جامعة المنارة

كلية: الهندسة

قسم: المعلوماتية

اسم المقرر: نظم تشغيل ٢

رقم الجلسة (٥)

عنوان الجلسة

استخدام المونيتور في نظم التشغيل

الفصل الدراسي

الأول

العام الدراسي

٢٠٢٣_٢٠٢٤

جدول المحتويات

Contents

الغاية من الجلسة:

سنقوم بهذه الجلسة العملية بالتعرف على استخدام المونيتور في نظم التشغيل و استخدامه كأداة مزامنة تنظم الوصول للتوزيعات المتعددة إلى الموارد المشتركة و الحفاظ على تماسك البيانات

Monitors

- المونيتور Monitor الموجودة في نظام التشغيل هي أداة مزامنة تسمح لسلاسل عمليات متعددة بالوصول إلى مورد مشترك بشكل متبادل فيما بينها ، ويتم تنفيذها كإجرائيات برمجية أنه يضمن تماسك البيانات في قسم التعليمات البرمجية الحرجة.، وتوفر الاستبعاد المتبادل، ويستخدم متغيرات الحالة، وتغليف البيانات في بنية واحدة
- تم تقديم مفهوم المونيتور بلغة البرمجة Concurrent Pascal بواسطة Per Brinch Hansen في عام ١٩٧٢. ومنذ ذلك الحين، تم تنفيذها بلغات برمجة مختلفة. تعد أجهزة المراقبة أدوات ديناميكية تساعد في إدارة الوصول المتزامن إلى الموارد المشتركة في نظام التشغيل. الوصول المتزامن يعني السماح لأكثر من مستخدم بالوصول إلى جهاز كمبيوتر في وقت واحد.

لماذا يتم استخدام المونيتور؟

تُستخدم أجهزة المونيتور في أنظمة التشغيل لإدارة الوصول إلى الموارد المشتركة، مثل الملفات أو البيانات، بين عمليات متعددة. فهي تضمن أن عملية واحدة فقط يمكنها استخدام المورد في وقت واحد، مما يمنع حدوث تعارضات وتلف البيانات. تعمل المونيتورعلى تبسيط المزامنة وحماية سلامة البيانات، مما يسهل على المبرمجين إنشاء برامج موثوقة.

تعتبر المونيتور "كحراس" لأقسام التعليمات البرمجية الهامة، مما يضمن عدم إمكانية إدخال عمليتين إليها في وقت واحد. تشبه أجهزة المراقبة إشارات المرور التي تتحكم في الوصول إلى الموارد، وتمنع الأعطال وتضمن التدفق السلس للبيانات والمهام في نظام التشغيل.

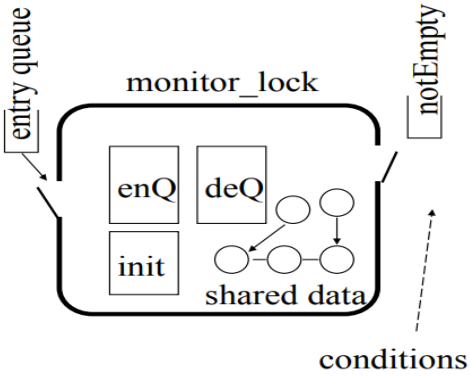
في الكود المماثل p1، p2، pn هي الإجراءات التي توفر الوصول إلى البيانات المشتركة والمتغيرات الموجودة في المونيتور تساعد على ضمان الوصول إلى مؤشر ترابط واحد فقط في كل مرة لاستخدام المورد المشترك. مما يمنع مشاكل حالة السباق والمزامنة.

تحدث حالة السباق عندما تتنافس عدة سلاسل عمليات للوصول إلى نفس المورد المشترك. ونتيجة لذلك، يتأثر ترتيب تنفيذها سلبًا.

```
monitor monitor_name{

    //declaring shared variables

  variables_declaration;
  condition_variables;

  procedure p1(...) {
   ...
  };

  procedure p2(...){
    ...
  };
  ...
  procedure pn(...){
    ..
  };

 {
   Initailisation Code();
 }

}
```

خصائص المونيتور

- **الاستبعاد المتبادل**: يعني أنه لا يمكن أن يكون هناك سوى عملية أو توزيعة واحدة داخل المونيتور في أي وقت. تمنع هذه الخاصية العمليات المتزامنة من الوصول إلى الموارد المشتركة في وقت واحد وتزيل خطر تلف البيانات أو النتائج غير المتناسقة بسبب ظروف السباق.
- **التغليف:** يقوم المونيتور بتغليف كل من المورد المشترك والإجراءات التي تعمل عليه. ومن خلال تجميع الموارد والإجراءات ذات الصلة معًا، يوفر المونيتور أسلوبًا نظيفًا ومنظمًا لإدارة الوصول المتزامن. يعمل هذا التغليف على تبسيط تصميم وصيانة البرامج المتزامنة، حيث يتم تحديد منطق المزامنة الضروري داخل المونيتور.



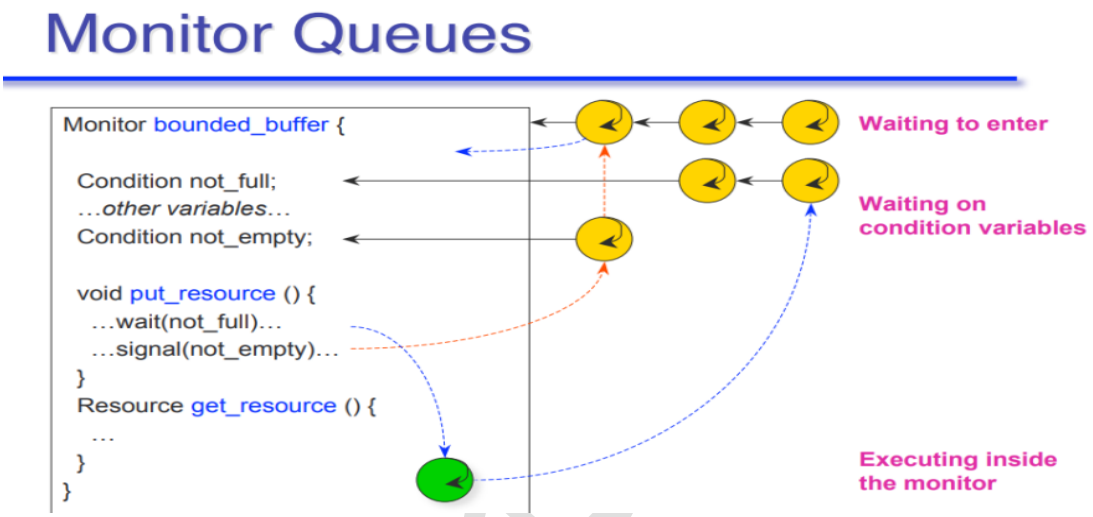
- **أساسيات المزامنة:** يدعم المونيتور غالبًا أساسيات المزامنة، مثل متغيرات الحالة Variable Condition. تعمل متغيرات الحالة على تمكين سلاسل العمليات الموجودة داخل المونيتور من الانتظار حتى تصبح شروط معينة صحيحة أو إرسال إشارة إلى سلاسل رسائل أخرى عند استيفاء شروط معينة. تسمح هذه الأساسيات بالتنسيق الفعال بين سلاسل العمليات وتساعد على تجنب الانتظار المزدحم، الذي يمكن أن يؤدي إلى إهدار دورات وحدة المعالجة المركزية.
- **آلية الحظر:** عندما تحاول عملية أو مؤشر ترابط الدخول إلى مونيتور قيد الاستخدام بالفعل، يتم حظرها ووضعها في قائمة انتظار (قائمة انتظار الإدخال) حتى يصبح المونيتور متاح. تتجنب آلية الحظر هذه الانتظار المشغول وتسمح للعمليات الأخرى بالمتابعة أثناء انتظار دورها للوصول إلى الشاشة.
- **وراثة الأولوية:** في بعض التطبيقات المتقدمة ، يمكن استخدام آلية وراثة الأولوية لمنع عكس الأولوية. عندما ينتظر مؤشر ترابط ذي أولوية أعلى أن يقوم مؤشر ترابط ذي أولوية أقل بتحرير مورد داخل المونيتور،
- **التجريد عالي المستوى:** يوفر المونيتور تجريدًا عالي المستوى لإدارة التزامن مقارنة بآليات المزامنة منخفضة المستوى مثل السيمافور. يقلل هذا التجريد من تعقيد البرمجة المتزامنة ويسهل كتابة التعليمات البرمجية الصحيحة والقابلة للصيانة.

مزايا المونيتورفي نظام التشغيل

- بالمقارنة مع الحلول المعتمدة على السيمافور، يتمتع المونيتور بميزة جعل البرمجة المتزامنة أكثر بساطة وأقل عرضة للخطأ.
- يساعد في مزامنة عمليات نظام التشغيل.
- يحقق المونيتور امكانية الإقصاء المتبادل.
- يعد إعداد السيمافور أكثر صعوبة من إعداد المونيتور.
- يمكن أن يؤدي السيمافور إلى حدوث أخطاء في التوقيت، والتي قد يتمكن المونيتور من إصلاحها.

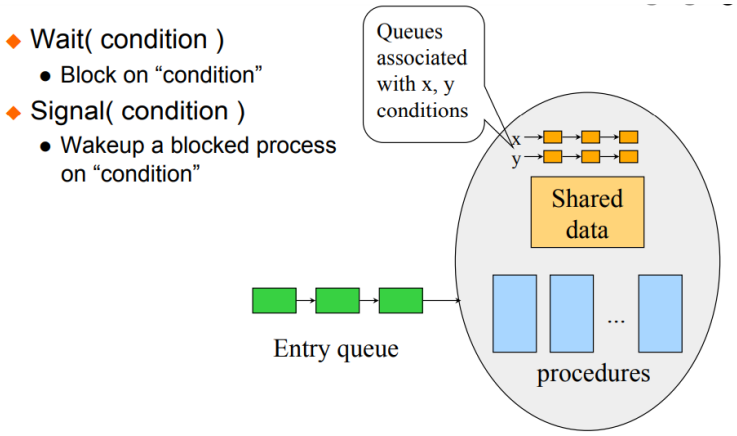
عيوب المونيتور في نظام التشغيل

- يجب تنفيذ المونيتور بلغة البرمجة.
- يعتمد المونيتور على المترجم compiler.
- يجب أن يكون المونيتور على دراية بالميزات التي يوفرها نظام التشغيل لإدارة الخطوات الحاسمة في العمليات المتوازية.



مقارنة بين السيمافور و المونيتور

- السيمافور عبارة عن متغير عدد صحيح يسمح للعديد من العمليات في نظام متوازي بإدارة الوصول إلى مورد مشترك مثل نظام تشغيل متعدد المهام. من ناحية أخرى، المونيتور هي تقنية مزامنة تمكن سلاسل الرسائل من الاستبعاد المتبادل والانتظار () حتى يصبح شرط معين صحيحًا.
- عندما تستخدم عملية ما موارد مشتركة في السيمافور، فإنها تستدعي طريقة WAIT() وتحظر الموارد. عندما تريد تحرير الموارد، فإنها تنفذ SIGNAL() في المقابل، عندما تستخدم العملية موارد مشتركة في المونيتور، يجب عليها الوصول إليها عبر الإجراءات.
- السيمافور عبارة عن متغير عدد صحيح، في حين أن المونيتور هي نوع بيانات مجردة.
- في السيمافور يُظهر المتغير الصحيح عدد الموارد المتاحة في النظام. في المقابل، تعد المونيتور أحد أنواع البيانات المجردة التي تسمح فقط بتنفيذ العملية في القسم المهم في المرة الواحدة.
- السيمافور ليس لها مفهوم لمتغيرات الحالة، في حين أن المونيتور لديها متغيرات الحالة.
- لا يمكن تغيير قيمة السيمافور إلا باستخدام WAIT() و SIGNAL() وفي المقابل، يحتوي المونيتور على المتغيرات المشتركة والأداة التي تمكن العمليات من الوصول إليها.



يمكن تنفيذ مونيتور في لغة C++ باستخدام :

(mutex) كائن المزامنة الخاص بالمكتبة القياسية

condition_variable. متغيرات شرطية في لغة C++، يتم تنفيذ أجهزة المونيتور باستخدام طرق متزامنة، والتي توفر الاستبعاد المتبادل.

std::mutex هو عبارة عن صف class لتحديد قفل الاستبعاد المتبادل.

std::condition_variable عبارة عن صف لتحديد متحولات الحالة للمزامنة. يتم تعريف الإجراءات داخل الكلاس class للوصول إلى الموارد المشتركة ومعالجتها.

std::unique_lock<std::mutex> يقوم بقفل المونيتور. يتم تحرير القفل تلقائيًا عندما تنتهي العملية "unique_lock" في نهاية تنفيذ التابع.

m_cv.wait() تابع يحدد حالة الانتظار و مرتبط بالإشارة الناتجة عن متحولات الحالة

حل لمشكلة القراءة / الكتابة باستخدام المونيتور

هناك مورد مشترك يمكن الوصول إليه من خلال عمليات متعددة، أي القراءة والكتابة. يمكن لأي عدد من القراء القراءة من المورد المشترك في وقت واحد، ولكن يمكن لكاتب واحد فقط الكتابة إلى المورد المشترك في المرة الواحدة. عندما يقوم الكاتب بكتابة البيانات إلى المورد، لا يمكن لأي عملية أخرى الوصول إلى المورد. لا يمكن للكاتب الكتابة إلى المورد إذا كان هناك أي قراء يصلون إلى المورد في ذلك الوقت. وبالمثل، لا يستطيع القارئ القراءة إذا كان هناك كاتب يصل إلى المصدر أو إذا كان هناك أي كتاب ينتظرون.
يمكن تنفيذ مشكلة القارئ والكاتب باستخدام المونيتور باستخدام pthreads. مكتبات سلاسل POSIX (أو pthread) هي واجهة برمجة تطبيقات لمؤشرات ترابط قائمة على المعايير لـ C/C++. توفر المكتبة آليات المزامنة التالية:
كائنات المزامنة (pthread_mutex_t) - قفل الاستبعاد المتبادل:
منع الوصول إلى المتغيرات عن طريق التفريعات الأخرى. يفرض هذا الوصول الحصري بواسطة تفريعة إلى متغير أو مجموعة من المتغيرات.
متغيرات الحالة – (pthread_cond_t):
تسمح آلية متغير الشرط للتفريعات بتعليق التنفيذ والتخلي عن المعالج حتى تتحقق بعض الشروط.

```
// Reader-Writer problem using monitors
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;

class monitor {
private:
	// no. of readers
	int rcnt;

	// no. of writers
	int wcnt;

	// no. of readers waiting
	int waitr;

	// no. of writers waiting
	int waitw;

	// condition variable to check whether reader can read
	pthread_cond_t canread;

	// condition variable to check whether writer can write
	pthread_cond_t canwrite;

	// mutex for synchronization
	pthread_mutex_t condlock;

public:
	monitor()
	{
		rcnt = 0;
		wcnt = 0;
		waitr = 0;
		waitw = 0;

		pthread_cond_init(&canread, NULL);
		pthread_cond_init(&canwrite, NULL);
		pthread_mutex_init(&condlock, NULL);
	}

	// mutex provide synchronization so that no other thread
	// can change the value of data
	void beginread(int i)
	{
```

```
        pthread_mutex_lock(&condlock);

        // if there are active or waiting writers
        if (wcnt == 1 || waitw > 0) {
                // incrementing waiting readers
                waitr++;

                // reader suspended
                pthread_cond_wait(&canread, &condlock);
                waitr--;
        }

        // else reader reads the resource
        rcnt++;
        cout << "reader " << i << " is reading\n";
        pthread_mutex_unlock(&condlock);
        pthread_cond_broadcast(&canread);
}

void endread(int i)
{

        // if there are no readers left then writer enters monitor
        pthread_mutex_lock(&condlock);

        if (--rcnt == 0)
                pthread_cond_signal(&canwrite);

        pthread_mutex_unlock(&condlock);
}

void beginwrite(int i)
{
        pthread_mutex_lock(&condlock);

        // a writer can enter when there are no active
        // or waiting readers or other writer
        if (wcnt == 1 || rcnt > 0) {
                ++waitw;
                pthread_cond_wait(&canwrite, &condlock);
                --waitw;
        }
        wcnt = 1;
        cout << "writer " << i << " is writing\n";
        pthread_mutex_unlock(&condlock);
}
```

```
        void endwrite(int i)
        {
                pthread_mutex_lock(&condlock);
                wcnt = 0;

                // if any readers are waiting, threads are unblocked
                if (waitr > 0)
                        pthread_cond_signal(&canread);
                else
                        pthread_cond_signal(&canwrite);
                pthread_mutex_unlock(&condlock);
        }

}

// global object of monitor class
M;

void* reader(void* id)
{
        int c = 0;
        int i = *(int*)id;

        // each reader attempts to read 5 times
        while (c < 5) {
                usleep(1);
                M.beginread(i);
                M.endread(i);
                c++;
        }
}

void* writer(void* id)
{
        int c = 0;
        int i = *(int*)id;

        // each writer attempts to write 5 times
        while (c < 5) {
                usleep(1);
                M.beginwrite(i);
                M.endwrite(i);
                c++;
        }
}
```

```
int main()
{
	pthread_t r[5], w[5];
	int id[5];
	for (int i = 0; i < 5; i++) {
		id[i] = i;

		// creating threads which execute reader function
		pthread_create(&r[i], NULL, &reader, &id[i]);

		// creating threads which execute writer function
		pthread_create(&w[i], NULL, &writer, &id[i]);
	}

	for (int i = 0; i < 5; i++) {
		pthread_join(r[i], NULL);
	}
	for (int i = 0; i < 5; i++) {
		pthread_join(w[i], NULL);
	}
return 0;
}
```