جَامعة
المَنارة
MANARA UNIVERSITY

البرمجة التفرعية

Parallel Programming

Dr.-Eng. Samer Sulaiman

2023-2024

مفردات المنهاج

- أساسيات البرمجة التفرعية
  - مقدمة
  - معامل التسريع
  - أنواع الأنظمة المتعددة المعالجات والبرمجيات الداعمة لها
  - موازنة الأعباء وتحمل الخلل
  - تطبيقات البرمجة التفرعية
  - أشكال معالجة المعطيات على التوازي
- الحواسيب التفرعية
  - تصنيف فلاين Flynn's Classification Scheme
  - شبكات الربط الداخلية Interconnection Networks
- مبادئ تصميم الخوارزميات المتوازية
  - مفاهيم أساسية
  - الإجرائيات والمقابلة
  - تقنيات التقسيم
- البرمجيات الداعمة للبرمجة التفرعية
  - المعتمدة على الذاكرة المشتركة
  - المعتمدة على تمرير الرسائل
- تحليل الأداء Performance Analysis

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - عملية تنفيذ مؤشرات ترابط (خيوط) متعددة في وقت واحد.
  - Thread (الخط) هو في الأساس عملية فرعية خفيفة الوزن، وهي أصغر وحدة معالجة.
  - المعالجة المتعددة والخيوط المتعدد، كلاهما يستخدم لتحقيق تعدد المهام.
    - يتم استخدام مؤشرات الترابط (الخيوط) المتعددة بدلاً من المعالجة المتعددة لأن مؤشر الترابط (Thread) يشترك في منطقة ذاكرة مشتركة.
    - لا تخصص منطقة ذاكرة منفصلة، لذا فهي توفر الذاكرة، ويستغرق تبديل السياق بين الخيوط وقتًا أقل من المعالجة
  - الاستفادة من تعدد الخيوط
    - لا يمنع المستخدم لأن الخيوط مستقلة ويمكنك إجراء عمليات متعددة في نفس الوقت
    - يمكنك إجراء العديد من العمليات معًا مما يوفر الوقت
    - الخيوط مستقلة لذلك لا تؤثر على الخيوط الأخرى إذا حدث استثناء في موضوع واحد.
  - سنستخدم مكتبات Pthreads التي تعمل مع لغة C/C++ ضمن بيئة الـ VS2010

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
    - مكتبات POSIX thread هي واجهة تطبيقات برمجية لدعم الخيوط للغة C/C++
    - تسمح بكتابة برامج متعددة الخيوط تعمل بالتوازي .
    - مناسبة وتظهر كفاءتها في الأنظمة متعددة المعالجات (multi-processor) والأنظمة متعددة النواة (multi-core) حيث يمكن لكل خيط أن يعمل في معالج منفصل مما يزيد سرعة التنفيذ خلال المعالجة المتوازية أو الموزعة.
    - يعتبر استخدام الخيوط أقل إزعاجا من استخدام التفريع (forking) والذي يسمح بتنفيذ أكثر من عملية في وقت واحد. لأنه لن يحتاج تهيئة مساحة ذاكرة ظاهرية وبيئة لكل عملية جديدة.
    - يمكن الاستفادة من هذه المكتبات في الأنظمة ذات المعالج الواحد (uniprocessor) حيث يمكن لخيط الاستفادة من نفس المعالج والعمل فيه إذا كان الخيط المنفذ في حالة انتظار دخل أو خرج أو أي شيء آخر ليس للمعالج دخل فيه (تعدد المهام multi-task)

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - يوجد العديد من النماذج الشائعة للبرامج متعددة الخيوط منها:
    - نموذج المدير/العامل (Manager/worker):
      - حيث يقوم الخيط المدير بتوزيع المهام على الخيوط الأخرى والتي تمثل العمال.
    - نموذج الأنبوب الانسيابي (pipeline) :
      - حيث يتم تقسيم المهمة إلى عمليات فرعية بحيث تعتمد كل عملية على مخرجات العملية الأخرى،
      - لكن تنفذ كل عملية في خيط منفصل (مثل تجميع السيارات)
    - الند (peer): يشبة نموذج المدير/العامل، لكنه يختلف في أن المدير بعد توزيعه المهام على العمال يشارك هو في التنفيذ

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - برمجة الخيط بأمان
    - هو مقدرة التطبيقات على تنفيذ عدة خيوط بالتوازي دون تخريب البيانات المشتركة أو توليد حالة سباق (race conditions)
    - مثال: إذا كان هنالك تطبيق يحتوي على عدة خيوط وكل خيط يستدعي نفس إجرائية المكتبة (library routine)،
      - إذا افترضنا أن هذا الإجراء يقوم بتعديل بيانات عامة او موقع في الذاكرة،
        - سيحاول كل خيط تعديل هذه البيانات في نفس الوقت مما يسبب تخريبا في هذه البيانات،
        - لذلك لابد من أن يكون هنالك نوع من الوصول المتزامن لهذه البيانات حتى نحميها من التخريب وحتى يصبح الخيط آمنا.
    - لذلك يتوجب في حال استخدام إجرائيات خارجية ضمان سلامتها والتأكد من أنها آمنة وإلا يجب تجنب استخدامها.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - واجهة التطبيقات البرمجية (Pthreads API)
  - يمكن تقسيم الواجهة البرمجية إلى أربعة مجموعات رئيسية هي:
    - إدارة الخيط:
      - هي الإجرائيات التي تعمل مباشرة مع الخيوط مثل creating, detaching, joining، الخ.
    - إجرائيات الـ mutex:
      - تستخدم في التزامن وتسمى (Mutex) من الكلمتين mutual exclusion، وهي معنية بتعديل الصفات المرتبطة مع الـ mutexes.
    - المتغيرات الشرطية (condition variables):
      - هي الإجرائيات التي تهتم بالاتصالات بين الخيوط المتشاركة في mutex وهي معتمدة على شروط يضعها المبرمج.
    - التزامن (synchronization):
      - إجرائيات لإدارة حجز القراءة والكتابة.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - واجهة التطبيقات البرمجية (Pthreads API)
    - كل التعريفات في مكتبة الخيوط تبدأ بالكلمة pthread_ كما موضح في الأمثلة التالية:

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_mutexattr_	Mutex attributes objects.
pthread_cond_	Condition variables
pthread_condattr_	Condition attributes objects
pthread_key_	Thread-specific data keys
pthread_rwlock_	Read/write locks
pthread_barrier_	Synchronization barriers

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
    - إنشاء خيط
- لإنشاء خيط نحتاج استخدام الدالة pthread_create() والتي تحتوي على المعطيات التالية:
  - المعطى الأول نحصل من خلاله على تعريف الخيط (thread identifier)
  - المعطى الثاني مؤشر إلى مكان الكائن الذي يحدد صفات الخيط،
    - إذا تم استخدام الـ null هنا فهذا يعني استخدام الصفات الافتراضية للخيط.
  - المعطى الثالث هو مؤشر إلى مكان الدالة التي ينفذها الخيط.
  - المعطى الأخير هو القيم (argument) التي نريد تمريرها إلى الدالة المنفذة داخل الخيط،
    - إذا لم يكن هناك قيم يراد تمريرها إلى الدالة يمكن كتابة null في هذه الخانة.
- مثال لو كتبت شفرة الدالة بهذه الطريقة:
  - pthread_create(&th_ID, NULL, th_fun, &value);
- فهذا يعني إنشاء خيط يخزن تعريف هذا الخيط في المتغير th_ID، ويستخدم هذا الخيط الصفات الافتراضية، وينفذ هذا الخيط الدالة th_fun التي تكون مدخلاتها value
- إذا كان المطلوب إنشاء ثلاث خيوط فيجب استدعاء الدالة pthread_create() ثلاث مرات.
- يمكن استدعاء الدالة pthread_join() مع كل خيط قمنا بإنشائه لضمان انتهاء الخيوط قبل انتهاء الدالة الرئيسية main

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
    - إنشاء خيط

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
    - .إنهاء خيط
      - لإنهاء خيط نستخدم الدالة ()pthread_cancel
        - لكن يجب التأكد من أن الخيط الذي تريد إنهاؤه لا يستخدم موارد قبل إنهاؤه.
      - مثال: إذا كان الخيط يحجز مساحة بالذاكرة وقمنا باستدعاء دالة الإنهاء ()pthread_cancel
        - سنفقد مكان هذه الذاكرة وستظل محجوزة بلا فائدة (memory leak)
      - الاجرائيات المستخدمة في إنشاء وإنهاء الخيوط.
        - pthread_create (thread, attr, start_routine, arg)
        - pthread_exit (status)
        - pthread_attr_init (attr)
        - pthread_attr_destroy (attr)

- تنبيهات عن إنشاء الخيط:
  - الدالة الرئيسية main لديها خيط واحد افتراضي، بقية الخيوط على المبرمج أن يقوم بإنشائها بنفسه.
  - الأمر pthread_create ينشيء خيط ويمكن استدعائه أكثر من مرة من أي مكان داخل الكود البرمجي لإنشاء أكثر من خيط.
  - عند انشاء الخيط يمكنه بدوره إنشاء خيوط أخرى، فليس هنالك هرمية بين الخيوط.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - واصفات الخيط (Thread Attributes)
    - افتراضيا ينشأ الخيط بصفات معينة، ويمكن للمبرمج تغير بعض هذه الصفات عبر كائن الصفات (thread attribute object)
    - تستخدم pthread_attr_init و pthread_attr_destroy لتهيئة/تدمير كائن الصفات .
    - هنالك إجرائيات اخرى تستخدم لمعرفة أو تغيير صفات معينة في كائن الصفات.
    - انهاء الخيط (Terminating Threads)
      - هنالك عدة طرق لإنهاء الخيط منها:
        - رجوع الخيط من الإجرائية التي بدأ فيه (الإجرائية الرئيسية التي قامت بتهيئة الخيط)
        - استدعاء الخيط للإجرائية pthread_exit .
        - إلغاء الخيط بخيط آخر وذلك باستدعاء الإجرائية pthread_cancel .
        - إذا انتهت العملية بكاملها باستدعاء الإجرائية مثل exec أو exit .

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - صفات الخيط (Thread Attributes)
  - انهاء الخيط (Terminating Threads)
- يمكن استخدام الإجرائية pthread_exit للخروج من الخيط ويتم استدعاءها في نهاية الخيط عند ما نريد عمل شيء آخر.
- إذا انتهى البرنامج الرئيسي ()main قبل الخيوط التي أنشأها وخرج بالأمر ()pthread_exit ، فإن الخيوط الأخرى ستظل تعمل.
  - أما إذا انتهت ()main فستنتهي معها كل الخيوط التي أنشاءها.
- يستطيع المبرمج (اختياريا) تحديد حالة الانتهاء (termination status) والتي تكون مخزنة كمؤشر من النوع void في أي خيط قد يشارك في استدعاء الخيط.
- الإجرائية ()pthread_exit لا تغلق الملفات المفتوحة داخل أي خيط وستظل مفتوحة حتى بعد انتهاء الخيط.
- ملاحظة: في الإجرائيات التي من المتوقع انتهاء تنفيذها بصورة طبيعة يمكن الاستغناء عن ()pthread_exit، ما لم يتم تمرير pass a return code back
  - لكن هنالك مشكلة وهي: عندما تكتمل ()main قبل توزيع الخيوط (threads it spawned)
    - فإذا لم يتم استدعاء الـ ()pthread_exit ضمنيا، عندما تكتمل ()main فإن العملية (وكل خيوطها) ستنتهي.
    - باستدعاء الـ ()pthread_exit في ()main، فإن العملية وكل خيوطها ستبقى حية حتى ولو أكتمل تنفيذ كل الكود الموجود في ()main

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - مثال: برنامج بسيط ينشئ خيط واحد

```
#include <pthread.h>
#include <stdio.h>
void * entry_point(void *arg) {
printf("Hello world!\n");
return NULL; }
int main(int argc, char **argv) {
pthread_t thr;
if(pthread_create(&thr, NULL, &entry_point, NULL)) {
printf("Could not create thread\n");
return -1; }
if(pthread_join(thr, NULL)) {
printf("Could not join thread\n");
return -1; }
return 0;
}
```

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - مكتبات POSIX thread (Pthread)
  - مثال: برنامج ينشئ 5 خيوط

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
void *PrintHello(void *threadid) {
long tid;
tid = (long)threadid;
printf("Hello World! It's me, thread #%ld!\n", tid);
pthread_exit(NULL);
return NULL; }
int main (int argc, char *argv[]) {
pthread_t threads[NUM_THREADS];
int rc;
long t;
for(t=0; t<NUM_THREADS; t++){
printf("In main: creating thread %ld\n", t);
rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
if (rc){
printf("ERROR; return code from pthread_create() is %d\n", rc);
//exit(-1);
} }
pthread_exit(NULL); }
```