Chapter 4: Threads

ملاحظات:

١. أي برنامج يتكون من عدة threads وهي تعتبر جزء من عمل البرنامج.

٢. تعتبر الـ threads أصغر من حجم الـ Process وتنجز أسرع منها.

الفوائد (Benefits of threads)

١. الاستجابة (Responsiveness):

- السماح باستمرار التنفيذ حتى إذا كان جزء من العملية محجوزًا blocked، وهذا مهم بشكل خاص لواجهات المستخدم.

٢. مشاركة الموارد (Resource Sharing):

- تشارك (threads) موارد العملية، مما يجعلها أسهل من استخدام الذاكرة المشتركة أو تمرير الرسائل.

٣. الاقتصادية (Economy):

- أرخص من إنشاء العمليات، حيث تكون تكلفة تبديل threads أقل من تكلفة تبديل السياق(التبديل بين العمليات).

٤. القابلية للتوسع (Scalability):

- يمكن للعملية الاستفادة من معمارية المعالجات المتعددة.
- يمكن لكل معالج تنفيذ thread واحد، بينما يمكن للعديد من المعالجات تنفيذ عملية واحدة فقط.

برمجة المعالجات المتعددة (Multicore Programming)

- **أنظمة المعالجات المتعددة: تضع أنظمة المعالجات المتعددة ضغوطًا على المبرمجين، والتحديات تشمل:**
  - **تقسيم الأنشطة (Dividing activities)**
  - **التوازن (Balance)**
  - **تقسيم البيانات (Data splitting)**
  - **اعتماد البيانات (Data dependency)**
  - **الاختبار وإزالة الأخطاء (Testing and debugging)**

- **التوازي (Parallelism):** يعني أن النظام يمكنه تنفيذ أكثر من مهمة في وقت واحد.
- **التزامن (Concurrency):** يعني أن النظام يدعم أكثر من مهمة واحدة تتقدم في الوقت نفسه.

أنواع التوازي Types of parallelism:

١. توازي البيانات (Data parallelism):

- يوزع مجموعات فرعية من نفس البيانات عبر نوى متعددة، تقوم بمعالجة البيانات (أكثر من معالج يقوم بمعالجة نفس العملية).

٢. توازي المهام (Task parallelism):

- توزيع العمليات المشتقة (threads) عبر النوى، حيث تقوم كل نوى بتنفيذ عملية فريدة.

عمليات المستخدم وعمليات النواة (User Threads and Kernel Threads)

عمليات المستخدم (User Threads)

يتم إدارتها والتحكم بها على مستوى المستخدم.

- ثلاث مكتبات رئيسية خاصة بالمستخدم:

١. POSIX Pthreads
٢. (Windows threads)
٣. (Java threads)

عمليات النواة (Kernel Threads)

تُدعم من قِبل نواة نظام التشغيل. وهي التي تقوم بإنجاز مهام المستخدم.

- أمثلة: تشمل جميع أنظمة التشغيل العامة تقريبًا، مثل:

١. ويندوز (Windows)
٢. Solaris
٣. Linux
٤. Tru64 UNIX
٥. Mac OS X

نماذج تعدد العمليات المشتقة (Multithreading Models)

١. العديد إلى الواحد (Many-to-One):

٢. الواحد إلى الواحد (One-to-One):

٣. العديد إلى العديد (Many-to-Many):

- **العديد إلى الواحد (Many-to-One)**

- الوصف: يتم ربط العديد من عمليات المستخدم بعملية نواة واحدة.

- عيوبها:

١. إذا تم حجب عملية واحد، فإن جميع العمليات الأخرى ستتوقف أيضًا.
٢. عدم التوازي: قد لا تتمكن العمليات المتعددة من العمل في وقت واحد على نظام متعدد النوى.
٣. نموذج نادر الاستخدام: قلة من الأنظمة تستخدم هذا النموذج حاليًا.

- أمثلة:

- Solaris Green Threads

- **الواحد إلى الواحد (One-to-One)**

- الوصف: يتم ربط كل عملية مستخدم بعملية نواة واحدة.

- الإيجابيات:

- زيادة التوازي: يوفر هذا النموذج مزيدًا من التوازي concurrency مقارنةً بنموذج العديد إلى الواحد.

- تحسين الأداء: يسمح بتنفيذ عدة خيوط في وقت واحد، مما يعزز التزامن.

- العيوب:

قد يتم تقييد عدد العمليات لكل عملية بسبب overhead المرتبط بعدد المعالجات (CPU). أي انه كلما زاد عدد الـ threads الموجودة بالـ kernel فاننا بحاجة الى عدد كبير من المعالجات.

- أمثلة:

- ويندوز (Windows)

- لينكس (Linux)

- Solaris 9 وما بعده

- **نموذج العديد إلى العديد (Many-to-Many Model)**

- الوصف: يسمح بربط العديد من عمليات المستخدم بالعديد من عمليات النواة.

- المميزات:

- يمكن لنظام التشغيل إنشاء عدد كافٍ من عمليات النواة حسب الحاجة، مما يعزز الأداء.

- أمثلة:

- Solaris قبل الإصدار ٩

- ويندوز باستخدام حزمة ThreadFiber

- IRIX

- HP-UX

- Tru64 UNIX

- Solaris 8 وما قبله

تعدد العمليات (Multithreading)

- يقوم بتكوين نواة منطقية من نواة مادية واحدة.

- **الفائدة**:

١. يعزز كفاءة استخدام الموارد، حيث يمكن تنفيذ مهام متعددة في وقت واحد على نفس النواة المادية.
٢. يدعم التوازي والتزامن، مما يحسن الأداء العام للنظام.

دلالات ()fork و ()exec Semantics of fork() and exec()

١. ()fork:

- استنساخ العملية المشتقة:

- في معظم أنظمة UNIX، يقوم fork() بإنشاء نسخة جديدة من العملية ، بما في ذلك جميع العمليات الموجودة في العملية الأصلية.

- ومع ذلك، بعض أنظمة UNIX قد توفر نسختين من ()fork، واحدة لاستنساخ جميع العمليات المشتقة والأخرى لاستنساخ العملية المشتقة الذي تستدعي ()fork فقط.

٢. ()exec:

- استبدال العملية:

- يعمل ()exec بشكل طبيعي لاستبدال العملية الجارية، بما في ذلك جميع العمليات المشتقة.

إشكاليات العمليات ()fork && ()exec

- السؤال: ماذا يحدث عندما تستدعي عملية ما fork()؟ هل يتم استنساخ جميع العمليات المرتبطة بالعملية التي تم عمل لها ()fork أم عملية واحد فقط؟

مما يعني أي نسخة من ()fork يجب استخدامها؟

الجواب: إذا تم استدعاء ()exce مباشرة بعد ()fork فإن استنساخ جميع العمليات المشتقة يكون غير ضروري. أي أن في هذه الحالة يتم فقط استنساخ العملية المشتقة الذي تستدعي ()fork فقط.

إذا لم تستدعِ العملية المنفصلة ()exec بعد ()fork:

- يجب على العملية المنفصلة استنساخ لجميع العمليات المرتبطة بالعملية التي تم عمل لها ()fork.

إلغاء العمليات (Thread Cancellation)

- إلغاء عملية يعني إنهائها قبل أن يكتمل عملها، مثل إيقاف تحميل أو فتح موقع ويب. أو عندما تحصل العملية على النتائج، يتم إلغاء جميع العمليات الأخرى.

- العملية المستهدفه:

- هي العملية التي سيتم إلغاؤه يُعرف تعرف بالعملية المستهدفة (target thread).

- النهجان العامان لإلغاء العمليات Two general approaches:

١. الإلغاء غير المتزامن (Asynchronous Cancellation):

- تقوم عملية اخرى بإنهاء العملية المستهدفة على الفور.

٢. الإلغاء المؤجل (Deferred Cancellation):

- يسمح للعملية المستهدفة بالتحقق دوريًا مما إذا كان يجب عليها إلغاء نفسها، ولكنها لا تُلغى بواسطة عمليات أخرى.

إشكاليات العمليات المشتقة (Threading Issues)

. إلغاء العملية المستهدفة (Thread Cancellation):

الإشكاليات التي تحدث عند الغاء العمليات:

- قد يتم استرداد الموارد أو لا يتم استردادها بالكامل بعد إلغاء العملية.
- بعض العمليات قد تظل مطلوبة من قبل عمليات أخرى.

. الحل:

يجب على العملية التحقق من نفسها قبل أن يتم إلغاؤها من قبل عملية أخرى. حيث يجب على العملية هي التي تقوم بإلغاء نفسها **الإلغاء المؤجل (Deferred Cancellation)**

هذا يضمن أن العمليات المهمة تبقى نشطة إذا كانت ضرورية للعملية الرئيسية.