جامعة المنارة

كلية: .....الهندسة.....

قسم: ..... الهندسة المعلوماتية.....

اسم المقرر: ..... نظم تشغيل 1.....

رقم الجلسة (...7...)

عنوان الجلسة

....السيمافور و المراقب في نظم التشغيل.....

الفصل الدراسي العام الدراسي 2024 /2025

جدول المحتويات

Contents

الغاية من الجلسة: تعريف الطلاب باستخدام اشارات السيمافور و المراقب في نظم التشغيل من أجل التغلب على مشاكل العمليات المتزامنة في المنطقة الحرجة ومشكلة المنتج و المستهلك، مشكلة القارئ و الكاتب

مزامنة العمليات وإدارة المنطقة الحرجة هما مكونان أساسيان في تصميم أنظمة التشغيل والتطبيقات التي تتعامل مع تعدد العمليات (Processes) أو الخيوط.(Threads) .

1-مزامنة العمليات

مزامنة العمليات هي العملية التي يتم من خلالها تنسيق العمليات المختلفة لضمان أن تعمل معًا بدون تعارض، خاصة عندما تشترك في موارد النظام.

الآليات المستخدمة لمزامنة العمليات:

1. السيمافور :(Semaphore) أداة قوية لإدارة التزامن.
2. الأساليب القائمة على القفل :(Lock-based mechanisms) مثل الأقفال (Locks) والمتغيرات الشرطية (Condition Variables).
3. المراقب :(Monitor) يوفر مراقبة ذاتية للتزامن ويحتوي على قفل داخلي ومتغيرات شرطية.
4. الحواجز :(Barriers) تستخدم لتنظيم نقطة التزامن حيث تتوقف كل العمليات وتنتظر أن يصل الجميع إلى نقطة محددة قبل المتابعة

2-المنطقة الحرجة(Critical Section)

المنطقة الحرجة هي جزء من الكود الذي يمكن أن يؤدي إلى مشاكل إذا تم تشغيله بواسطة أكثر من عملية أو خيط في نفس الوقت. تحتاج المنطقة الحرجة إلى حماية لضمان أن عملية واحدة فقط تقوم بتنفيذها في وقت واحد.

القواعد لضمان سلامة المنطقة الحرجة:

1. المنع المتبادل :(Mutual Exclusion) يجب ضمان أن عملية واحدة فقط تستطيع الدخول إلى المنطقة الحرجة في أي وقت.

2. التقدم :(Progress) إذا لم تكن أي عملية في المنطقة الحرجة، وأرادت عملية الدخول إليها، فيجب أن تستطيع الدخول بدون تأخير غير مبرر.

3. الانتظار المحدود :(Bounded Waiting) يجب وضع حد على عدد المرات التي يمكن لعملية أخرى الدخول إلى المنطقة الحرجة قبل أن يتم منح الدخول للعملية التي تنتظر.

3-حالة السباق Race Condition:

تحدث حالة سباق عندما تقوم عمليات أو خيوط متعددة بالوصول إلى الموارد المشتركة وتعديلها في وقت واحد، وتعتمد النتيجة النهائية على الترتيب غير المتوقع للتنفيذ.

الموارد المشتركة: هذه هي الموارد مثل الملفات أو المتغيرات أو مكونات الأجهزة التي يمكن الوصول إليها من خلال عمليات/خيوط متعددة .الوصول المتزامن: تحاول عمليات أو خيوط متعددة تعديل المورد المشترك في وقت واحد .الترتيب غير المتوقع: التسلسل الدقيق الذي يتم به تنفيذ هذه العمليات/الخيوط غير حتمي، مما يعني أنه يمكن أن يختلف في كل مرة .نتيجة غير صحيحة: بسبب الترتيب غير المتوقع للتنفيذ، قد تكون الحالة النهائية للمورد المشترك غير متسقة أو غير صحيحة.

مثال: لنفترض وجود نظام مصرفي بسيط تحاول فيه عمليتان سحب الأموال من نفس الحساب :العملية **P1** :قراءة رصيد الحساب (لنقل 100 دولار) .خصم المبلغ المراد سحبه (لنقل 50 دولارًا) .كتابة الرصيد الجديد (50 دولارًا) مرة أخرى في الحساب .العملية **P2** :قراءة رصيد الحساب (100 دولار) .خصم المبلغ المراد سحبه (50 دولارًا) .كتابة الرصيد الجديد (50 دولارًا) مرة أخرى في الحساب .**حالة السباق** :إذا تم تشغيل العمليتين **P1** و**p2** في نفس الوقت، فإن السيناريو التالي ممكن: تقرأ العملية**P1** الرصيد (100 دولار) .تقرأ العملية **P2** الرصيد (100 دولار) .تكتب العملية **P2** الرصيد الجديد (50 دولارًا) . تكتب العملية **P1** الرصيد الجديد (50 دولارًا) .النتيجة: سيكون الرصيد النهائي في الحساب 50 دولارًا، بينما يجب أن يكون 0 دولارًا (نظرًا لأن العمليتين سحبتا 50 دولارًا) .يرجع ذلك إلى أن ترتيب التنفيذ غير متوقع، مما يؤدي إلى نتيجة غير صحيحة.

3-1-حلول لمنع حالةالسباق:

آليات المزامنة: استخدام آليات مثل أدوات المزامنة المتبادلة mutexes أو السيمافور semaphores أو أجهزة المراقبة لضمان أن عملية/خيط واحد فقط يمكنه الوصول إلى المورد المشترك في كل مرة.

3-1-1-السيمافور.(Semaphore)

Semaphore (السيميافور) هو مفهوم أساسي في علوم الكمبيوتر، يُستخدم للتحكم في الوصول إلى الموارد المشتركة في بيئة متعددة العمليات أو الخيوط (threads) من خلال مزامنة العمليات. يُعتبر السيما فور واحداً من آليات التزامن الأساسية في نظم التشغيل.

1-تعريف السيميافور

السيما فور هو متغير أو كائن يُستخدم للتحكم في الوصول إلى مورد مشترك من قبل عمليات متعددة في نظام تشغيل متعدد المهام. يمكن استخدامه لمنع الشروط المتسارعة والتأكد من أن العمليات لا تتداخل مع بعضها البعض.

2- أنواع السيمافور

- **السيمافور العددي :(Counting Semaphore)** هي نوع من **السيما فور الثنائي** التي تسمح لعدد محدود من العمليات أو الخيوط بالوصول إلى مورد مشترك في وقت واحد. إنها شكل أكثر عمومية من **السيما فور الثنائي** ، والتي تسمح فقط لعملية واحدة أو خيط واحد بالوصول إلى المورد في وقت واحد.
  - ✓ التهيئة: يتم تهيئة السيما فور العددي بقيمة، تُعرف باسم count، والتي تحدد الحد الأقصى لعدد العمليات أو الخيوط التي يمكنها الوصول إلى المورد المشترك في وقت واحد.
  - ✓ الاستحواذ: عندما تريد عملية أو خيط الوصول إلى المورد المشترك، تحاول الاستحواذ على السيمافور. إذا كانت قيمة السيمافور أكبر من 0، يمكن للعملية/الخيط الوصول إلى المورد ويتم تقليل قيمة السيمافور بمقدار 1. إذا كانت قيمة السيمافور 0، يتم حظر العملية/الخيط حتى يتم تحرير السيمافور.
  - ✓ التحرير: عندما تنتهي عملية/سلسلة من استخدام المورد المشترك، فإنها تقوم بتحرير السيمافور، مما يزيد قيمة السيمافور بمقدار 1. وهذا يشير إلى العمليات/الخيوط الأخرى أن المورد أصبح متاحًا مرة أخرى.
- **السيما فور الثنائي :(Binary Semaphore)** يمكن أن يأخذ فقط القيمتين 0 و1، وهو يُستخدم للتحكم في الوصول إلى مورد يمكن استخدامه مرة واحدة فقط في الوقت نفسه (مثل القفل).
  - ✓ التهيئة: يتم تهيئة السيمافور الثنائي بقيمة 1، مما يشير إلى أن المورد المشترك متاح.
  - ✓ الاستحواذ: عندما تريد عملية أو خيط الوصول إلى المورد المشترك، تحاول الاستحواذ على السيمافور. إذا كانت قيمة السيمافور 1، يمكن للعملية/الخيط الوصول إلى المورد ويتم تعيين قيمة السيمافور على 0. إذا كانت قيمة السيمافور 0 بالفعل، يتم حظر العملية/الخيط حتى يتم تحرير السيمافور.
  - ✓ التحرير: عندما تنتهي عملية/خيط من المورد المشترك، فإنها تحرر السيمافور، وتعيد تعيين قيمتها إلى 1. وهذا يشير إلى العمليات/الخيوط الأخرى بأن المورد متاح مرة أخرى.

3-آلية عمل السيمافور:

- **إشارة :(Signal)** تُستخدم لزيادة قيمة السيما فور. عندما يتم تنفيذ عملية الإشارة، فهذا يشير إلى أن عملية قد انتهت من استخدام المورد أو أنه تم تحرير المورد.
- **انتظار :(Wait)** تُستخدم لإنقاص قيمة السيما فور. عندما يتم تنفيذ عملية الانتظار، فهذا يعني أن العملية تحاول الوصول إلى المورد، وإذا كانت القيمة صفرًا، فستنتظر العملية حتى يصبح المورد متاحًا.

مثال بسيط باستخدام لغة C :

هذا الكود ينشئ ثلاثة خيوط، وكل خيط ينتظر حتى يصبح السيمافور متاحًا، ثم يدخل القسم الحرج، يقوم بطباعة رسالة، وأخيرًا يحرر السيمافور. يتم استخدام السيمافور هنا لضمان أن خيطًا واحدًا فقط يمكنه الدخول إلى القسم الحرج في نفس الوقت.

```c
#include <stdio.h>
 2  #include <pthread.h>
 3  #include <semaphore.h>
 4
 5  sem_t semaphore;
 6
 7  void* thread_function(void* arg) {
 8      sem_wait(&semaphore); // الانتظار للحصول على المورد
 9      // القسم الحرج
10      printf("Thread %d is in the critical section.\n", *(int*)arg);
11      sem_post(&semaphore); // تحرير المورد
12      return NULL;
13  }
14
15  int main() {
16      pthread_t threads[3];
17      int thread_ids[3] = {1, 2, 3};
18
19      sem_init(&semaphore, 0, 1); // 1 تهيئة السيمي فور بقيمة
20
21      for (int I = 0; I < 3; i++) {
22          pthread_create(&threads[i], NULL, thread_function, &thread_ids[i]);
23      }
24
25      for (int I = 0; I < 3; i++) {
26          pthread_join(threads[i], NULL);
27      }
28
29      sem_destroy(&semaphore); // تدمير السيمي فور
30      return 0;
31  }
```

stdio.h مكتبة الإدخال والإخراج القياسية في C.

semaphore.h مكتبة لمعالجة السيما فور.(semaphores)

تعريف متغير من نوع sem_t يسمى semaphore، وهو السيما فور الذي سنستخدمه لمزامنة الوصول إلى القسم الحرج.

sem_wait(&semaphore) تنتظر حتى يصبح السيما فور متاحًا (قيمة أكبر من الصفر)، ثم تقلل من قيمته بمقدار واحد.

printf("Thread %d is in the critical section.\n", *(int*)arg) تطبع رسالة تشير إلى أن الخيط قد دخل القسم الحرج.

sem_post(&semaphore) تزيد قيمة السيميافور بمقدار واحد، مما يسمح للخيوط الأخرى بالدخول إلى القسم الحرج

pthread_t threads[3] تعريف مصفوفة من ثلاث خيوط.

int thread_ids[3] = {1, 2, 3} تعريف مصفوفة من ثلاث معرفات للخيوط.

sem_init(&semaphore, 0, 1) تهيئة السيما فور بقيمة ابتدائية 1. القيمة 0 تشير إلى أن السيما فور مشترك بين العمليات في نفس الذاكرة (داخل نفس العملية).

pthread_create(&threads[i], NULL, thread_function, &thread_ids[i]) إنشاء ثلاثة خيوط، كل خيط يستدعي thread_function مع تمرير معرّف الخيط كمعامل.

pthread_join(threads[i], NULL) انتظار انتهاء كل خيط قبل مواصلة تنفيذ البرنامج الرئيسي

sem_destroy(&semaphore) تدمير السيمي فور لتحرير الموارد.

return 0; إنهاء البرنامج بنجاح.

المراقب Monitor هي بنية مزامنة تستخدم في أنظمة التشغيل لإدارة الوصول إلى الموارد المشتركة من خلال عمليات أو خيوط متعددة. وهي تساعد في منع حالات السباق، مما يضمن أن خيطًا واحدًا فقط يمكنه الوصول إلى مورد في كل مرة مع توفير تجريد أعلى مستوى من المزامنة المتبادلة أو السيمافور. قد تكون البيانات داخل Monitor إما عالمية لجميع الاجرائيات داخل Monitor أو محلية لإجراء معين. أي طريقة ضمن المراقب لا يمكنها الوصول إلى أي بيانات خارج المراقب.

يجمع المراقب بين ثلاث ميزات:

- البيانات المشتركة.
- العمليات على البيانات.
- المزامنة والجدولة.

وهي ملائمة بشكل خاص للمزامنة التي تتضمن الكثير من الحالات.

المفاهيم الأساسية للمراقب:

- التغليف Encapsulation: يغلف المراقب المتغيرات المشتركة والإجراءات التي يعمل عليها، مما يضمن أنه لا يمكن تغيير الحالة المشتركة إلا بطريقة خاضعة للرقابة. لا يمكن الوصول إلى هذه المتغيرات إلا من خلال الإجراءات المحددة داخل المراقب.
- الإجراءات Procedure: تحدد المراقبين الأساليب (أو الإجراءات) التي تعمل على المتغيرات المشتركة. هذه الإجراءات هي الوسيلة الوحيدة التي يمكن للخيوط من خلالها التفاعل مع البيانات المشتركة.

- الاستبعاد المتبادل Mutual Exclusion: لا يمكن أن يكون سوى خيط واحد نشطًا في المراقب في أي وقت. يتم تحقيق ذلك من خلال استخدام الأقفال locks.
- المتغيرات الشرطية Condition Variables: غالبًا ما يتضمن المراقب متغيرات شرطية تسمح للخيوط بالانتظار حتى تصبح شروط معينة صحيحة قبل المتابعة. وهذا مفيد في السيناريوهات التي يحتاج فيها الخيط إلى الانتظار حتى يصبح المورد متاحًا.

يمكن إجراء عمليتين على المتغيرات الشرطية wait و signal

(Wait(condition: تحرير قفل المراقب، ووضع العملية في وضع السكون. وعندما تستيقظ العملية مرة أخرى، تكتسب قفل المراقب على الفور.

(Signal(condition: ايقاظ عملية واحدة قيد الانتظار على المتغير الشرطي (FIFO)، إذا لم يكن هناك أي عملية قيد الانتظار، فلا تفعل شيئًا

(Broadcast(condition: ايقاظ جميع العمليات التي تنتظر على المتغير الشرطي. إذا لم يكن هناك أي عملية قيد الأنتظار ينتظر، فلا تفعل شيئًا.

- القفل التلقائي Automatic Locking: عندما يدخل الخيط إلى المراقب، فإنه يكتسب قفل المراقب، وعندما يخرج، يتم تحرير القفل تلقائيًا.

بنية المر اقب Monitor Structure:

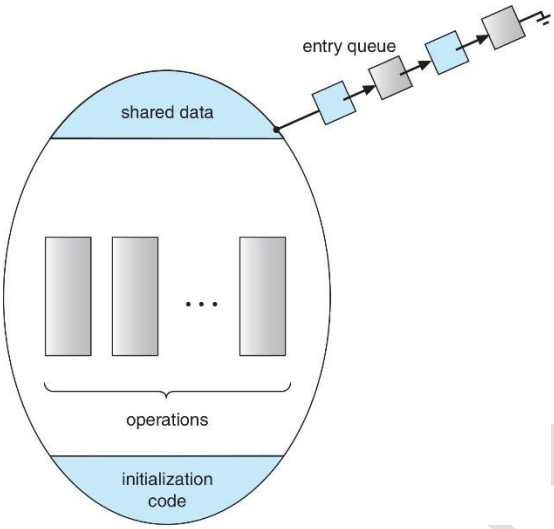
قد تتضمن بنية المراقب النموذجية ما يلي:

- المتغيرات المشتركة: وهي الموارد المشتركة بين الخيوط.
- الإجراءات: الوظائف التي تعمل على المتغيرات المشتركة.
- المتغيرات الشرطية: تستخدم للإشارة بين الخيوط.

يمكن الوصول إلى بيانات Monitor فقط داخل Monitor. يمكن حماية بنية البيانات المشتركة عن طريق وضعها في Monitor. إذا كانت البيانات الموجودة في Monitor تمثل بعض الموارد، فإن Monitor يوفر مرفق استبعاد متبادل للوصول إلى المورد. يمكن للإجراء المحدد داخل المراقب الوصول فقط إلى تلك المتغيرات المعلنة محليًا داخل المراقب ومعامِلاتها الرسمية.

عندما تستدعي عملية إجراء مراقب، فإن التعليمات القليلة الأولى للإجراء ستتحقق لمعرفة ما إذا كانت هناك أي عملية أخرى نشطة حاليًا داخل المراقب. إذا كانت العملية نشطة، فسيتم تعليق العملية المستدعاة حتى تغادر العملية الأخرى المراقب. إذا لم تكن هناك عملية أخرى تستخدم المراقب، فيمكن للعملية المستدعاة الدخول.

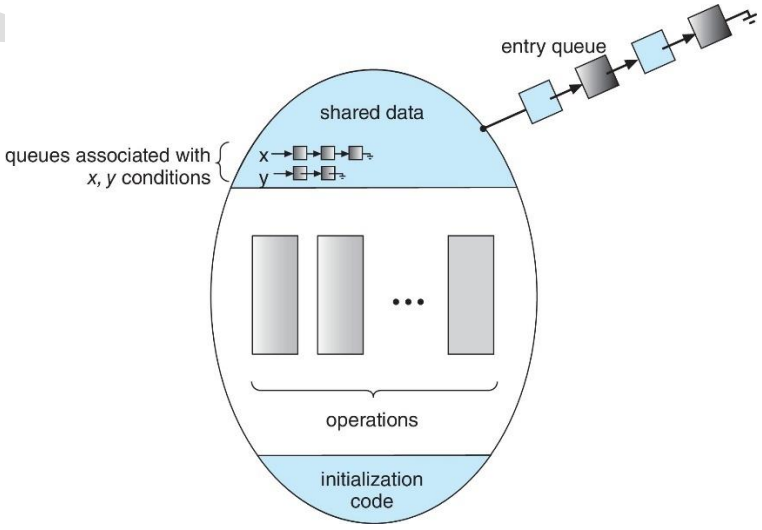
الشكل التالي يُظهر مخططًا للمراقب، مع قائمة إدخال للعمليات التي تنتظر دورها لتنفيذ عمليات المراقب (الطرق methods).



يدعم Monitor المزامنة باستخدام المتغيرات الشرطية الموجودة داخل Monitor والتي يمكن الوصول إليها فقط من داخل Monitor. كل متغير شرطي له قائمة انتظار مرتبطة به. تعمل المتغيرات الشرطية على اثنين، من أجل تحقيق إمكانات المراقبين بالكامل، نحتاج إلى تقديم نوع بيانات جديد إضافي، يُعرف باسم الشرط condition. يحتوي المتغير من النوع الشرطي على عمليتين قانونيتين فقط، الانتظار wait والإشارة signal . أي إذا تم تعريف X من النوع الشرطي، فإن العمليات القانونية ستكون X.wait() و X.signal()

- تمنع عملية الانتظار العملية حتى تستدعي عمليات أخرى الإشارة signal، وتضيف العملية المحظورة إلى قائمة مرتبطة بهذا الشرط.
- لا تفعل عملية الإشارة signal أي شيء إذا لم تكن هناك عمليات تنتظر هذا الشرط. وإلا فإنها توقظ عملية واحدة فقط من قائمة العمليات المنتظرة للشرط.

يوضح الشكل التالي أدناه مراقبًا يتضمن متغيرات شرط داخل مساحة البيانات الخاصة به. لاحظ أن متغيرات الشرط، إلى جانب قائمة العمليات التي تنتظر حاليًا الشروط، موجودة في مساحة البيانات الخاصة بالمراقب - والعمليات الموجودة في هذه القوائم ليست "داخل" المراقبة، بمعنى أنها لا تنفذ أي تعليمات برمجية في المراقبة.



ولكن الآن هناك مشكلة محتملة - إذا أصدرت العملية P داخل المراقب إشارة signal من شأنها إيقاظ العملية Q أيضًا داخل المراقب، فسيكون هناك عمليتان تعملان في وقت واحد داخل المراقب، مما ينتهك متطلب الاستبعاد. وفقًا لذلك، هناك حلين محتملين لهذه المعضلة:

- الإشارة signal والانتظار wait - عندما تصدر العملية P الإشارة لإيقاظ العملية Q، تنتظر P بعد ذلك، إما حتى تخرج Q من المراقب أو في حالة أخرى.
- الإشارة signal والاستمرار continue - عندما تصدر P الإشارة، تنتظر Q، إما حتى تخرج P من المراقب أو في حالة أخرى.

هناك بارامترات لصالح وضد أي من الخيارين. يقدم باسكال المتزامن بديلاً ثالثًا - يتسبب استدعاء الإشارة signal في خروج العملية على الفور من المراقب، بحيث يمكن لعملية الانتظار بعد ذلك الاستيقاظ والمتابعة.

مثال: استخدام المتغير الشرطي :

لنفترض أن P1 وP2 يحتاجان إلى تنفيذ الحالتين S1 وS2 والمطلب حدوث S1 قبل S2، أنشئ مراقبًا بإجراءين F1 وF2 يتم استدعاؤهما بواسطة P1 وP2 على التوالي

- متغير شرطي واحد "x" تم تهيئته إلى 0
- متغير منطقي واحد "done"

```
1  F1:
2     S1;
3     done = true;
4     x.signal();
5  F2:
6     if done = false
7        x.wait()
8     S2;
```

- ✓ S1: هذا عنصر نائب لعملية يقوم بها الخيط (threat 1). يمكن أن تكون هذه أي مهمة، مثل إنتاج البيانات أو معالجة عنصر، إلخ.
- ✓ done = true: يضبط هذا السطر علامة (done) على true، مما يشير إلى أن العملية التي يمثلها S1 قد اكتملت. تعمل هذه العلامة كشرط للخيط الآخر (threat 2) لمعرفة ما إذا كان يمكنه المتابعة.
- ✓ x.signal: يشير هذا التابع إلى المتغير الشرطي x. إنها توقظ أحد الخيوط التي قد تكون في انتظار متغير الشرط هذا، مما يسمح لها باستئناف التنفيذ.
- ✓ إذا done = false: يتحقق هذا الشرط من حالة علامة done. إذا كانت false، فسينفذ الخيط (threat 2) طريقة wait.
- ✓ x.wait: يضع هذا التابع الخيط في حالة انتظار، ويحرر القفل المرتبط بالمتغير الشرطي x. سيظل الخيط في حالة الانتظار هذه حتى يتم الإشارة إليه بواسطة خيط آخر (في هذه الحالة، threat 1) مما يشير إلى أنه يمكنه المتابعة (عندما يصبح done = true).

✓ S2: بمجرد استيقاظ الخيط بنجاح (بمعنى أن علامة done تم تعيينها على true)، فسوف ينفذ العملية التي يمثلها S2، مثل استهلاك البيانات.

منطق المزامنة

تفاعلات الخيوط threats:

- ينفذ الخيط 1 عمليته، ويضبط علامة "done" على "true"، ويرسل إشارة إلى الخيط 2 للاستيقاظ.
- يتحقق الخيط 2 من حالة "done" قبل المتابعة. إذا كانت حالة "done" تساوي false ، فإنه ينتظر الإشارة من الخيط 1.

استخدام متغيرات الشرط:

- يساعد هذا النمط في منع الانتظار المشغول busy waiting، حيث يتحقق الخيط threat باستمرار من شرط ما. بدلاً من ذلك، سوف ينام threat 2 ولا يستيقظ إلا عند إخطاره، مما يجعل استخدام موارد وحدة المعالجة المركزية أكثر كفاءة.

فيما يلي مثال بسيط لمراقب يتحكم في الوصول إلى عداد مشترك: يحدد هذا الكود مراقب يسمى Counter ، والذي ينفذ عدادًا آمنًا للخيوط

```
monitor Counter {
    int count = 0; // Shared variable
    condition not_empty;

    procedure increment() {
        count = count + 1;
    }

    procedure decrement() {
        while (count == 0) {
            wait(not_empty); // Wait until count is not zero
        }
        count = count - 1;
    }

    procedure getCount() {
        return count;
    }
}
```

شرح المثال

المتغير المشترك: يتم مشاركة متغير count بين جميع الخيوط التي تستخدم مراقب العداد.

الإجراءات:

- increment: يزيد من العدد.
- decrement: يقلل من العدد ولكنه ينتظر إذا كان العدد صفرًا.

- getCount: يعيد القيمة الحالية للعدد.
- المتغير الشرطي: يسمح المتغير الشرطي not_empty لإجراء التناقص بالانتظار حتى يكون هناك عدد موجب.
- while (count == 0) { ... } تضمن هذه الحلقة عدم استمرار عملية التناقص إذا كان العداد صفرًا بالفعل. وهذا يمنع الأخطاء المحتملة مثل القيم السلبية.
- wait(not_empty); إذا كان العداد صفرًا، ينتظر الخيط على متغير الشرط .not_empty وهذا يحظر الخيط حتى يرسل خيط آخر إشارة signal إلى متغير الشرط.

عندما تستدعي خيوط متعددة ()increment أو ()decrement في نفس الوقت، يمكن لخيط واحد فقط الوصول إلى متغير count المشترك في كل مرة بسبب خاصية الاستبعاد المتبادل للمراقب.

- إذا استدعى خيط ما ()decrement ووجد أن العداد يساوي صفرًا، فسوف ينتظر على متغير الشرط.not_empty
- عندما يستدعي خيط آخر()increment ، فإنه يزيد العداد ثم يشير إلى متغير الشرط.not_empty

تعمل هذه الإشارة على إيقاظ الخيط المنتظر من حالة الانتظار، مما يسمح له بمواصلة عملية.()decrement

مزايا المراقب

- البساطة: توفر تجريدًا أعلى مستوى من بدائيات المزامنة ذات المستوى الأدنى، مما يجعل الكود أسهل في الفهم.

المعالجة