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:
  - تمرير قيم إلى الخيط (Passing Arguments to Threads)
    - من تابع الانشاء نلاحظ أن القيمة التي يتم تمريرها للدالة هي مخزنة في متحول، وهي قيمة واحدة.
      - ذلك لان الدالة ()pthread_create تسمح بتمرير قمية واحدة (one argument) للدالة التي تنفذ الخيط.
    - لتمرير أكثر من قيمة يمكن استخدام بنية بيانات (data structure) تحتوي كل القيم المراد تمريرها (مثل السجلات أو الأصناف أو المؤشرات)، ثم وضع مؤشر هذه البنية في الدالة ()pthread_create

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - تمرير قيم إلى الخيط (Passing Arguments to Threads)
    - مثال:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 8
char *messages[NUM_THREADS];
struct thread_data {
int thread_id;
int sum;
char *message; };
struct thread_data thread_data_array[NUM_THREADS];
void *PrintHello(void *threadarg) {
int taskid, sum;
char *hello_msg;
struct thread_data *my_data;
my_data = (struct thread_data *) threadarg;
taskid = my_data->thread_id;
sum = my_data->sum;
hello_msg = my_data->message;
printf("Thread %d: %s Sum=%d\n", taskid, hello_msg, sum);
pthread_exit(NULL);
return NULL;
}
```

```
int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int *taskids[NUM_THREADS];
int rc, t, sum;
sum=0;
messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le
monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvytye,
mir!";
messages[6] = "Japan: Sekai e
konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";
for(t=0;t<NUM_THREADS;t++) {
sum = sum + t;
thread_data_array[t].thread_id = t;
thread_data_array[t].sum = sum;
thread_data_array[t].message =
```

```
messages[t];
printf("Creating thread %d\n", t);
rc = pthread_create(&threads[t], NULL,
PrintHello, (void *)
&thread_data_array[t]);
if (rc) {
printf("ERROR; return code from
pthread_create() is %d\n", rc);
exit(-1);
}
}
system("pause");
pthread_exit(NULL);
}
```

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - الإنضمام (Joining):
    - تعتبر احدى الطرق لتنفيذ التزامن (synchronization) بين الخيوط
    - التابع pthread_join() يحجز الخيط المستدعي (calling thread)حتى ينتهي الخيط المحدد (threaded)
    - يمكن الحصول على حالة انتهاء الخيط الهدف إذا كان محددا عند استدعاء التابع . pthread_exit()
    - تتم عملية إنضمام الخيط (joining thread) مرة واحدة فقط
      - أي استدعاء الروتين pthread_join()مرة واحدة لنفس الخيط
      - يعتبر خطأ منطقي محاولة عمل أكثر من إنضمام (multiple joins) لنفس الخيط.
    - هنالك طرق أخرى للتزامن مثل mutexes و المتغيرات الفرعية (condition variables)
    - هل الخيط قابل للإنضمام أم لا (Joinable or Not؟)
      - عند إنشاء الخيط هنالك أحد صفاته توضح هل هذا الخيط قابل للإنضمام (joinable) أم لا (detached)
        - فقط الخيوط القابلة للإنضمام هي التي يمكن ضمها
        - إذا أنشيء الخيط كخيط منفصل (detached) فلا يمكن ضمه أبدا.
      - في معايير POSIX الاخيرة تم تحديد أن الخيط يجب أن يكون من النوع القابل للضم (joinable) عند إنشائه.
      - لتحديد حالة الخيط عند إنشائة هل سيكون joinable أو detached، يمكن استخدام القيمة المدخلة attr ضمن تابع الانشاء pthread_create() .:

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - الإنضمام (Joining):
    - خطوات تغيير خاصية الخط:

- Declare a pthread attribute variable of the pthread_attr_t data type
- Initialize the attribute variable with pthread_attr_init()
- Set the attribute detached status with pthread_attr_setdetachstate()
- When done, free library resources used by the attribute with pthread_attr_destroy()

- الإنفصال (Detaching)
  - يمكن استخدام التابع pthread_detach() لجعل الخيط منفصل حتى ولو تم انشائه قابل للانضمام (joinable)
    - لا يوجد تابع عكسي.
  - ملاحظة:
    - يجب انشاء الخط من البداية من النوع القابل للانضمام (joinable) عندما يكون الهدف البرمجي يتطلب ذلك ، الأمر الذي يسمح بالتنقلية (portability)
      - في الحالة العامة ليس كل المكتبات تنشيء الخيط من البداية قابل للانضمام (joinable by default)
    - في حال كان الهدف البرمجي يتطلب أن الخيط لن يحتاج إنضمام مع خيط آخر فيمكن إنشائه في حالة المنفصل حيث يوفر ذلك تحرير بعض موارد النظام.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - يوجد العديد من التوابع والاجرائيات الأخرى المفيدة والتي يمكن استخدامها مع الخطوط:
    - التابع pthread_self : يعطي (returns) رقم خيط فريد للخيط المستدعي (calling thread)
    - التابع pthread_equal: يقارن بين رقمي خيطين (thread Ids) ويعطي 0 إذا كانا مختلفان وقيمة غير صفرية في غير ذلك.
      - يعتبر رقم تعريف الخيط كائن وبالتالي لا يمكن استخدام العلامة == لمقارنة قيم الارقام التعريفية للخيوط (thread Ids)
    - متغيرات الميوتكس) (Mutex Variables
      - كلمة Mutex هي اختصار لـ "mutual exclusion"
      - تعتبر متغيرات Mutex من أهم التطبيقات التي تستخدم لتحقيق التزامن بين الخيوط (thread synchronization) وحماية البيانات المشتركة عند حدوث أكثر من أمر كتابة عليها (multiple writes)
      - يعمل المتغير mutex كإقفال (lock) لمنع الوصول إلى البيانات المشركة.
        - المعنى من وراء فكرة المتغير mutex المستخدمة في Pthreads هو أن خيط واحد فقط هو من يقفل (يمتلك) متغير mutex في أي لحظة.
      - أي إذا كان هنالك عدد من الخيوط تحاول قفل mutex واحدة فقط هي التي ستنجح. ولن يستطيع أي خيط آخر قفل (إمتلاك) هذا المتغير (mutex) ما لم يتركه الخيط الذي حصل عليه مسبقاً.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس (Mutex Variables)
    - يمكن استخدام Mutexes في الوقاية من (منع) حدوث حالات سباق ("race" conditions)
    - مثال:

Thread 1	Thread 2	Balance
Read balance: $1000		$1000
	Read balance: $1000	$1000
	Deposit $200	$1000
Deposit $200		$1000
Update balance $1000+$200		$1200
	Update balance $1000+$200	$1200

- يجب عمل mutex لغلق الرصيد ("lock the "Balance) بينما يقوم الخيط باستخدام هذا البيانات المشتركة.
- المتغيرات التي تم تحديثها تنتمي إلى مقطع حرج (critical section)
- .

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس (Mutex Variables)
    - يمكن استخدام Mutexes في الوقاية من (منع) حدوث حالات سباق ("race" conditions)
      - مثال:
        - مقارنة بين حالة استخدام Mutexes وعدم استخدامه

Without Mutex	With Mutex
`int counter=0;`	`/* Note scope of variable and mutex are the same */` `pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;`

`/* Function C */`
`void functionC()`
`{`
`counter++`
`}`
	`int counter=0;`
	
	`/* Function C */`
	`void functionC()`
	`{`
	`pthread_mutex_lock( &mutex1 );`
	`counter++`
	`pthread_mutex_unlock( &mutex1 );`
	`}`

Possible execution sequence

Thread 1	Thread 2	Thread 1	Thread 2
counter = 0	counter = 0	counter = 0	counter = 0
counter = 1	counter = 1	counter = 1	Thread 2 locked out. Thread 1 has exclusive use of variable `counter`
			counter = 2

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس) (Mutex Variables
    - خطوات استخدام mutexكما يلي:
      - إنشاء وتهيئة المتغير . mutex
      - عدة خيوط تحاول غلق الـ mutex
      - خيط واحد فقط ينجح في امتلاك الـ mutex
      - ينفذ الخيط المالك للـ mutex التعامل مع البيانات.
      - يترك الخيط المالك الـ . mutex
      - يحصل خيط آخر على الـ mutex يؤدى عملا ما.
      - في النهاية يتم تدمير الـ . mutex
  - عندما تتنافس عدة خيوط على mutex، تظل الخيوط التي لم تحصل على الـ mutex محجوزة في ذلك الوضع وتستدعى طلب الحجز بالامر "trylock" بدلا من استدعائه بالامر . "lock"
  - عند حماية بيانات مشتركة فإن مهمة المبرمج التأكد من أن كل خيط يحتاج استخدام الـ mutex قد فعل.
    - فمثلا لوكان لدينا ثلاث خيوط تريد تعديل نفس البيانات، ولكن خيط واحد فقط هو من استخدم الـ mutex، فستظل البيانات مخربة ولا فائدة من استخدام الـ mutex

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس (Mutex Variables)
    - إنشاء وتدمير ال Mutexes
    - الاجرائيات المستخدمة:

- pthread_mutex_init (mutex,attr)
- pthread_mutex_destroy (mutex)
- pthread_mutexattr_init (attr)
- pthread_mutexattr_destroy (attr)

- طريقة الاستخدام:
  - يجب الإعلان عن متغيرات الـMutex بالنوع pthread_mutex_t، ثم تهيئتها قبل استخدامها.
  - يوجد طريقتين لتهيئة المتغير mutexهي:
    - ساكن (Statically)، عند الاعلان عنه. مثلا:
      - pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
    - ديناميكي (Dynamically)، باستخدام التابع pthread_mutex_init()
      - يمكن إعداد صفات كائن الـ mutexوهي . attr
  - عند التهيئة يكون المتغير mutex غير محجوز (unlocked)

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس (Mutex Variables)
    - غلق وفتح ال Mutexes
      - الاجرائيات المستخدمة:

- pthread_mutex_lock (mutex)
- pthread_mutex_trylock (mutex)
- pthread_mutex_unlock (mutex)

- طريقة الاستخدام:
  - التابع ()pthread_mutex_lock يستخدم بواسطة الخيط للحصول على المتغير mutex(الغالق).
    - إذا كان الـ mutex مغلق بخيط آخر سيتم حجز هذا النداء حتى يتم فتح (unlocked) المتغير mutex
  - التابع ()pthread_mutex_trylock يحاول غلق المتغير mutex، لكنه يعطي عبارة الخطأ busy مباشرة إذا وجد المتغير mutex مغلق.
    - This routine may be useful in preventing.
      - يكون هذا الروتين مناسب لتجنب شرط من شروط الاختناق (deadlock)
  - التابع ()pthread_mutex_unlock يفتح الـ mutex إذا استدعي بواسطة الخيط المالك للـ mutex، وهو مطلوب استدعاءه بعد أن ينتهي الخيط من استخدام البيانات المحمية.
  - قد يحدث خطأ في الحالات التالية:
    - إذا كان الـ mutex أصلا مفتوح (unlocked)
    - إذا كان الـ mutex مملوك لخيط آخر.

البرمجة المتوازية
البرمجة عن طريق Multithreading

- البرمجة متعددة الخيوط Multithreading:
  - متغيرات الميوتكس (Mutex Variables)
  - مثال:

```
#include <pthread.h>
#include <iostream>
#include <math.h>
#define ITERATIONS 10000
// A shared mutex
pthread_mutex_t mutex;
double target;
void* opponent(void *arg) {
for(int i = 0; i < ITERATIONS; ++i) {
// Lock the mutex
pthread_mutex_lock(&mutex);
target -= target * 2 + tan(target);
// Unlock the mutex
pthread_mutex_unlock(&mutex);
}
return NULL;
}
```

```
int main(int argc, char **argv) {
pthread_t other;
target = 5.0;
// Initialize the mutex
if(pthread_mutex_init(&mutex, NULL)) {
printf("Unable to initialize a
mutex\n");
return -1;
}
if(pthread_create(&other, NULL,
&opponent, NULL)) {
printf("Unable to spawn thread\n");
return -1;
}
for(int i = 0; i < ITERATIONS; ++i) {
pthread_mutex_lock(&mutex);
```

```
target += target * 2 + tan(target);
pthread_mutex_unlock(&mutex);
 }
 if(pthread_join(other, NULL))
 {
 printf("Could not join thread\n");
 return -1;
 }
 // Clean up the mutex
 pthread_mutex_destroy(&mutex);
 printf("Result: %f\n", target);
 system("pause");
 return 0;
 }
```