البرمجة التفرعية باستخدام الـ Multithreading ضمن بيئة الـ VS2010

1 مفردات الجلسة:

- ✓ مزامنة الخيوط (Thread Synchronization)
- ✓ تدريب عملي

2 مزامنة الخيوط (Thread Synchronization):

من أجل العمل بشكل فعال معًا، تحتاج الكائنات الموجودة في البرنامج عادةً إلى مشاركة المعلومات أو تنسيق نشاطها. لقد تم طرح العديد من الطرق للقيام بذلك وعادةً ما تكون هذه التقنيات تحت اسم مزامنة الخيط. فيما يلي بعض الطرق المستخدمة في عملية المزامنة.

1.2 الاستبعاد المتبادل (Mutual Exclusion)

عند كتابة برامج متعددة الخيوط، يكون من الضروري في كثير من الأحيان فرض الوصول الحصري المتبادل إلى كائن بيانات مشترك. يتم ذلك باستخدام كائنات mutex. تتمثل الفكرة في ربط كائن المزامنة (mutex) بكل كائن بيانات مشترك ثم مطالبة كل خيط يرغب في استخدام كائن البيانات المشترك بقفل كائن المزامنة (mutex) أولاً قبل القيام بذلك. وهنا التفاصيل:

1- **Declare an object of type pthread mutex t.**
2- **Initialize the object by calling pthread mutex init() or by using the special static initializer PTHREAD MUTEX INITIALIZER.**
3- **Call pthread mutex lock() to gain exclusive access to the shared data object.**
4- **Call pthread mutex unlock() to release the exclusive access and allow another thread to use the shared data object.**
5- **Get rid of the object by calling pthread mutex destroy().**

من المهم أن نفهم أنه إذا حاول خيط ما قفل كائن المزامنة (mutex) بينما قام خيط آخر بقفله، فسيتم حظر الخيط الثاني حتى يقوم الأول تحرير كائن المزامنة (mutex) باستخدام pthread mutex unlock.

يمكن استخدام التهيئة الديناميكية dynamic أو التهيئة الثابتة static باستخدام الرمز الخاص - PTHREAD MUTEX INITIALIZER.

يجب مراعاة النقاط التالية:

1. يجب ألا يحاول أي خيط قفل أو إلغاء قفل كائن المزامنة (mutex) الذي لم تتم تهيئته.

2. يجب أن يكون الخيط الذي يقوم بقفل كائن المزامنة هو الخيط الذي يحرره.

3. لا ينبغي أن يتم قفل كائن المزامنة (mutex) في أي خيط عند تدمير كائن المزامنة (mutex).

من الناحية العملية، يتم في بعض الأحيان حظر الخيوط على كائن المزامنة (mutex) عندما يرغب البرنامج في الإنهاء. في مثل هذه الحالة قد يكون من المنطقي إلغاء pthread cancel تلك الخيوط قبل تدمير كائنات المزامنة (mutex) المحظورة عليها. ومع ذلك، قد يكون تنسيق ذلك بشكل صحيح أمرًا صعبًا.

لاحظ أنه من الممكن تعيين "سمات كائن المزامنة" mutex attributes الخاصة لكائن المزامنة عندما يتم إنشاؤه. يتم ذلك عن طريق إنشاء كائن سمة mutex، وتعيين سمات للكائن، ثم تمرير مؤشر إلى كائن السمة في pthread mutex init. يتم استدعاء السمات الافتراضية من خلال تمرير مؤشر NULL بدلاً من ذلك. وفي كثير من الحالات يكون هذا كافيًا تمامًا.

من المهم أن نفهم أن قفل كائن المزامنة (mutex) يعد أمرًا اختيارياً. وهذا يعني أنه لا يوجد أي جزء من النظام (operating system, runtime system, or any other system) يتطلب اتباع القواعد. إذا لم يتم قفل كائن المزامنة (mutex) قبل الوصول إلى البيانات المشتركة المحمية، فقد تتداخل الخيوط فيما بينها.

3 تدريب عملي:

1.3 تدريب 1

المطلوب كتابة برنامج يعتمد مبدأ الـ Multithreading ويقوم بمزامنة الخيوط عن طريق mutex

الكود:

```
#include <pthread.h>
#include <unistd.h>
Pthread_mutex_t lock ;
int shared_data ;
// Usually shared data is more complex than just an int.
void * thread_function ( void * arg ) {
int i ;
for ( i = 0 ; i < 1024*1024; ++i ) {
// Access the shared data here.
Pthread_mutex_lock(&lock ) ;
Shared_data++;
Pthread_mutex_unlock(&lock ) ;
}
```

```
return NULL;
}
int main ( ){
pthread_t thread_ID ;
void *threadresult ;
int i ;
// Initialize the mutex before trying to use it.
pthread_mutex_init (&lock , NULL) ;
pthread_create (&thread ID , NULL, thread_function , NULL) ;
// Try to use the shared data.
for ( i = 0 ; i < 1 0 ; ++i ) {
sleep ( 1 ) ;
pthread_mutex_lock(&lock ) ;
printf ( "\rShared integer 's value = %d\n" , shared_data ) ;
pthread_mutex_unlock(&lock ) ;
}
printf ( "\n" ) ;
pthread_join ( thread_ID , &thread_result ) ;
// Clean up the mutex when we are finished with it.
pthread_mutex_destroy(&lock ) ;
return 0 ;
}
```

المطلوب: تجريب البرنامج السابق. هل يتصرف بالطريقة التي توقعتها؟ جرب قيمًا مختلفة لمؤشر الحلقة الأقصى في وظيفة الخيط وأوقات sleep مختلفة في التابع main. حاول إزالة استدعاء Sleep بالكامل.

الوظيفة: شرح ما يحدث؟