الإجراءين يتشاركان في متحول صحيح مشترك turn يبدأ بالصفر (أو الواحد)، فإذا كان turn == i، عندها يُسمح للإجراء p_i بالتنفيذ في المقطع الحرج الخاص به.

```
Do {
    while (turn != i) ;

    critical section

    turn = j ;

    remainder section

} while (1);
```

يضمن هذا الحل وجود إجراء واحد فقط داخل المقطع الحرج الخاص به، لكنه لا يحقق شرط التقدم، لأنه يتطلب تبديلاً يدوياً ثابتاً للإجراءات لتنفيذ المقطع الحرج. مثلاً إذا كان turn == 0 و p_1 مستعد لدخول المقطع الحرج الخاص به، فإن p_1 لا يستطيع ذلك، حتى ولو لم يكن p_0 في المقطع الحرج.

5. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 2)

تكمن مشكلة الخوارزمية 1 بأنها لا تحتفظ بمعلومات كافية عن حالة كل إجراء، فهي تتذكر فقط الإجراء الذي يُسمح له بدخول المقطع الحرج الخاص به، ولحل هذه المشكلة، يمكن أن نستخدم بدلاً من المتغير turn المصفوفة flag التالية: Boolean flag[2].

تأخذ عناصر المصفوفة القيمة false في البداية، فإذا كان flag[i] يساوي true فإن الإجراء p_i مستعد لدخول المقطع الحرج الخاص به.

في هذه الخوارزمية يقوم الإجراء p_i بإسناد القيمة true إلى flag[i] إشارةً إلى أنه مستعد لدخول المقطع الحرج الخاص به، بعد ذلك، يتحقق p_i أن P_j ليس مستعداً لدخول المقطع الحرج الخاص به، لأنه لو كان مستعد فينبغي على p_i أن ينتظر إشارة من p_j تدل على أنه لم يعد في مقطعه الحرج.

```
Do {
    flag[i] = true;
    while (flag[j]) ;
        critical section
    flag[i] = false;
        remainder section
} while (1);
```

تحقق هذه الخوارزمية شرط الاستبعاد المتبادل، في حين لا تحقق شرط التقدُم: فعلى سبيل المثال، لن يتحقق التقدم إذا نفذ p_1 عملية إسناد flag[0] إلى true ، ونفذ p_0 ينفذ عملية إسناد flag[1] إلى true .

flag[0] = true

flag[1] = true

6. حلول لمشكلة المقطع الحرج في حالة إجراءين (الخوارزمية 3)

تستفيد هذه الخوارزمية من الأفكار الأساسية في الخوارزميتين السابقتين، للحصول على حل صحيح لمشكلة المقطع الحرج، تتحقق فيه المتطلبات الثلاثة.

نُعرِّف: الجدول البولياني flag والمتحول الصحيح turn حيث تكون قيمة كلٍ من flag[0] و flag[1] تساوي false وتكون قيمة turn 0 أو 1.

كي يدخل الإجراء Pi إلى المقطع الحرج، يُسنِد الإجراء القيمة true إلى flag[i] والقيمة j إلى turn، ليتأكد بذلك أنه دور الإجراء الآخر للدخول في حال كان الشرط محققاً لذلك.

إذا حاول الإجراءان الدخول في نفس الوقت، فإن turn سيأخذ القيمتين i و j بنفس الوقت، وإحدى عمليتي الإسناد سوف تبقى، والقيمة الأخيرة التي يأخذها turn (بعمليتي إسناد متتاليتين) هي التي تقرر أي منهما سيسمح له بالدخول أولاً إلى مقطعه الحرج.

```
Do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = false;

        remainder section

} while (1);
```

تضمن هذه الخوارزمية تحقق الشروط الثلاثة لمشكلة المقطع الحرج أي، الاستبعاد المتبادل، والتقدم، والانتظار المحدود.

7. حلول في حالة عدة إجراءات (خوارزمية المخبز)

يمكن تلخيص مبدأ خوارزمية المخبز بما يلي:

- عند الدخول إلى المخزن، يحصل كل زبون على رقم، والزبون الحاصل على أصغر رقم هو من يُخدَّم في المرة القادمة.
- في حال حصول إجراءين (زبونين) مختلفين على نفس الرقم، فإن الإجراء ذو الاسم الأصغر هو الذي يُخدَّم أولاً (إذا حصل الإجراءان P_i و P_j على نفس الرقم وكان i<j فإن P_i هو الذي يُخدَّم أولاً).
- تستخدم هذه الخوارزمية بنى المعطيات التالية:
  - مصفوفة من n متحول منطقي choosing
  - مصفوفة من n عدد صحيح number.

  حيث n هو عدد الإجراءات
- يجري في البداية إسناد القيمتين false و 0 إلى المصفوفتين السابقتين على الترتيب.

  int number[n]
- ونُعرِّف فيها العلاقات التالية:

$$(a,b) < (c,d) \quad \text{if} \quad [(a = = c \text{ and } b < d) \text{ or } (a < c)]$$

$$\max(a_0,\ldots,a_{n-1})==k \quad \text{where } (k >= a_i) \quad \text{for } i = 0,\ldots,n-1$$

```
Do {

    choosing[i] = true;
    number[i] = max(number[0], number[1], …, number[n-1]) + 1;
    choosing[i] = false;
    for (j = 0; j < n; j++) {
        while (choosing[j]);
        while (number[j] != 0) &&
          ((number[j],j]) <(number[i],i)));
    }

critical section

    number[i] = 0;

remainder section

} while (1);
```

تحقق هذه الخوارزمية شروط المقطع الحرج.

8. تعليمات التزامن على مستوى العتاديات

- هناك تعليمات عتادية بسيطة متاحة في العديد من النظم، من أجل حل مشكلة المقطع الحرج.
- ففي بيئة أحادية المعالج، يمكن أن تُحلّ مشكلة المقطع الحرج من خلال منع حدوث مقاطعة أثناء تعديل متحول مشترك، مما يعني أن تنفيذ سلسلة من التعليمات الحالية، يجري بالترتيب، وبدون مقاطعة، وبدون إجراء أية تعديلات غير متوقعة على المتحول المشترك.
- أما في بيئة متعددة المعالجات، فتتوفر في العديد من الآلات، تعليمات عتادية خاصة، تسمح إما بتعديل محتوى كلمة، أو التبديل بين محتوى كلمتين، وذلك دون تجزئة.
- نُعرِّف تعليمة TestAndSet (اختبر وأسند)، وهي تعليمة لا تقبل التجزئة، فإذا تم تنفيذ تعليمتي TestAndSet في نفس الوقت (كل منها على معالج مختلف)، فإن تنفيذهما يجري على التسلسل وفق ترتيب معين.
- تعريف تعليمة TestAndSet:

```
boolean TestAndSet (boolean &target) {
    boolean rv = target;
    target = true;
    return rv;
}
```

- من أجل تحقيق الاستبعاد المتبادل باستخدام TestAndSet، يجري التصريح عن متغير بولياني lock، يأخذ القيمة الابتدائية false، وتتغير القيمة وفقاً لحجز المقطع الحرج أو لا.
- خوارزمية تحقيق الاستبعاد المتبادل باستخدام TestAndSet:

```
Do {
        while (TestAndSet(lock));
            critical section
        lock = false;
            remainder section
} while (1);
```

- كذلك، نُعرِّف تعليمة التبديل (swap)، التي تعمل على تبديل محتوى كلمتين من دون تجزئة، ويجري تنفيذ الاستبعاد المتبادل باستخدام swap، عبر التصريح عن متحول بولياني عام lock، يأخذ القيمة false، بالإضافة إلى متحول بولياني محلي key لكل إجراء.
- تعريف تعليمة التبديل swap:

```
boolean Swap (boolean &a, boolean &b) {
    boolean temp = a;
    a = b;
    b= temp;
}
```

- خوارزمية تحقيق الاستبعاد المتبادل باستخدام swap:

```
Do {

    key = true;
    while (key == true);
       swap(lock,key)

        critical section

    lock = false;

        remainder section

} while (1);
```

- خوارزمية تحقق جميع متطلبات المقطع الحرج باستخدام تعليمة TestAndSet:

```
Do {
```

```
waiting[i] = true;
key = true;
while (waiting[i] && key)
    key = TestAndSet(lock);
waiting[i] = false;
```

critical section

```
j = (i+1) % n;
while ((j != i) && !waiting[j])
    j = (i+1) % n;
if (j == i)
    lock = false;
else
    waiting[j] = false;
```

remainder section

```
} while (1);
```

ولكن الخوارزميتين السابقتين لا تحققان شرط الانتظار المحدود، ومن أجل تحقيق ذلك يجري تعريف خوارزمية تحقيق جميع متطلبات المقطع الحرج، باستخدام تعليمة TestAndSet وبنية معطيات مشتركة مؤلفة من جدول بولياني waiting (بطول n حيث n عدد الإجراءات)، ومتحول بولياني lock. تأخذ بنى المعطيات الآنفة الذكر، القيمة الابتدائية false.

9. سيمافور

- ليس سهلاً في المسائل المعقدة، تعميم حلول مشكلة المقطع الحرج، ولذلك يمكننا اللجوء إلى أداة لتحقيق التزامن ندعوها سيمافور.
- نُعرِّف السيمافور بأنه عبارة عن متحول صحيح S، يجري النفاذ إليه من خلال عمليتين قياسيتين غير قابلتين للتجزئة wait (انتظر) و signal (أرسل).
- تعريف التعليمة wait.

```
wait (S) {
    while (S ≤ 0)
    ;//no-op
    S--;
}
```

- تعريف التعليمة signal.

```
signal (S) {
    S++;
}
```

- يجب أن تكون التعديلات التي تُجريها العمليتان wait و signal على قيمة المتغير الصحيح الخاص بالسيمافور، غير قابلة للتجزئة، أي إذا كان أحد الإجراءات يعدّل قيمة السيمافور، فيجب ألا يتمكن أي إجراء آخر من تعديل قيمة السيمافور نفسها في نفس الوقت.
- يمكن استخدام السيمافور من أجل حل مشكلة المقطع الحرج في حالة n إجراء، من خلال تشارك هذه الإجراءات في نفس السيمافور وبحيث يأخذ المتحول mutex القيمة الابتدائية 1.

10. التحقيق البرمجي للسيمافور

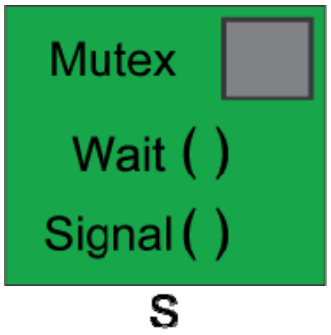
- تكمن السيئة الرئيسية لحلول الاستبعاد المتبادل المذكورة باستخدام السيمافور، في أنها تستخدم انتظاراً مشغولاً، فأثناء وجود إجراء في المقطع الحرج الخاص به، يجب على كل إجراء آخر يحاول دخول المقطع الحرج الخاص به، أن يدور في حلقة ضمن مقطع الدخول، وهذا الدوران يستهلك دورات وحدة المعالجة، التي يمكن استخدامها بعمل مفيد لإجراء آخر.
- لحل المشكلة السابقة يمكننا تعديل تعريف عمليتي wait و signal. فعندما يُنفِذ إجراء عملية wait وتكون قيمة السيمافور غير موجبة، فإن على هذا الإجراء أن ينتظر، وبدلاً من الانتظار المشغول، يتوقف الإجراء وينتظر في رتل انتظار خاص بالسيمافور وتتبدل حالته إلى حالة الانتظار.
- يتابع الإجراء المتوقف عمله عندما ينفذ إجراء آخر عملية signal، وذلك من خلال عملية wakeup (إيقاظ).

نُعرِّف السيمافور من خلال تسجيلة تحوي عنصرين: الأول value يحتوي على قيمة السيمافور، والثاني L يمثل رتلاً من الإجراءات المنتظِرة. يتناسب هذا التعريف الجديد مع المتطلبات الجديدة التي نسعى لتحقيقها.

كما نعيد تعريف العمليتين wait و signal لتأخذان بالحسبان التعريف الجديد للسيمافور. حيث تؤدي عملية block إلى توقف الإجراء الذي ينفذها، وتقوم العملية wakeup(p) بمتابعة تنفيذ الإجراء المتوقف p.

- تعريف بنية السيمافور.

```
typedef struct {
    int value;
    struct process *L;
} semaphore;
```



- التعريف الجديد لعملية wait (يتوقف الإجراء وينتظر في رتل انتظار خاص بالسيمافور وتتبدل حالته إلى حالة الانتظار).

```
void wait(semaphore S) {
    S.value--;
    if (S.value < 0) {
        add this process to S.L;
        block();
    }
}
```

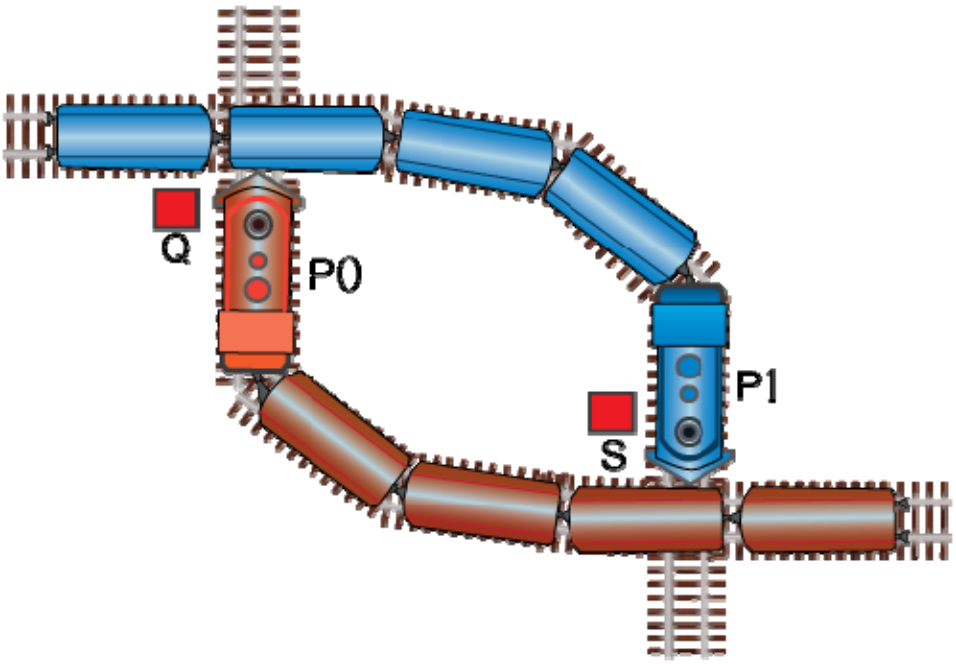
- التعريف الجديد لعملية signal.

```
void signal(semaphore S) {
    S.value++;
    if (S.value <= 0) {
        remove a process P from S.L;
        wakeup(P);
    }
}
```

11. الإقفال المتبادل

- يمكن أن يؤدي استخدام رتل الانتظار في تحقيق السيمافور إلى وضعٍ نجد فيه إجرائين أو أكثر في حالة انتظار لانهائي لحدث معين، وبحيث لا يمكن لهذا الحدث أن يتم إلا بتشغيل واحد من الإجراءات المنتظرة (الحدث المشار إليه هنا هو عملية signal لأحد السيمافور الموجودين في حالة انتظار).
- عندما تقع مثل هذه الحالة، نقول إن هذه الإجراءات قد وصلت إلى حالة الإقفال المتبادل.
- لتوضيح هذه الحالة، نأخذ نظام مؤلف من إجرائين P_0 و P_1 يريدان الدخول إلى السيمافور، S وإلى السيمافور، Q.
- لنفرض أن P_0 ينفذ wait(S) وأن P_1 ينفذ wait(Q)، وعندما ينفذ P_0 wait(Q)، يجب أن ينتظر أن ينفذ P_1 العملية signal(Q)، وبالمثل عندما ينفذ P_1 العملية wait(S)، يجب أن ينتظر أن ينفذ P_0 العملية signal(S)، ولأنه لا يمكن تنفيذ هاتين العمليتين، يتواجد P_0 و P_1 في حالة إقفال متبادل.
- مثال:

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
.	.
signal(S);	signal(Q);
signal(Q);	signal(S);



12. مشكلات التزامن – مشكلة المنتج والمستهلك (producer consumer)

- نفترض وجود مجموعة تحوي n صِوان، يتسع كل واحد إلى عنصر واحد. كما نستخدم السيمافور mutex لتوفير وظيفة الاستبعاد المتبادل أثناء النفاذ إلى مجموعة لصِوانات، حيث تأخذ mutex القيمة الابتدائية 1.
- نستخدم أيضاً السيمافور empty والسيمافور full، من أجل عدد الأصوِنة الفارغة والممتلئة على الترتيب، كما تُسند القيمتين n و 0 لهما على الترتيب أيضاً.

- بنية إجراء المنتج.

```
Do {
        ...
    produce an item in nextp
        ...
    wait(empty);
    wait(mutex);
        ...
    add nextp buffer to
        ...
    signal(mutex);
    signal(full);
} while(1);
```

- بنية إجراء المستهلك.

```
Do {
    wait(full);
    wait(mutex);
        ...
    remove an item from buffer to nextc
        ...
    signal(mutex);
    signal(empty);
        ...
    consume the item in nextc
        ...
} while(1);
```

13. مشكلة القراء والكتاب (readers writers)

- تتشارك الإجراءات المتوازية في أغراض المعطيات، وتجري عليها عمليات قراءة أو كتابة، ومن هنا نميز نوعين من الإجراءات القرّاء والكتّاب، إذا قام قارئان بالنفاذ إلى غرض مشترك في نفس الوقت، فليس هنالك من مشكلة، في حين إذا قام كاتب وإجراءات أخرى (قرّاء أو كتّاب) بالنفاذ إلى الغرض التشاركي فقد تحدث مشكلة، لذلك سنجعل الوصول إلى الغرض التشاركي بالنسبة للكتّاب، وصول حصري (أي يمنع أي إجراء آخر من الوصول إلى هذا الغرض).
- تتشارك إجراءات القرّاء ببنى المعطيات التالية: السيمافور، mutex و wrt بالإضافة إلى متحول صحيح readcount.
- تبدأ السيمافور mutex والسيمافور wrt بالقيمة 1، كما يبدأ المتغير readcount بالقيمة 0.
- يُستخدم السيمافور mutex لضمان الاستبعاد المتبادل عندما يجري تحديث المتغير readcount، ويُستخدم السيمافور wrt (وهو عبارة عن متحول مشترك بين إجراءات الكتّاب والقرّاء)، للاستبعاد المتبادل بين الكتّاب، كما يستخدمه القارئ الأول والأخير الذي يدخل المقطع الحرج أو يخرج منه.

- بنية إجراء الكاتب.

```
wait(wrt);
...
writing is performed
...
signal(wrt);
```

- نلاحظ من خوارزميات القرّاء والكتّاب، أنه إذا وُجِد كاتب في المقطع الحرج، وكان عدد القراء الذين ينتظرون يساوي n، فإن قارئاً واحداً سوف يدخل في رتل wrt، و $n-1$ قارئاً سوف يدخل في رتل mutex.
- بنية إجراء القارئ.

```
wait(mutex);
readcount++
if (readcount == 1)
    wait(wrt);
signal(mutex);
    ...
    reading is performed
    ...
wait(mutex);
readcount--;
if (readcount == 0)
    signal(wrt);
signal(mutex);
```

- كما نلاحظ أنه عندما ينفذ كاتب العملية signal(wrt) فإنه يمكن متابعة تنفيذ إما جميع القراء الذين ينتظرون وإما كاتب واحد ينتظر.

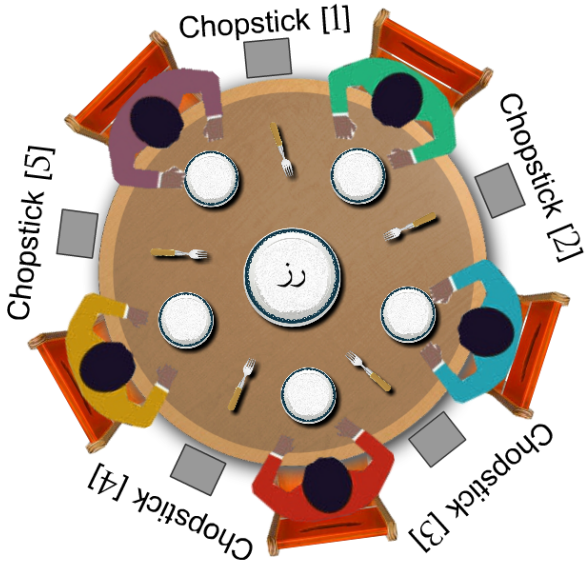
14. مشكلة الفلاسفة المفكرين والطاعمين (dining philosophers)

لنأخذ خمسة فلاسفة يقضون حياتهم في التفكير وتناول الطعام، حيث يتشارك الفلاسفة في طاولة دائرية يحيط بها خمسة كراسي (كرسي لكل فيلسوف)، وفي وسط الطاولة صحن أرز، ويوجد على الطاولة خمس أعواد. عندما يفكر الفيلسوف لا يتفاعل مع زملائه، وعندما يشعر بالجوع يحاول التقاط أقرب عودين إليه (العودين اليميني واليساري)، لا يستطيع الفيلسوف سوى التقاط عود واحد في وقت واحد، كما لا يستطيع أن يلتقط العود الموجود في يد جاره.

عندما يحصل الفيلسوف على عوديه يستطيع أن يأكل، على أن يتخلى عن عوديه عندما ينتهي.

تُعتبر هذه المسألة أحد المشاكل التقليدية في التزامن، ويعتمد حل المشكلة على تمثيل كل عود بسيمافور حيث يحاول الفيلسوف التقاط عود بتنفيذ عملية wait على ذلك السيمافور، ويحرر عوداً بتنفيذ عملية signal. يجري تمثيل المعطيات المشتركة في مصفوفة chopstick من خمسة سيمافورات وتأخذ جميع عناصر المصفوفة القيمة الابتدائية 1.

يضمن هذا الحل ألا يأكل جاران في نفس الوقت، لكنه لا يحل مشكلة الإقفال المتبادل الذي يسبب مجاعة (في حال جاع الفلاسفة جميعاً في نفس الوقت والتقط كل منهم أحد الأعواد على جانبيه).



- بنية إجراء الفيلسوف i.

```
Do {
    wait(chopstick[i]);
    wait(chopstick[i+1] % 5);
      ...
      eat
      ...
    signal(chopstick[i]);
    signal(chopstick[i+1] % 5);
      ...
      think
      ...
} while (1);
```

هنالك عدة أفكار لحل مشكلة الإقفال المتبادل:

- يُسمح لأربعة فلاسفة على الأكثر بالجلوس إلى الطاولة في نفس الوقت
- يُسمح لفيلسوف بالتقاط عوديه إذا كانا متوفران معاً
- يُسمح بحل لا متناظر يلتقط فيه الفيلسوف ذو الرقم الفردي عوده اليساري ومن ثم اليميني، في حين يلتقط الفيلسوف ذو الرقم الزوجي عوده اليميني ومن ثم اليساري

15.المراقب

يُعتبر المراقب بنية أخرى لتحقيق التزامن بين الإجراءات، وحل مشكلات المقطع الحرج.

يتألف المراقب من مجموعة من المتحولات (متحولات المراقب)، ومجموعة من الإجرائيات التي تنفذ العمليات على هذه المتحولات، حيث لا يمكن الوصول إلى المتحولات إلا من خلال هذه الإجرائيات، كما لا يمكن لهذه الإجرائيات إلا النفاذ لهذه المتحولات.

تضمن هذه البنية ألا يوجد سوى إجراء واحد فعال داخل المراقب في لحظة معينة.

16. نشاط: مسألة الحلاق النائم (sleeping barber)

- لدينا صالون للحلاقة يحوي على n كرسي للانتظار، بالإضافة إلى كرسي للحلاق.
- يمكن للحلاق أن ينام في حال عدم وجود زبائن، ولكن إذا دخل زبون إلى صالون الحلاقة، وكانت جميع الكراسي مشغولة، فإن الزبون يغادر الصالون.
- إذا كان الحلاق مشغولاً وكان هنالك كراسي فارغة، جلس الزبون في أحد الكراسي، وفي حال كان الحلاق نائماً، فإن الزبون يُوقظ الحلاق.
- اكتب برنامج ينظم عمل الحلاق والزبائن.

الحل:

```
#define CHAIRS 5  /* # chairs for waiting customers * /
typedef int semaphore; / *use your imagination * /
semaphore customers = 0; / *# of customers waiting for service*/
semaphore barbers = 0; / * # of barbers waiting for customers * /
semaphore mutex = 1; / * for mutual exclusion * /
int waiting = 0; / * customers are waiting (not being cut) * /

void barber(void)
{
while (TRUE) {
down(customers); / * go to sleep if # of customers is 0 * /
down(mutex); / * acquire access to 'waiting' * /
waiting = waiting - 1; / * decrement count of waiting customers * /
up(barbers); / * one barber is now ready to cut hair * /
up(mutex); / * release 'waiting' * /
cut

hair(); / * cut hair (outside critical region) * /
}
}


void customer(void)
{
down(mutex); / * enter critical region * /
if (waiting < CHAIRS) { / * if there are no free chairs, leave * /
waiting = waiting + 1; / * increment count of waiting customers * /
up(customers); / * wake up barber if necessary * /
up(mutex); / * release access to 'waiting' * /
down(barbers); / * go to sleep if # of free barbers is 0 * /
get

haircut(); / * be seated and be serviced * /
} else {
up(mutex); / * shop is full; do not wait * /
}
```

}

17.نشاط: مسألة مدخِّنات السجائر (cigarette smokers)

- لدينا ثلاثة إجراءات مدخِّنة وإجراء وكيل، حيث يقوم كل إجراء مدخِّن بلف سيجارة ثم تدخينها.
- يحتاج الإجراء المدخِّن إلى ثلاث مكونات: (تبغ، وورق، وأعواد ثقاب).
- يمتلك أحد الإجراءات المدخِّنة الورق، و يمتلك آخر التبغ، و يمتلك الثالث أعواد الثقاب، بينما يمتلك الإجراء الوكيل مخزون من المكونات الثلاثة.
- يضع الوكيل مكونين من ثلاثة على الطاولة، حيث يقوم الإجراء المدخن الذي يمتلك المكون الناقص بلف سيجارة وتدخينها، ومن ثم يقوم الوكيل بوضع مكونين من جديد على الطاولة وهكذا تستمر الحلقة.
- اكتب برنامج لمزامنة الإجراءات المدخِّنة والإجراء الوكيل.

18. التمارين:

1. **يمكن أن تتشارك الإجراءات المتعاونة مباشرةً في فضاء العنونة المنطقية (الرماز والمعطيات):**
   A. **صح**
   B. **خطأ**

2. **على الرغم من أن إجرائي المنتج والمستهلك صحيحان كلاً على حدة، إلا أنهما يقومان بسلوك صحيح إذا نفذا على التوازي:**
   A. صح
   B. خطأ

3. يجري حلّ مشكلة المقطع الحرج عبر توفير المتطلبات التالية:
   A. التقدم
   B. الاستبعاد المتبادل
   C. الانتظار المحدود
   D. جميع الإجابات صحيحة

4. تعليمة TestAndSet (اختبر وأسند)، وهي تعليمة تقبل التجزئة:
   A. صح
   B. خطأ

5. نُعرِّف السيمافور بأنه عبارة عن متحول صحيح S، يجري النفاذ إليه من خلال عمليتين قياسيتين غير قابلتين للتجزئة wait (انتظر) و signal (أرسل):
   A. صح
   B. خطأ

6. يتألف المراقب من مجموعة من المتحولات (متحولات المراقب)، ومجموعة من الإجرائيات التي تنفذ العمليات على هذه المتحولات:
   A. صح
   B. خطأ

رقم التمرين	الإجابة الصحيحة
.1	(A)
.2	(B)
.3	(D)
.4	(B)
.5	(A)
.6	(A)