جامعة المنارة

كلية: .....الهندسة.....

قسم: ..... الهندسة المعلوماتية.....

اسم المقرر: ..... نظم تشغيل 1.....

رقم الجلسة (...1...)

عنوان الجلسة

....نظام التشغيل .....

م.عمار مصطفى

الفصل الدراسي العام الدراسي 2025/ 2024

جدول المحتويات

Contents

الغاية من الجلسة: حل بيترسون (Peterson's Solution) لمزامنة العمليات

مسائل حل بيترسون (Peterson's Solution) لمزامنة العمليات هي أمثلة كلاسيكية تُستخدم لتوضيح كيفية التعامل مع مشكلة الوصول إلى الموارد المشتركة في بيئة متعددة العمليات (multiprocessing) أو متعددة الخيوط (multithreading). تهدف خوارزمية بيترسون إلى ضمان أن عملية واحدة فقط يمكنها الوصول إلى المنطقة الحرجة (Critical Section) في وقت معين، مع تحقيق شروط المزامنة: الاستبعاد المتبادل(Mutual Exclusion) ، التقدم (Progress)، والانتظار المحدود.(Bounded Waiting)

المسألة الأولى: فهم أساسي لخوارزمية بيترسون

السؤال:

افترض أن لديك عمليتين P0 و P1 تشتركان في مورد مشترك (مثل متغير مشترك). اكتب كود خوارزمية بيترسون لضمان الاستبعاد المتبادل عند الوصول إلى المنطقة الحرجة. اشرح كيف يتم تحقيق الشروط الثلاثة (الاستبعاد المتبادل، التقدم، الانتظار المحدود).

الحل:

خوارزمية بيترسون تستخدم متغيرين مشتركين:

turn يحدد أي عملية لها الأولوية لدخول المنطقة الحرجة .

interested مصفوفة منطقية تُشير إلى ما إذا كانت العملية تريد دخول المنطقة الحرجة.

```
#include <stdbool.h>
#define N 2 // عدد العمليات
bool interested[N] // ; مصفوفة لتتبع رغبة العمليات
int turn // ; متغير يحدد أي عملية لها الأولوية
void enter_region(int process){
    int other = 1 - process // ; العملية الأخرى
    interested[process] = true // ; الإشارة إلى رغبة العملية
    turn = other // ; إعطاء الأولوية للعملية الأخرى
    while (turn == other && interested[other])}
```

```
// الانتظار حتى تسمح الشروط بدخول المنطقة الحرجة
    }
}

void leave_region(int process){
    interested[process] = falseالحرجة // ; الإشارة إلى انتهاء العملية من المنطقة
}
void process(int id) {
    enter_region(id);
// المنطقة الحرجة: يتم تنفيذ العمليات الحساسة هنا
    printf("Process %d is in critical section\n", id);
    leave_region(id);
                                                    }
```

تحليل الشروط:

1. **الاستبعاد المتبادل (Mutual Exclusion):**
   - يتم تحقيقه لأن العملية تدخل المنطقة الحرجة فقط إذا لم تكن العملية الأخرى مهتمة (interested[other] == false) أو إذا لم يكن دورها (turn != other).
   - إذا حاولت كلتا العمليتين الدخول في نفس الوقت، فإن متغير turn يضمن أن واحدة فقط تدخل، بينما تنتظر الأخرى.
2. **التقدم (Progress):**
   - إذا لم تكن العملية الأخرى مهتمة بدخول المنطقة الحرجة (interested[other] == false) ، فإن العملية الحالية تدخل مباشرة دون انتظار، مما يضمن عدم وجود تعطيل غير ضروري.
3. **الانتظار المحدود (Bounded Waiting):**
   - بما أن متغير turn يتم تبديله بين العمليتين، فإن أي عملية تنتظر لن تظل في حالة انتظار إلى أجل غير مسمى، لأن العملية الأخرى ستغادر المنطقة الحرجة في النهاية وتضع interested[other] = false.

المسألة الثانية: تحليل سيناريو تنفيذ

السؤال:

افترض أن العملية P0 تريد دخول المنطقة الحرجة، ثم بعد ذلك مباشرة تريد العملية P1 الدخول. باستخدام خوارزمية بيترسون، اشرح خطوة بخطوة كيف يتم تنفيذ العمليتين. افترض أن القيم الابتدائية هي: interested[0] = false, interested[1] = false, turn = 0.

الحل:

1. **الخطوة 1 P0 تدخل: enter_region(0):**
   - P0 تضع interested[0] = true.
   - P0 تضع turn = 1 (تعطي الأولوية لـP1).
   - P0 تتحقق من الحلقة: interested[1] && turn == 1
   - بما أن interested[1] = false (لأن P1 لم تبدأ بعد)، فإن P0 تدخل المنطقة الحرجة مباشرة.
2. **الخطوة 2 P0 في المنطقة الحرجة:**
   - P0 تنفذ عملياتها في المنطقة الحرجة.
3. **الخطوة 3 P1 تدخل enter_region(1) بينما P0 في المنطقة الحرجة:**
   - P1 تضع interested[1] = true.
   - P1 تضع turn = 0 (تعطي الأولوية لـP0).
   - P1 تتحقق من الحلقة: interested[0] && turn == 0
   - بما أن interested[0] = true (لأن P0 لا تزال في المنطقة الحرجة) و turn = 0، فإن P1 تنتظر في الحلقة.
4. **الخطوة 4 P0 تغادر المنطقة الحرجة:**
   - P0 تنفذ leave_region(0)، فتضع interested[0] = false.
   - الآن، عندما تعود P1 لتتحقق من الحلقة، تجد interested[0] = false، فتخرج من الحلقة وتدخل المنطقة الحرجة.
5. **الخطوة 5 P1 في المنطقة الحرجة:**
   - P1 تنفذ عملياتها، ثم تنفذ leave_region(1)، فتضع interested[1] = false.

النتيجة:

- P0 تدخل وتكمل أولاً، ثم تنتظر P1 حتى تنتهي P0 ، ثم تدخل P1. يتم تحقيق الاستبعاد المتبادل والانتظار المحدو

المسألة الثالثة: تحليل أداء خوارزمية بيترسون

السؤال:

افترض أن لديك ثلاث عمليات (P0، P1، P2) بدلاً من عمليتين. هل يمكن تعميم خوارزمية بيترسون لثلاث عمليات أو أكثر؟ إذا لم يكن ذلك ممكنًا، اشرح السبب وما هي البدائل.

الحل:

- **الإجابة** :خوارزمية بيترسون الأصلية صُممت لعمليتين فقط .(N=2) تعميمها لثلاث عمليات أو أكثر غير ممكن مباشرة لأن متغير turn يعتمد على اختيار عملية واحدة من بين اثنتين فقط. عند وجود أكثر من عمليتين، تصبح إدارة الأولوية (turn) وتتبع حالة الرغبة (interested) معقدة للغاية، ولا يمكن ضمان الاستبعاد المتبادل أو الانتظار المحدود بسهولة.
- **السبب** :
  - متغير turn يعمل فقط كمفتاح ثنائي بين عمليتين. لثلاث عمليات، يلزم نهج مختلف لتحديد أولوية العمليات.
  - الحلقة while (turn == other && interested[other]) لا يمكنها التعامل مع أكثر من عملية أخرى في نفس الوقت.
- **البدائل** :

1. **خوارزمية المخبز (Bakery Algorithm)**: تُستخدم لعدد N من العمليات، حيث تُعطي كل عملية رقمًا (ticket) وتدخل العملية ذات الرقم الأدنى إلى المنطقة الحرجة.
2. **السيمافورات (Semaphores)**: توفر آلية أكثر مرونة لمزامنة عدد كبير من العمليات.
3. **الأقفال (Locks)**: مثل أقفال المراقبة (Monitors) في أنظمة التشغيل الحديثة.

المسألة الرابعة: خطأ شائع في التطبيق

السؤال:

افترض أن أحد المبرمجين قام بتنفيذ خوارزمية بيترسون، لكنه عكس ترتيب السطرين في :enter_region

```
turn = other;
interested[process] = true;
```

بدلاً من:

```
interested[process] = true;
turn = other;
```

هل سيؤدي هذا التغيير إلى خرق الاستبعاد المتبادل؟ اشرح بالتفصيل.

الحل:

- **الإجابة** :نعم، عكس ترتيب السطرين قد يؤدي إلى خرق الاستبعاد المتبادل في بعض الحالات.
- **التحليل** :
  - في الخوارزمية الأصلية، تضع العملية interested[process] = true أولاً للإشارة إلى رغبتها، ثم تعطي الأولوية للعملية الأخرى باستخدام turn = other. هذا يضمن أن العملية الأخرى، إذا كانت مهتمة، يمكنها الدخول إذا كان دورها.
  - إذا تم عكس الترتيب (وضع turn = other أولاً)، فقد تحدث مشكلة في توقيت معين. لنفترض السيناريو التالي :
    1. P0 تنفذ turn = 1 (تعطي الأولوية لـP1).
    2. قبل أن تضع P0 interested[0] = true ، يتم تبديل السياق (context switch) إلى P1.
    3. P1 تنفذ turn = 0 و interested[1] = true.
    4. P1 تتحقق من الحلقة while (interested[0] && turn == 0) :، وتجد interested[0] = false (لأن P0 لم تضع interested[0] = true بعد)، فتدخل P1 المنطقة الحرجة.
    5. يُعاد السياق إلى P0 ، التي تضع interested[0] = true وتتحقق من الحلقة while (interested[1] && turn == 1) : بما أن turn = 0 الآن (بسبب P1 ، فإن P0 تدخل المنطقة الحرجة أيضًا.)
  - النتيجة :كلتا العمليتين (P0وP1) تدخلان المنطقة الحرجة في نفس الوقت، مما يخرق الاستبعاد المتبادل.
- الاستنتاج :ترتيب السطرين مهم جدًا لضمان أن العملية تعلن عن رغبتها قبل إعطاء الأولوية للعملية الأخرى، مما يمنع هذا النوع من السباق.(race condition)