البرمجة التفرعية باستخدام الـ Multithreading ضمن بيئة الـ VS2010

1 مفردات الجلسة:

- ✓ إجرائيات إدارة الخيوط (Thread Management)
- ✓ تدريب عملي

2 إجرائيات إدارة الخيوط (Thread Management):

1.2 سمات الخيوط:

افتراضيًا، يتم إنشاء خيط بسمات معينة. يمكن للمبرمج تغيير بعض هذه السمات عبر كائن سمة الخيط (thread attribute)

يتم استخدام التابعين pthread_attr_init وpthread_attr_destroy لتهيئة/تدمير كائن سمة الخيوط.

يمكن استخدام توابع أخرى للاستعلام/تعيين سمات محددة في كائن سمة الخيوط. نذكر منها:

- ✓ Detached or joinable state
- ✓ Scheduling inheritance
- ✓ Scheduling policy
- ✓ Scheduling parameters
- ✓ Scheduling contention scope
- ✓ Stack size
- ✓ Stack address
- ✓ Stack guard (overflow) size

2.2 الانضمام Joining

يقوم الإجراء الفرعي pthread_join() بحظر خيط (Thread) الاستدعاء حتى ينتهي الخيط المحدد. يمكن للمبرمج الحصول على حالة إنهاء الخيط الهدف إذا تم تحديدها في استدعاء الخيط الهدف بـ pthread_exit(). لا يمكن ضم الخيط إلا مرة واحدة. من الخطأ المنطقي محاولة عمليات الانضمام المتعددة على نفس الخيط. يعتبر هذا التابع احدى طرق المزامنة بين الخيوط

قابلة للانضمام أم لا:

عندما يتم إنشاء خيط، تحدد إحدى سماته ما إذا كان قابلاً للانضمام أم منفصلاً. يمكن فقط ضم الخيوط التي تم إنشاؤها كقابلة للانضمام. إذا تم إنشاء خيط منفصل، فلا يمكن ضمه أبدًا. يفضل إنشاء الخيوط على أنها قابلة للانضمام.

لإنشاء مؤشر ترابط بشكل صريح قابل للانضمام أو منفصل، يتم استخدام الخاصية attr ضمن الاجرائية pthread_create().

الخطوات العملية:

- ✓ تعريف متغير من نوع البيانات pthread_attr_t
- ✓ تهيئة متغير السمة باستخدام pthread_attr_init()
- ✓ تعيين حالة السمة المنفصلة باستخدام pthread_attr_setdetachstate()
- ✓ في هذه الحالة سيتم استخدام السمة مع pthread_attr_destroy()

3.2 الفصل Detaching

يمكن استخدام الإجرائية pthread_detach() لفصل خيط بشكل صريح على الرغم من أنه تم إنشاؤه كقابل للانضمام. لا يوجد إجراء عكسي.

4.2 بعض التوصيات في البرمجة:

إذا كان الخيط يتطلب الانضمام، يفضل إنشاءه بشكل صريح كقابل للانضمام، حيث لا يمكن لجميع التطبيقات إنشاء خيوط قابلة للانضمام بشكل افتراضي.

إذا كان معلوم مسبقًا أن الخيط لن يحتاج أبدًا إلى الانضمام إلى خيوط أخرى، يفضل إنشائه في حالة منفصلة، حيث قد يؤدي ذلك إلى تقليل الحمل.

5.2 بعض الإجرائيات الأخرى

pthread_self(): يقوم بإرجاع معرف الخيط الفريد المخصص للنظام للخيط المستدعى

pthread_equal(thread1, thread2): يقارن بين معرفي خيطين. إذا كان المعرفان مختلفان، فسيتم إرجاع 0، وإلا فسيتم إرجاع قيمة غير الصفر.

ملاحظة: بالنسبة لكلا الإجراءين، تكون كائنات معرف الخيط غير شفافة ولا يمكن فحصها بسهولة. نظرًا لأن معرفات الخيوط عبارة عن كائنات غير شفافة، فلا ينبغي استخدام عامل التكافؤ (==) لمقارنة معرفي خطين مع بعضهما البعض، أو لمقارنة معرف خط مع قيمة أخرى.

pthread_once(once_control, init_routine): يقوم بتنفيذ تابع التهيئة init_routine مرة واحدة فقط أثناء المعالجة. الاستدعاء الأول لهذا التابع بواسطة أي خط يقوم بتنفيذ init_routine المحدد، بدون وسطاء (بارامترات). ولن يكون لأي استدعاءات لاحقة أي تأثير. عادةً ما يكون التابع init_routine بمثابة تابع تهيئة.

البارامتر once_control عبارة عن بنية تحكم في المزامنة تتطلب التهيئة قبل استدعاء pthread_once. على سبيل المثال:

```
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

pthread_yield(): يفرض على الخط المستدعي التخلي عن استخدام معالجه، والانتظار في قائمة انتظار التشغيل قبل جدولته مرة أخرى.

3 تدريب عملي:

1.3 تدريب 1

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

الحل:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#define NUM_THREADS      4
void *BusyWork(void *t){
  double i;
  long tid;
  double result=0.0;
  tid = (long)t;
  printf("Thread %ld starting...\n",tid);
  for (i=0; i<1000000; i++)  {
    result = result + sin(i) * tan(i);
  }
  printf("Thread %ld done. Result = %e\n",tid, result);
  pthread_exit((void*) t);
  return NULL;
}
int main (){
  pthread_t thread[NUM_THREADS];
  pthread_attr_t attr;
  int rc;
  long t;
  void *status;
```

```
  /* Initialize and set thread detached attribute */
  pthread_attr_init(&attr);
  /* To be implemented */
  for(t=0; t<NUM_THREADS; t++) {
    printf("Main: creating thread %ld\n", t);
    rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
    if (rc) {
      printf("ERROR; return code from pthread_create() is %d\n", rc);
      exit(-1);
      }
    }
  /*To be Implemented */
pthread_exit(NULL);
}
```

2.3 تدريب2

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

الحل:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#define NUM_THREADS      4
void *BusyWork(void *t){
  long tid;
  double result=0.0;
  tid = (long)t;
  printf("Thread %ld starting...\n",tid);
  for (double i=0; i<1000000; i++) {
    result = result + sin(i) * tan(i);
    }
  printf("Thread %ld done. Result = %e\n",tid, result);
  return NULL;
}
int main(int argc, char *argv[]){
pthread_t thread[NUM_THREADS];
pthread_attr_t attr;
int rc;
```

```
long t;
pthread_attr_init(&attr);
  /* To be implemented */
for(t=0;t<NUM_THREADS;t++) {
  printf("Main: creating thread %ld\n", t);
  rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
  if (rc) {
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
 }
  /* To be implemented */
pthread_exit(NULL);
}
```