Advanced Operating System

Lecture Notes

Dr. Professor, J.M. Khalifeh

قسم المعلوماتية

الوحدة الثانية

Unit-2

Classic Synchronization Problems

ملخص

في الوحدة السادسة من نظم التشغيل–1، بينا مشكلة القسم الحرج وركزنا على تحقق شروط السباق عندما تقوم عدة عمليات متزامنة بمشاركة البيانات. كما أوضحنا آلية عمل العديد من الأدوات التي تستخدم في معالجة مشكلة القسم الحرج من خلال منع تحقق شروط السباق. وتتراوح هذه الأدوات من حلول التي تعتمد على البنية الفيزيائية منخفضة المستوى (مثل حواجز الذاكرة وعملية المقارنة والمبادلة) إلى أدوات ذات مستوى أعلى بشكل متزايد (من أقفال المزامنة mutex إلى إشارات السيمافورات إلى الشاشات monitors). كما ناقشنا أيضًا تحديات مختلفة في تصميم التطبيقات مع السعي إلى عدم تحقق شروط السباق، بما في ذلك المخاطر التي تؤدي إلى توقف النظام مثل الجمود. في هذه الوحدة، حيث نطبق أدوات المزامنة على العديد من مشاكل المزامنة الكلاسيكية.

أهداف الوحدة

- التعرف على المشاكل الأساسية في المزامنة مثل مشكلة الخازن المحدود ومشكلة القاري الكاتب ومشكلة عشاء الفلاسفة
- شرح كيفية حل هذه المشاكل

مشاكل المزامنة الكلاسيكية Classic Problems of Synchronization

يعد الاتصال بين العمليات ضروريًا لكي تتمكن العمليات من التواصل ومشاركة البيانات. وفي حين قد يبدو الاتصال الأساسي بين العمليات بسيطًا، إلا أن بعض المواقف قد تتسبب في حدوث مشكلات تتطلب حلولًا محددة. تُعرف هذه المواقف باسم المشكلات الكلاسيكية حين الاتصال بين العمليات، والتي تتضمن إدارة المزامنة وتجنب الجمود وضمان الوصول إلى الموارد بطريقة خاضعة للرقابة، وهذه بعض منها:

- مشكلة المنتج والمستهلك
- مشكلة القراء والكتاب
- مشكلة الفلاسفة في تناول الطعام

تُستخدم هذه المشاكل لاختبار كل مخطط مزامنة مقترح حديثًا تقريبًا. في حلولنا للمشاكل، نستخدم السيمافورات للمزامنة، لأن هذه هي الطريقة التقليدية لتقديم مثل هذه الحلول. ومع ذلك، يمكن للتطبيقات الفعلية لهذه الحلول استخدام أقفال المزامنة المتبادلة mutex بدلاً من السيمافورات الثنائية.

مشكلة المخزن المحدود The Bounded-Buffer Problem

تطرقنا سابقا في نظم التشغيل1 إلى مشكلة المخزن المحدود وهي تُستخدم عادةً لتوضيح دور حل هذه المشكلة في حل قضايا المزامنة. وذلك من خلال تقديم نموذج برمجي عام.

Buffer of n Slots

مشكلة المنتج والمستهلك

تتضمن مشكلة المنتج والمستهلك نوعين من العمليات: المنتج الذي ينشئ البيانات، والمستهلك الذي يعالج تلك البيانات. ويتلخص التحدي في ضمان عدم قيام المنتج بملء المخزن المؤقت بشكل زائد، وعدم محاولة المستهلك استهلاك البيانات من مخزن مؤقت فارغ.

المواضيع الرئيسية في مشكلة المنتج والمستهلك:

تجاوز سعة المخزن المؤقت: إذا حاول المنتج إضافة بيانات عندما يكون المخزن المؤقت ممتلئًا، فلن تكون هناك مساحة للبيانات الجديدة، مما يتسبب في حظر المنتج.

فراغ المخزن المؤقت: إذا حاول المستهلك استهلاك البيانات عندما يكون المخزن المؤقت فارغًا، فلن يكون لديه ما يستهلكه، مما يتسبب في حظر المستهلك.

حل مشكلة المنتج والمستهلك

يجب التأكد من أن المنتج لن يحاول إضافة بيانات إلى المخزن المؤقت إذا كان ممتلئًا وأن المستهلك لن يحاول استهلاك البيانات من المخزن المؤقت الفارغ.

الحل: يجب على المنتج إما أن ينتقل إلى وضع السكون أو يتجاهل البيانات إذا كان المخزن المؤقت ممتلئًا. في المرة التالية التي يستخدم فيها المستهلك عنصرًا من المخزن المؤقت، فإنه يخطر المنتج، الذي يبدأ في ملء المخزن المؤقت مرة أخرى. وبنفس الطريقة، يمكن للمستهلك أن ينتقل إلى وضع السكون إذا وجد أن المخزن المؤقت فارغ. في المرة التالية التي يضع فيها المنتج البيانات في المخزن المؤقت، فإنه يوقظ المستهلك النائم.

قد يؤدي الحل غير المناسب إلى طريق مسدود حيث تنتظر كلتا العمليتين أن يتم إيقاظهما.

يمكن حل هذه المشكلة باستخدام تقنيات المزامنة مثل الإشارات أو المزامنات للتحكم في الوصول إلى المخزن المؤقت المشترك وضمان المزامنة المناسبة بين المنتج والمستهلك

تشترك عمليات المنتج المستهلك عامة في البنية التالية:

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```

نفترض أن المجموعة تتكون من n مخزن، كل منها قادر على تخزين عنصر واحد. يوفر السيمافور الثنائي الاستبعاد المتبادل لعمليات الوصول إلى مجموعة المخازن المؤقتة ويتم تهيئتها بقيمة 1. تعد السيمافورات الفارغة والممتلئة عدد المخازن الفارغة والممتلئة. يتم تهيئة السيمافور الفارغ بقيمةn ؛ ويتم تهيئة السيمافور الممتلئ بقيمة 0.

يبين الشكل 7.1 الكود الخاص بعملية المنتج بينما يبين الشكل 7.2 الكود الخاص بعملية المستهلك في الشكل 7.2. لاحظ التماثل بين المنتج والمستهلك. يمكننا تفسير هذا الكود على أنه المنتج الذي ينتج مخازن ممتلئة للمستهلك أو المستهلك الذي ينتج مخازن فارغة للمنتج.

```
while (true) {
. . .
produce an item in next produced */ */
. . .
wait(empty);
wait(mutex);
. . .
add next produced to the buffer */ */
. . .
signal(mutex);
signal(full);
}
```

الشكل 1: كود المنتج

```
while (true)
{
wait(full);
wait(mutex);
. . .
/* remove an item from buffer to next consumed */
. . .
signal(mutex); signal(empty);
. . .
/* consume the item in next consumed */ _
. . .
}
```

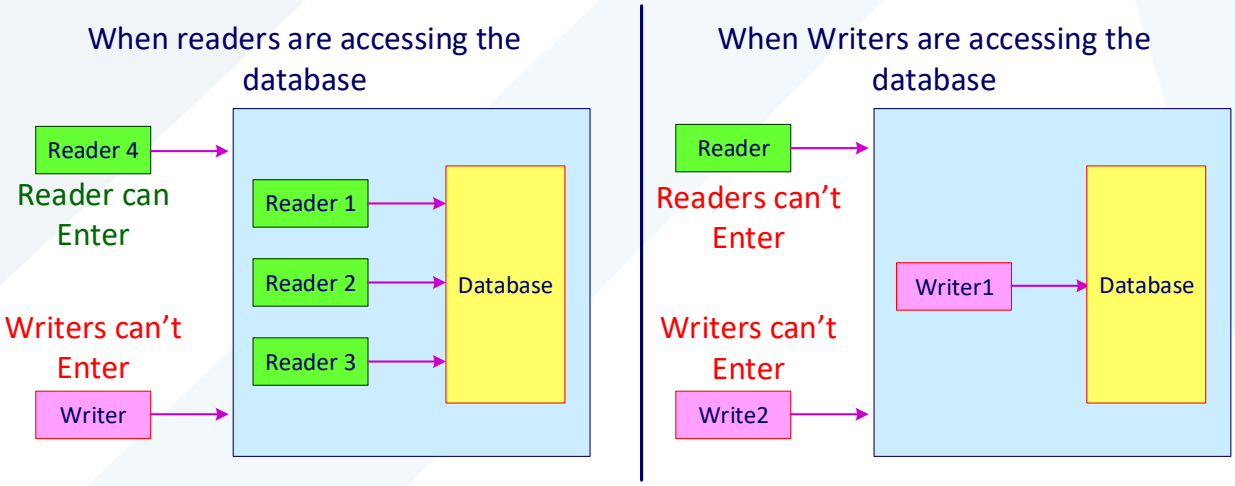
الشكل 2: كود المستهلك

مشكلة القارئ– الكاتب

بفرض أن هناك عدة عمليات متزامنة تتشارك في قاعدة بيانات. ومن الطبيعي أن ترغب بعض هذه العمليات في قراءة قاعدة البيانات فقط، في حين قد ترغب عمليات أخرى في تحديث (أي قراءة وكتابة) قاعدة البيانات. حيث نميز بين هذين النوعين من العمليات بالإشارة إلى الأولى باسم القراء وإلى الثانية باسم الكتاب. من الواضح أنه إذا قام قارئان أو أكثر بالوصول إلى البيانات المشتركة في وقت واحد، فلن تحدث أي آثار سلبية. ولكن إذا قام كاتب وبعض العمليات الأخرى (إما قارئ أو كاتب) بالوصول إلى قاعدة البيانات في وقت واحد، فقد تحدث فوضى.

لضمان عدم ظهور هذه الصعوبات، يجب أن يمتلك الكاتب حق الوصول الحصري إلى قاعدة البيانات المشتركة أثناء الكتابة في قاعدة البيانات. وهذا ما يشار إليه بأنه مشكلة القراء – الكتاب. منذ ذكرها في الأصل.

تحتوي مشكلة القراء – الكتاب على العديد من الخيارات، وكلها تنطوي على أولويات. أبسطها، يشار إليها باسم مشكلة القراء – الكتاب الأولى، تتطلب عدم إبقاء أي قارئ منتظرًا إذا لم يكن أي من الكتاب قد حصل بالفعل على إذن لاستخدام المورد المشترك. بعبارة أخرى، لا ينبغي لأي قارئ أن ينتظر القراء الآخرين لإنهاء القراءة لمجرد أن كاتبًا ينتظر. تتطلب مشكلة القراء والكتاب الثانية أنه بمجرد أن يكون الكاتب جاهزًا، يقوم هذا الكاتب بتنفيذ عملية الكتابة في أسرع وقت ممكن. بعبارة أخرى، إذا كان الكاتب ينتظر للوصول إلى الكائن، فلا يجوز لقراء جدد البدء في القراءة.



قد يؤدي حل أي من المشكلتين إلى المجاعة. في الحالة الأولى، قد يجوع الكتاب؛ وفي الحالة الثانية، قد يجوع القراء. لهذا السبب، تم اقتراح متغيرات أخرى للمشكلة. بعد ذلك، نقدم حلاً لمشكلة القراء والكتاب الأولى. راجع الملاحظات الببليوغرافية في نهاية الفصل للحصول على مراجع تصف حلولًا خالية من المجاعة لمشكلة القراء والكتاب الثانية.

في حل مشكلة القراء والكتاب الأولى، تشترك عمليات القارئ في هياكل البيانات التالية:

```
mutex = 1;_semaphore rw
semaphore mutex = 1;
count = 0;_int read
```

يتم تهيئة السيمافور mutex_rw والسيمافور mutex الثنائيين إلى 1؛ بينما تتم تهيئة السيمافور count_read إلى 0. يعد السيمافور mutex_rw مشتركًا لكل من عمليات القراءة والكتابة. يتم استخدام إشارة mutex لضمان الاستبعاد المتبادل عند تحديث قيمة متغير read_count. يتتبع متغير read_count عدد العمليات التي تقرأ الكائن حاليًا. تعمل إشارة

rw_mutex كإشارة استبعاد متبادل للكتاب. يتم استخدامها أيضًا بواسطة القارئ الأول أو الأخير الذي يدخل أو يخرج من القسم الحرج. لا يتم استخدامها بواسطة القراء الذين يدخلون أو يخرجون بينما القراء الآخرون في أقسامهم الحرجة.

```
while (true)
{
wait(rw_mutex);
. . .
/* writing is performed */
. . .
signal(rw_mutex);
}
```

الشكل 2: كود المستهلك

يبين الشكل 3 الكود الخاص بعملية الكاتب بينما يبين الشكل 4 الكود الخاص بعملية القارئ. لاحظ أنه إذا كان الكاتب في القسم الحرج وكان n قارئ ينتظر، فسيتم وضع قارئ واحد في قائمة انتظار على mutex القراءة والكتابة، ويتم وضع $n-1$ قارئ في قائمة انتظار على mutex. لاحظ أيضًا أنه عندما ينفذ الكاتب signal(rw_mutex)، فقد نستأنف السماح بتنفيذ القراء المنتظرين أو كاتب واحد منتظر. حيث يتم إجراء الاختيار بواسطة المجدول.

تم تعميم مشكلة القراء والكتاب وحلولها لتوفير الأقفال على بعض الأنظمة. يتطلب الحصول على قفل القارئ والكاتب تحديد وضع القفل: إما الوصول للقراءة أو الكتابة.

عندما ترغب إحدى العمليات في قراءة البيانات المشتركة فقط، فإنها تطلب قفل القارئ–الكاتب في وضع القراءة. يجب على العملية التي ترغب في تعديل البيانات المشتركة أن تطلب القفل في وضع الكتابة. يُسمح لعمليات متعددة بالحصول على قفل القارئ–الكاتب في وقت واحد في وضع القراءة، ولكن لا يجوز إلا لعملية واحدة الحصول على القفل للكتابة، حيث يلزم الوصول الحصري للكتاب.

```
while (true) {
wait(rw mutex);
. . .
/* writing is performed */
. . .
signal(rw mutex);
}
```

الشكل 3: كود الكاتب

```
while (true) {
wait(mutex);
read count++;
if (read count == 1)
wait(rw mutex);
signal(mutex);
. . .
/* reading is performed */
. . .
wait(mutex);
read count--;
if (read count == 0)
signal(rw mutex);
signal(mutex);
}
```

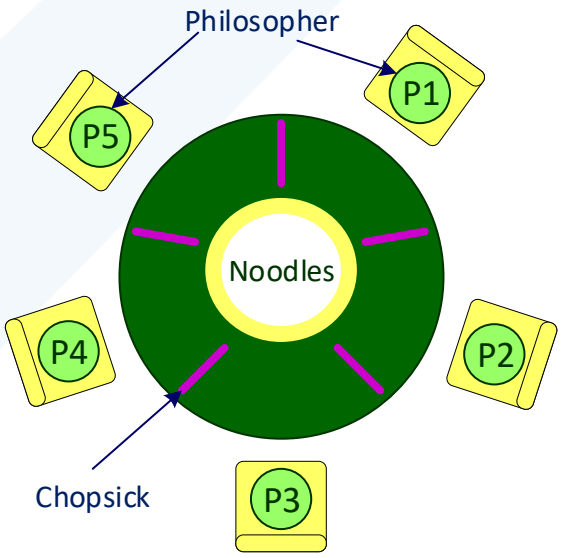
الشكل 4: كود القارئ

أقفال القارئ والكاتب مفيدة للغاية في المواقف التالية:

- في التطبيقات حيث يسهل تحديد العمليات التي تقرأ البيانات المشتركة فقط والعمليات التي تكتب البيانات المشتركة فقط.
- في التطبيقات التي تحتوي على عدد من القراء أكبر من عدد الكتاب. وذلك لأن أقفال القارئ–الكاتب تتطلب عمومًا تكلفة إضافية أكبر لتأسيسها من السيمافورات أو أقفال الاستبعاد المتبادل. إن التزامن المتزايد للسماح بقراء متعددين يعوض عن التكلفة الإضافية المتضمنة في إعداد قفل القارئ–الكاتب.

مشكلة عشاء الفلاسفة The Dining-Philosophers Problem

لنتأمل هنا خمسة فلاسفة يقضون حياتهم في التفكير والأكل. ويتقاسم الفلاسفة طاولة دائرية تحيط بها خمسة كراسي، كل كرسي منها يخص فيلسوفاً واحداً. وفي وسط الطاولة وعاء من الأرز، وتوضع على الطاولة خمسة عيدان طعام منفردة (الشكل 7.5). وعندما يفكر الفيلسوف، فإنه لا يتفاعل مع زملائه. ومن وقت لآخر، يشعر الفيلسوف بالجوع ويحاول التقاط عيدان الطعام الأقرب إليه (العيدان اللذان يقعان بينه وبين جاريه من اليمين واليسار). ولا يجوز للفيلسوف أن يلتقط أكثر من عود طعام واحد في كل مرة. ومن الواضح أنه لا يستطيع أن يلتقط عود طعام موجود بالفعل في يد جاره. وعندما يكون لدى الفيلسوف الجائع عيدان طعام في نفس الوقت، فإنه يأكل دون أن يترك عيدان الطعام. وعندما ينتهي من الأكل، يضع عيدان الطعام جانباً ويبدأ في التفكير من جديد. وتُعَد مشكلة تناول الطعام بين الفلاسفة مشكلة مزامنة كلاسيكية، ليس بسبب أهميتها العملية ولا لأن علماء الكمبيوتر يكرهون الفلاسفة، بل لأنها مثال على فئة كبيرة من مشاكل التحكم في التزامن. إنه تمثيل بسيط للحاجة إلى تخصيص العديد من الموارد بين العديد من العمليات بطريقة خالية من الجمود أو المجاعة.



الشكل 7: عشاء الفلاسفة

أحد الحلول البسيطة هو تمثيل كل عيدان طعام باستخدام السيمافور. يحاول الفيلسوف الإمساك بعود طعام من خلال تنفيذ عملية wait() معتمدة على السيمافور. ويطلق سراح عيدان الطعام الخاصة به من خلال تنفيذ عملية signal() على السيمافور المناسب. وبالتالي، فإن البيانات المشتركة هي

```
semaphore chopstick[5];
```

حيث يتم تهيئة جميع عناصر chopstick إلى 1. يظهر هيكل الفيلسوف i في الشكل 7.6.

```
while (true)
{
wait(chopstick[i]);
wait(chopstick[(i+1) % 5]);
. . .
/* eat for a while */
. . .
signal(chopstick[i]);
signal(chopstick[(i+1) % 5]);
. . .
/* think for a while */
. . .
}
```

وعلى الرغم من أن هذا الحل يضمن عدم تناول جارين للطعام في نفس الوقت، إلا أنه يجب رفضه لأنه قد يؤدي إلى حالة من الجمود. لنفترض أن الفلاسفة الخمسة أصبحوا جائعين في نفس الوقت وأمسك كل منهم بعوده الأيسر. ستكون جميع عناصر chopstick مساوية الآن لـ 0. عندما يحاول كل فيلسوف الإمساك بعوده الأيمن، فسوف يتأخر إلى الأبد.

هناك عدة حلول ممكنة لمشكلة الجمود وهي التالية:

- السماح لأربعة فلاسفة على الأكثر بالجلوس في نفس الوقت على الطاولة.
- لا تسمح للفيلسوف بأخذ عيدان تناول الطعام إلا إذا كانت كلتا العيدان متاحتين (وللقيام بذلك، يجب عليها أخذهما في قسم حرج).
- استخدم حلاً غير متماثل–أي أن الفيلسوف ذي الرقم الفردي يلتقط أولاً عيدان تناول الطعام اليسرى ثم عيدان تناول الطعام اليمنى، بينما الفيلسوف ذي الرقم الزوجي تلتقط عيدان تناول الطعام اليمنى ثم عيدان تناول الطعام اليسرى.

في القسم 6.7، نقدم حلاً لمشكلة الفلاسفة في تناول الطعام يضمن التحرر من الجمود. ومع ذلك، لاحظ أن أي حل مرضٍ لمشكلة الفلاسفة في تناول الطعام يجب أن يحمي من احتمالية موت أحد الفلاسفة جوعًا. فالحل الخالي من الجمود لا يلغي بالضرورة احتمالية الموت جوعًا.

الحلاق النائم

يوجد محل حلاقة به حلاق واحد وعدد من الكراسي للعملاء المنتظرين. يصل العملاء في أوقات عشوائية وإذا كان هناك كرسي متاح، فإنهم يجلسون وينتظرون أن يصبح الحلاق متاحًا. إذا لم تكن هناك كراسي متاحة، يغادر العميل. عندما ينتهي الحلاق من العمل مع العميل، يتحقق مما إذا كان هناك أي عملاء ينتظرون. إذا كان هناك، يبدأ في قص شعر العميل التالي في قائمة الانتظار. إذا لم يكن هناك عملاء ينتظرون، فإنه ينام.

المشكلة هي كتابة برنامج ينسق تصرفات العملاء والحلاق بطريقة تتجنب مشاكل المزامنة، مثل الجمود أو المجاعة.

أحد الحلول لمشكلة الحلاق النائم هو استخدام السيمافورات لتنسيق الوصول إلى كراسي الانتظار وكرسي الحلاقة. يتضمن الحل الخطوات التالية:

- قم بتشغيل سيمافورين: واحد لعدد كراسي الانتظار وآخر لكرسي الحلاقة. يتم تهيئة سيمافور كراسي الانتظار بعدد الكراسي، ويتم تهيئة سيمافور كرسي الحلاقة إلى الصفر.
- يجب على العملاء الحصول على سيمافور كراسي الانتظار قبل الجلوس في غرفة الانتظار. إذا لم تكن هناك كراسي متاحة، فيجب عليهم المغادرة.

- عندما ينتهي الحلاق من قص شعر العميل، فإنه يطلق سيمافور كرسي الحلاقة ويتحقق مما إذا كان هناك أي عملاء ينتظرون. إذا كان هناك، فإنه يحصل على سيمافور كرسي الحلاقة ويبدأ في قص شعر العميل التالي في قائمة الانتظار.

إذا افترضنا أن لدينا صالون حلاقة به حلاق واحد وكرسي حلاقة واحد وعدد من الكراسي للانتظار. إذا لم يكن هنالك زبائن سيجلس الحلاق في كرسي الحلاقة وينام، وعندما يصل زبون فسيقوم بإيقاظ الحلاق. إذا جاء زبون آخر وكان الحلاق يحلق للزبون الأول فسيجلس الزبون الثاني على أحد كراسي الانتظار. وكلما يصل زبون سيجلس في كرسي انتظار إلى أن تمتلي كراسي الانتظار. إذا جاء زبون ووجد كراسي الانتظار مشغولة فعليه مغادرة الصالون.

الحل.

لحل هذه المشكلة نستخدم ثلاث سيمافورات:

سيمافور للزبائن المنتظرين

سيمافور للحلاق (لنعرف هل هو نائم (عاطل) أم يعمل)

سيمافور لتحقيق والتأكد من المنع المتبادل (:)mutual exclusionبحيث لا نسمح لشخصين

بالحلاقة في وقت واحد.

يجب على الحلاق الانتظار على كرسي الحلاقة إذا لم يكن هناك عملاء ينتظرون.

يضمن الحلاق عدم قيام الحلاق بقص شعر أكثر من عميل واحد في وقت واحد، وانتظار العملاء إذا كان الحلاق مشغولاً. كما يضمن أن ينام الحلاق إذا لم يكن هناك عملاء ينتظرون.

ومع ذلك، هناك اختلافات في المشكلة قد تتطلب آليات مزامنة أكثر تعقيدًا لتجنب مشكلات المزامنة. على سبيل المثال، إذا تم توظيف العديد من الحلاقين، فقد تكون هناك حاجة إلى آلية أكثر تعقيدًا لضمان عدم تداخلهم مع بعضهم البعض.

كل ما يحضر زبون سيحاول الحصول على كرسي الحلاقة ( )mutex وسيظل كذلك حتى ينجح. على الزبون الذي يحضر للصالون أن يحسب عدد الزبائن المنتظرين فإذا كان أقل من عدد الكراسي فسيجلس وإلا فسيغادر (يحاول الحصول على كرسي سواء في غرفة الانتظار أو كرسي الحلاق نفسه.)

إذا وجد الزبون كرسي سيجلس وينقص عدد الكراسي الفارغة بواحد (.)critical section يقوم الزبون بإرسال إشارة إلى الحلاق لإيقاظه، وسيحرر mutex للسماح للزبائن (أو الحلاق) بالمقدرة على الحصول عليه. إذا كان الحلاق مشغول فعلى الزبائن الانتظار. سيجل الحلاق في انتظار دائم، يتم ايقاظه بأي زبون من المنتظرين، وعندما يستيقظ سيرسل إشارات للزبائن المنتظرين بواسطة السيمافور للسماح لهم بالحلاقة ، واحد كل مرة.

هذه المشكلة تحتوي على حلاق واحد لذلك تسمى أحيانا (.)single sleeping barber problemفيما يلي شفرات مبدئية (،)pseudo-codeتضمن التزامن بين الحلاق والزبائن خالية من الجمود، ولكنها قد تقود إلى حرمان الزبون أي المجاعة.

نجد P و Vهما دالتين توفرهما السيمافور

```
Semaphore Customers = 0;
Semaphore Barber = 0;
Mutex Seats = 1;
int FreeSeats = N;
```

عملية أو مسلك الحلاق

```
Barber {
       while(true) {
              /* waits for a customer (sleeps). */
              down(Customers);

    /* mutex to protect the number of available seats.*/
              down(Seats);

              /* a chair gets free.*/
              FreeSeats++;

              /* bring customer for haircut.*/
              up(Barber);

              /* release the mutex on the chair.*/
              up(Seats);
              /* barber is cutting hair.*/
       }
}
```

عملية أو مسلك الزبون

```
Customer {
      while(true)
{
     /* protects seats so only 1 customer tries to sit
in a chair if that's the case.*/
            down(Seats); //This line should not be here.
            if(FreeSeats > 0) {
                  /* sitting down.*/
                  FreeSeats--;
                 /* notify the barber. */
                  up(Customers);
                /* release the lock */
                  up(Seats);
/* wait in the waiting room if barber is busy. */
                  down(Barber);
                  // customer is having hair cut
            } else {
                  /* release the lock */
                  up(Seats);
                  // customer leaves
            }
      }
}
```