المهمة أو العملية[32] (Process):

يتكون البرنامج (application) من واحد أو أكثر من المهام , ووجدت العديد من التعاريف للمهمة منها:

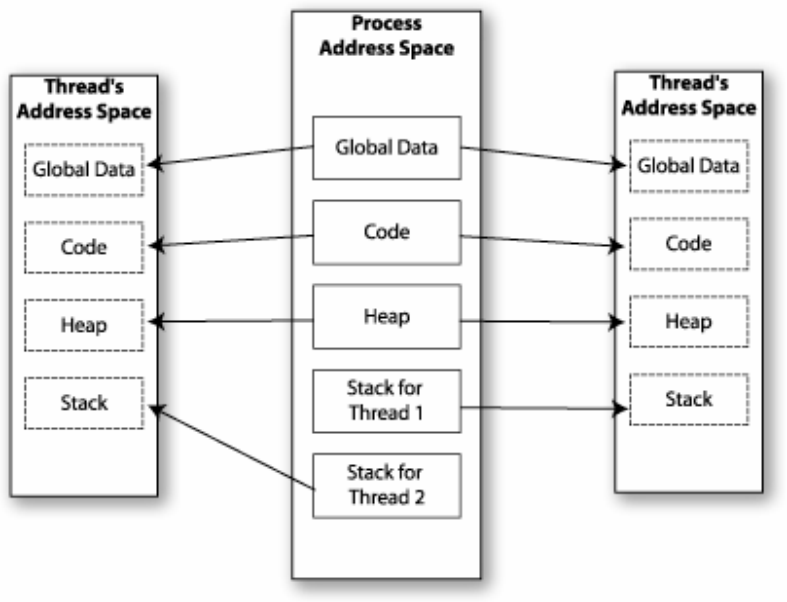
1-المهمة هو برنامج في قيد التنفيذ (an executing program) ويمتاز بأن له مصدر يأخذ منها البيانات مثل الذاكرة , أو أي مصادر يحتاجها في تنفيذ البرنامج.
2- (Process)هي عبارة عن بنية معطيات يقوم نظام التشغيل بإنشائها في الذاكرة الرئيسية ويوضع (mapping)التطبيق المراد تشغيله ضمن هذه البنية.
3- المهمة هو برنامج يعمل له تنفيذ على جهاز الكمبيوتر ، مثل انترنت إكسبلورر أو مايكروسوفت

الخيط أو سلسلة التعليمات[33](Thread):

وهي مجموعة من المهام التي ينقسم إليها البرنامج وذلك ليقوم بأكثر من مهمة بشكل متزامن حقيقي (وذلك عند وجود أكثر من معالج)أو تزامن كاذب (عند وجود معالج واحد)حيث يتم التبديل بين المهام بسرعة كبيره تعطينا انطباع بالتزامن.
وتعتبر (thread)الوحدة الأساسية لوحدة المعالجة المركزية.

---

[32]منى البريه
[33]لمياء السدحان



نلاحظ هنا بالرسم التوضيحي مهمة بأكثر من خيط ونلاحظ اشتراك الخيوط في الفضاء (address space) .

ولو ترجمنا **thread** *نجد معناه خيط ! !*

ماذا يعني خيط ؟.....

هو المسار أو الطريق أو الخيط الذي يؤدي للمهمة نفسها , وهو يعتبر جزء من المهمة , وبما أنه مجرد طريق أو مسار أو خيط فبالطبع لا يحتاج مصادر مثل ذاكره حتى يأخذ البيانات منها , لأنه سوف يأخذ المصادر من مصادر المهمة نفسها .

ولكل خيط :

1- رقم (ID) للتعريف به
2- عداد(program counter)

3- سجل(register)
4- كومة(stack)

أما العلاقة بين الخيط والمهمة:

1- الـــخيوط تعتبر أجزاء من المهام بحيث أن كل جزء يقوم بمهمة معينه ، ولكن جميع الـــخيوط تستعين بالـــمهمة كمصدر لها , ويتم تحديد المهام من قبل المبرمج أو البرنامج .
2 – العملية ستنفذ الخيوط (مجموعة من التعليمات) ، والذي يمكن أن يحتوي على عدة خيوط في بعض الأحيان.
3- جميع العمليات تتكون من واحد أو أكثر من الخيوط .
4- الخيوط والمهام كلاهما يشتركان في المعالج (SHARE CPU) وإنشاء طفل (creat child).

ما هو الفرق بين المهمة و الخيط[34]؟

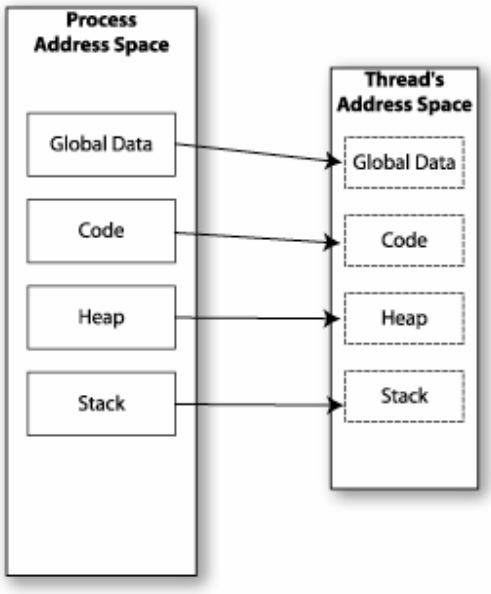
1- الخيط هي أجزاء من المهمة
2- لكل مهمة عنوان(address space) مختلف و بيئة وقت التشغيل(runtime enivorment) ورقم تعريف (process ID) أما الخيوط طالما
أنها مشتقة من المهمة لها عنوان واحد , ويشترك الخيط مع الخيوط الأخرى الموارد التابعة له.
3- لا يوجد خيط من غير مهمة لكن العكس ممكن
4- في حالة سياق التحول (context switch) مع المهمة يظهر over head , أما بحالة الخيط لا يظهر إلا إذا استدعينا نظام التشغيل وغالبا
لا نستدعيه .
5- الخيوط هي أجزاء متزامنة التنفيذ داخل مهمة ما. (thread is a concurrent unit of process execution inside a)
6- المهمة هي كلمة أعم و لفترة أطول , الخيط يقتصر على مفهوم "خط التنفيذ".

---

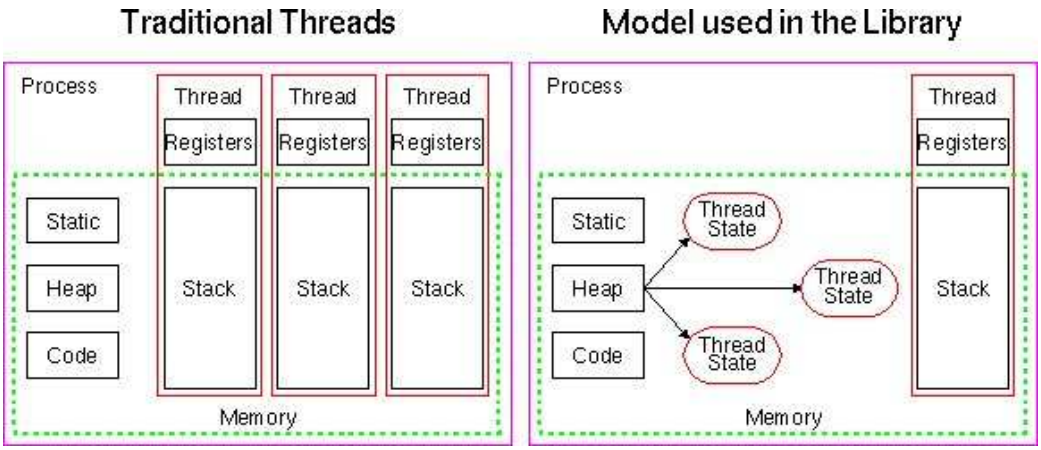
[34] منى البريه و بيان اليوسف

7- المهمة مرتبطة غالباً مع مستوى نظام التشغيل (مثل في متعدد المعالجة) ، بينما الخيط مرتبط بمستوى اللغة المنطقي المجرد.

8- الخيوط ليست مستقلة مثل العملية.



نلاحظ هنا بالرسم التوضيحي مهمة واحدة وخيط واحد وهذا يعني اشتراك تام في كل شيء.



الصورة توضح أن الخيوط تشترك مع المهام في الفضاء ولكن هناك بعض العناصر تكون خاصة بخيط معين مثل:
سجل(register) و كومة (stack)

(تشترك سلاسل المهام التي في نفس العملية الـــمدونة(code) والبيانات(data) وموارد عملية التشغيل(files) .

وتستفرد عن بقية سلاسل المهام بـــالـــ (register) و (stack) .)

(يوجد نوعين مختلفين للعمليات (processes)وهما[35]:

(1) عمليات مفرده المهام (single Threaded)
(2)عمليات متعددة سلاسل المهام(multi Threaded)

وتدعم العمليات المتعددة سلاسل المهام (multi Threaded)تنفيذ سلسلة مهام بشكل متوازي على عدة أنظمة حاسوبيه ويحدث عن طريق تعدد المهام (multitasking)أو ما يدعى بتجزئة الزمن (time slicing)

في الوقت الحالي تدعم العديد من أنظمة التشغيل تجزئة الزمن وتعدد المهام أو التنفيذ متعدد المعالجات(multiprocessor thread) عن طريق منسق العمليات (scheduler).

ومن الأمثلة التي توضح الإختلاف بين(multi Threaded) و(single Threaded):[36]

[35] بدور دهش الدهش و إيمان البلالي
[36]قطر الندى السماعيل

في الــ (single Thread) عندما نبحث في الويب (web server) فإنه لا يقبل إلا لعميل واحد فقط أن يستفيد في الوقت الواحد ولحل هذه المشكلة قمنا بإيجاد أكثر من سلسلة مهام (multi Threaded) في نفس العملية حتى تستجيب لطلبات العميل .

(من المعلوم أن (single threaded) تستطيع عمل فقط عملية واحده على (CPU) , بينما الــ (multi Threaded) تستطيع عمل العديد من العمليات على (multi-CPU) , ميزة (multi thread)أنها تستطيع زيادة بناء (multi process) .)

المميزات من إيجاد الــ (multi Threaded)[37]:

(1) رفع مستوى الاستجابة للمستخدم (responsiveness) :

إن إنشاء أفرع للمهمة النشطة يرفع مستوى الإستجابة للمستخدم حيث أن المستخدم أثناء تحميل صفحة من الإنترنت ولتكن على سبيل المثال صفحة موفري خدمة الرسائل الإلكترونية يمكنه أن يسجل الدخول لبريده الإلكتروني بينما الصفحة الرئيسية لموفر الخدمة لازالت تقوم بتحميل بعض الصور، كما يمكن للمستخدم أثناء استخدام أحد برامج معالجة النصوص أن يعطي أمر طباعة وفي نفس الوقت يمكنه طلب تدقيق إملائي ونحوي مما يساعد على رفع مستوى الإستجابة بشكل ملحوظ خلاف لو لم يكن هناك أفرع للمهمة النشطة لما تمكن المستخدم من إجراء أكثر من أمر أو تفاعل مع الجهاز في نفس اللحظة

(2) تقاسم الموارد (resourse sharing):

عند أنشاء أفرع للمهمة النشطة فإن هذه الأفرع تشترك افتراضيا في موارد النظام مثل الذاكرة مما يساعد على توفير هذه الموارد أي أنه يتم تقليل المساحة المحجوزة للمهمة الواحدة حيث أن جميع أفرعها تشترك في هذه المساحة مما يمكن المهمة النشطة من إجراء العديد من الوظائف في أفرعها المختلفة في نفس الحيز المحجوز ومن هنا يظهر تعددية الوظائف للمهمة الواحدة خلاف لو لم يكن هناك أفرع لاضطررنا لإنشاء

[37]أشواق العتيبي

أكثر من مهمة واحدة بعناوين متعددة في الذاكرة لأجراء تلك الوظائف مما يتسبب في إهدار موارد النظام , من المعلوم أن الـــ (threads) يتقاسم الذاكرة والمصادر في نفس العملية والميزة من هذا التقاسم أنه يسمح للتطبيقات أن تملك العديد من سلاسل المهام النشطة في نفس العنوان(address space) .

(3) الاقتصاد (economy) :

من عملية إنشاء مهمة نشطة يستغرق وقت من النظام وذلك بسبب وجوب مراعاة حدود تلك المهمة حتى لا يتم اختراقها من قبل مهمة آخرى وكذلك عملية التبديل بين المهام المختلفة لرفع مستوى التز امنية في الأداء يضيف حملا أكبر على كاهل النظام . أما في إنشاء الأفرع لاتكون الإحترازات متشددة حيث أن جميع الأفرع تشترك في نفس مساحة الذاكرة إضافة إلى أن التبديل يكون أسرع بحوالي خمس مرات من التبديل بين المهام النشطة في بعض أنظمة التشغيل ويكون إنشاء المهام فيها أبطأ بثلاثين مرة من إنشاء الأفرع مما يجعل أنشاء أفرع متعددة للمهمة الواحدة لإجراء عدة وظائف مترابطة أرخص للنظام من إنشاء عدة مهام لإجراء تلك الوظائف .

(4)الاستغلال الأمثل في حالة وجود أكثر من وحدة معالجة مركزية في النظام :

تعدد المعالجات في النظام يرفع مستوى أداء الجهاز بشكل عام حيث أنه يمكن تنفيذ أكثر من مهمة نشطة في نفس الوقت مما يساعد على إنتاج العمل بشكل أسرع إلا أن المهمة الواحدة لم تستفد من هذه الميزة لأنه سيتم تنفيذها بشكل متتابع ولكن إذا قمنا بإنشاء أفرع للمهمة الواحدة سيتم تسريع تنفيذ المهمة نفسها حيث أن كل فرع من أفرعها تتم معالجته في معالج منفصل مما يتيح لها الاستفادة من الموارد على الوجه الأمثل .

مكتبة سلسلة المهام:

توفر مكتبة سلسلة المهام للمبرمج واجهة برمجة تطبيقات لخلق وإدارة سلسلة المهام.

- هناك طريقتين أساسيتين لتطبيق سلسلة المهام هما:

1- توفير المكتبة كاملة في مساحه المستخدم بدون أي دعم من النواة:
كل الرموز وهياكل البيانات الخاصة بالمكتبة موجودة في مساحة المستخدم.
2- إنشاء مكتبة على مستوى النواة بدعم مباشر من نظام التشغيل:
كل الرموز وهياكل البيانات الخاصة بالمكتبة موجودة في مساحة النواة.

يوجد ثلاث مكتبات تستخدم حالياً هي[38]:

1- POSIX Pthread
توفر إما مكتبه على مستوى المستخدم أو على مستوى النواة
2- WIN32
مكتبه على مستوى النواة وهي متاحة في نظام النوافذ.
3 - Java. Pthread
واجهة برمجة التطبيقات لسلسله مهام الجافا تتيح إنشاء وإدارة سلسلة المهام مباشره في برامج الجافا.

وهناك طريقتان لإنشاء تجزيء للعمليات[39]:

1- (User threads) تجزيء المستخدمين :
وهي عملية تجزيء للبرامج من خلال المستخدمين داخل برنامج معين دون المرور بلب النظام (Kernel) بواسطة مناداة الدوال المكتبية (library function) وتعتمد هذه الطريقة علـــى نوع نظام التشغيل المستخدم ولا تتحكم في أجزاء النظام المادية (hard ware).
وفائدة هذا النوع انه يخفف من الضغط على لب النظام (Kernel)

2- (Kernel threads) تجزيء النظام :

[38] أمل الحماد
[39] فاطمة الفرج

وهي عملية تجزيء للبرامج من خلال لب النظام (Kernel) بواسطة مناداة الدوال (function) المسؤولية عن إنشاء تجزيء للعمليات وتسمي هذه الطريقة (system call)

ومن الأمثلة عليها :

- Windows XP/2000, Solaris, Linux, Mac OS X
- وكلتا الطريقتين تنفذان من خلال لب النظام (Kernel)

(هناك نوعان للـــ(thread)سلسلة المهام[40]:

- سلسلة مهام مجال المستخدم (user_space thread) : تُنتج في مستوى المستخدم .
- سلسلة مهام النواة (kernal) : معموله من قبل النواة (kernal)

ويتم الربط بين سلسلة مهام مجال المستخدم (user_space thread) و سلسلة مهام النواة (kernel) عن طريق ثلاث طرق .

نماذج الخيوط المتعددة (multi Threading Models)[41]:

(1) متعدد إلى واحد Many-to-one Mode :

في هـــــذه الطريقة يوجـــــــد العديد من (user threads) تنتـــــقل إلى (kernel threads) , وهذه الـــ (thread)تدار في مساحة المستخدم عن طريق (library) , لكن هذه الطريقة الواحد يعني لا يمكن أن يكون تنفيذ العديد من العمليات بشكل متوازي.

---

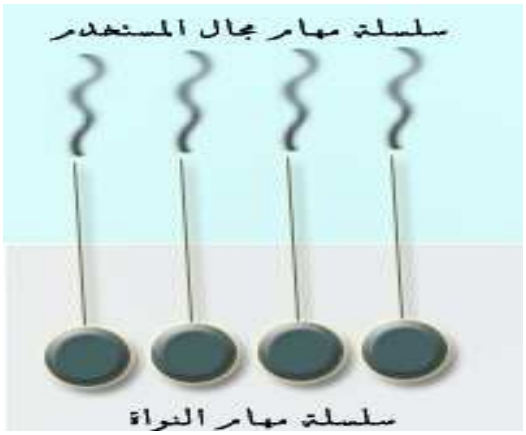
[40] منى الماجد
[41] غادة القرعاوي

عيوبها :فيها إغلاق بسبب أنه فقط واحد من سلسلة المهام يستطيع المرور إلى (kernel)في الوقت ذاته

وهذه الطريقة تستخدم في:

.Solaris Green Threads, GNU Pthreads

One-to-one Mode(2) من واحد إلى واحد



في هذه الطريقة كل (user threads) يملك (kernel threads) له لوحده ,

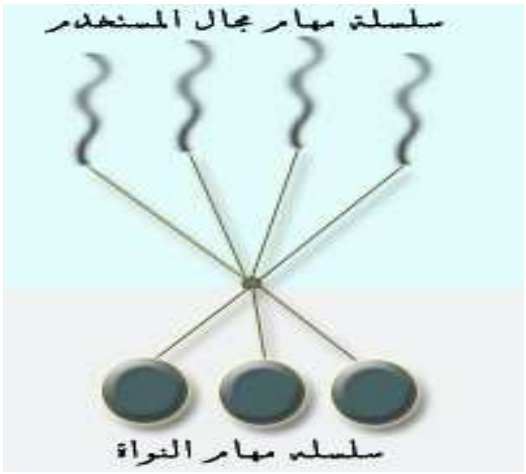
وبهذه الطريقة يمكن لكـــــل سلسلة مهام أن تعمل بشكل متوازي مع الأخرى,ولذلك لابد مع كل (user threads) جديد أن نوجد (kernel) متوافق معه , هذه الطريقة أفضل من السابقة .

مشكلة هذه الطريقة:

أنه في حالة تعطل الـــ (kernel) لا يمكـــن لـــ (user thread) التابعة له أن تستفيد من أي (kernel) أخر و تسبب ضغط على لب النظام (Kernel).

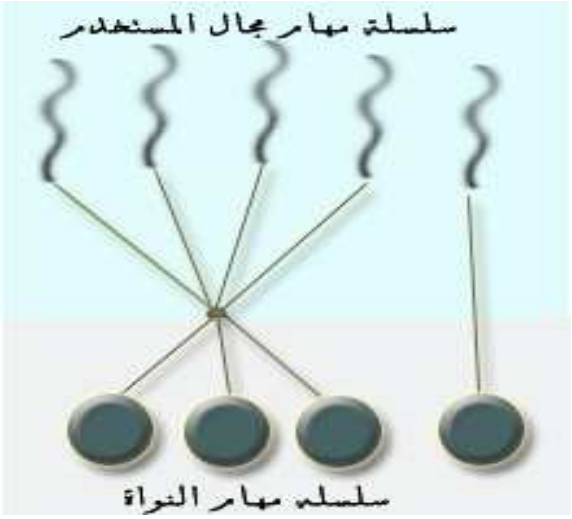
وتستخدم هذه الطريقة في: Linux, windows 95,98,NT,2000,XP .

(3) من متعدد إلى متعدد Many-to-many Model :



في هذه الطريقة يوجد العديد من (user threads) التي تملك نفس عددها أو أقل منها (kernel threads) وهذه الطريقة أفضل من السابقتين لأنه كل (user threads) ممكن يتعامل مع أكثر من (kernel).

(4) نموذج الطبقتين (Two-level Model) مشابه لـ M:M (Similar to M:M)



كما أنه في بعض الأحيان يجمع بين طريقتين وهذا النوع يعطي صلاحية لأن ينعزل كل جزء من أجزاء البرنامج ويتصل بلب نظام خاص به.

وهذا مدعوم في أنظمة التشغيل:-IRIX,HP UNIX UX and Tru64 .)

تحتاج (M: M) و(Two-level-model) إلى وسيلة اتصال أو ربط , لكي تحدد العدد المناسب من الـ kernel thread التي تريدها لعمل نشاطاتها , لذلك تستخدم LWP وهي عبارة عن تراكيب بيانات يقوم الـ kernel بخلقها

وذلك لربط الـــ kernel thread بالـــ user thread .

الفرق بين خيوط قلب النظام وخيوط المستخدم[42]

User Level Threads vs. Kernel Level Threads

	User level thread	Kernel level thread
الإدارة	يتم بواسطة مكتبة الـ User level thread	تتم بواسطة نظام التشغيل
المساحة	مساحة المستخدم	مساحة نظام التشغيل
تدخّل نظام التشغيل	لا يوجد تدخّل من نظام التشغيل	مباشرة مدعومة من قِبل نظام التشغيل
الميزات	سرعة الإنشاء والتحكم	إذا كان thread يؤدي إلى منع مناداة النظام فإن قلب النظام (النواةKernel) يمكن أن يحدد thread آخر في التطبيق للتنفيذ.
المساوئ	إذا كان قلب النظام وحيد(single threaded) ،فإن أي user thread أدى لمنع مناداة النظام سيتسبب في منع كل العملية، وحتى لو وجد other threads متاحة للتشغيل في إطار التطبيق.	أبطأ في الإنشاء والتحكم

[42] حنان وحيد الهندي

أمرا التفرع والتنفيذ[43]:

fork() and exec()

عند إنشاء مهمة نشطة بأمر التفرع فأنه يتم نسخ جميع محتوى المهمة الأم إلى المهمة المنشاة
التساؤل المطروح هنا : هل سيتم نسخ جميع الأفرع لهذه المهمة إلى المهمة المنشأة أم سيتم فقط نسخ الفرع الذي أصدر الأمر بإنشاء مهمة جديدة ؟

إجابة التساؤل تعتمد على التطبيق المتضمن لهذه المهام , إن تم إصدار أمر التنفيذ مباشرة بعد أمر التفرع فإن محتوى المهمة المنشأة سيتحول بكامله إلى البرنامج المرسل له في أمر التنفيذ مما يعني أنه لا أهمية لأفرع المهمة الأم لهذه المهمة المنشأة ففي هذه الحالة سيتم فقط نسخ الفرع الذي أصدر أمر التفرع في المهمة الأم إلى المهمة المنشأة , أما إذا لم يتم مناداة أمر التنفيذ بعد أمر التفرع مباشرة فإنه يتوجب نسخ جميع أفرع المهمة الأم إلى المهمة المنشأة .

(Cancellation) الإلغاء:

عندما تكون هناك رغبة في إنهاء التفرع على غير المناخ الطبيعي وهو إتمام وظائفها فإن هذا الإنهاء يكون إلغاء في بعض الأحيان يرغب المبرمج في إلغاء التفرع لعدم أهميته كأن يكون أنشأ أكثر من فرع للبحث في قاعدة بيانات , فعندما يعود أحد هذه الأفرع بالنتيجة فإنه لا حاجة لبقية الأفرع فيلجـــــــأ المبرمج إلى إلغائها مباشرة لتحرير موارد النظام المحجوزة لها , إلا أنه بهذه الطريقة لن يتم تحرير جميع هذه الموارد علاوة على أنه قد يكون هذا الفرع المراد إلغائه "الهدف" يعمل على تعديل قيمة مستخدمة من قبل الأفرع الأخرى مما سيؤدي إلى أخطاء لا حصر لها , لذلك فُضل أن يكون الإلغاء مؤجـــــل مما يعني أن يعطى المجال للفرع الهدف بأن ينهي نفسه في اللحظات الأكثر أماناً وذلك يكون بإعطاء علم يدل على إمكانية إنهائه في هذه اللحظة أو الانتظار قليلاً ريثما يكون الوضع أكثر أمناً .

---

[43] حنان وحيد الهندي

ويوجد هنـــــاك حالتين مختلفتين عند الإلغـــــاء[44]:

1-(Asynchronous cancellation) :
وهذي تعني أن سلسلة المهام تلغى مباشرة بمجرد طلب إلغـــــائـــــها.
2-(Deferred cancellation):
وهذي تعني أن سلسلة المهام لا تلغى مباشرة إلى أن تنتهي من عملها الذي تقوم به لأنه عند إنهائهـــــا قد تؤدي إلي تضرر الملفات.

مثال على طريقة كتابته:

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

لإيقاف عملية في نظام لينكس يوجد عدة طرق[45]:

1- CTRL-C (for foreground processes)
هذا الأمر لإيقاف العملية الحالية – التي في الواجهة

2-Kill process-identifier (for Background processes)
وهذه ابسط وأسهل وأنظف طريقة لإيقاف أو قتل عملية

وإذا أردتي معرفة الـــ Process-identifier
باستخدام الأمر ps
سوف يظهر لنا جميع العلميات التي تعمل لدينا في النظام و أول عمود هو الذي به الرقم الخاص لكل عملية process-identifier

وإذا لم ينجح هذا الأمر استخدمي
Kill -1 process-identifier

---

[44]إيمان البلالي
[45]بتول الحناكي

هذا الأمر سوف يخبر العملية بأن تستسلم لأنها سوف تعطيها إشارة بأنك قد خرجت من النظام فلابد لها من الاستسلام بدلا من أن تحاول إكمال عملها وهذا الأمر سوف يقتل كل الـ child التابعين لهذه العميلة

Kill -9 process-identifier

وهذا الأمر سوف يوقف العميلة ولكنه لن يوقف عمل أي child تابعين لها

وتستطيعين أن توقفي عمليتين في نفس الوقت

Kill process-identifier process-identifier

((Signal handling) حامل الإشارة[46]:

الإشـــــارة تستخدم في نظام اليونكس لإعطاء ملاحظه لـــــ العملية في حالة حدوث شيء ما مثل الإلغاء , الانقطاع وقد تستقبل الإشارة إما متزامنة أو غير متزامنة.

النوع الأول : (التزامن) الإشارة تكون بداخل العملية .

النوع الثاني: (غير متزامن) الإِشـــــــارة تأتي من خارج العملية.

كل إشارة قد تُحمل بواسطة أحد الحاملين:

1- (default): قد يعالج حاله واحدة أو حالتين فقط.

2-(user-defined): قـــــــــــد يعالج حـــالات كثيـــــــــــرة.

أمثلة على الـــــــ Synchronous Signals:

هذا النوع من الإشارات يظهر عند الدخول غير المسموح به لجزء من الذاكرة أو عند القسمة على الصفر مثلا.

---

[46] رويدا العندس

وبهذا إذا قام برنامج معين بحدث من هذا النوع, لابد أن تظهر إشارة , وفي هذا النوع من الإشارات , توصل الإشارة لنفس العملية التي تسببت بظهورها ولهذا السبب سميت بالـــــ Synchronous Signals.

أمثلة على الـــ Asynchronous Signals:

في هذا النوع, السبب الذي يؤدي لظهور الإشارة يكون (خارج العملية) .
مثلا,عند الضغط على أوامر معينة مثل Control + c وهذا يؤدي إلى إنهاء عملية ما بشكل مفاجئ.

تـــــأثيـــــر الإشـــــــارة له أربع حالات:

1- الإشارة ترسل فقط لكل سلسلة المهام التي حدث فيها الخطأ فقط.
2- الإشارة ترسل لكل سلسلة مهام موجودة في العملية.
3- الإشارة ترسل لسلسة مهام معينه في العملية.
4- الإشارة ترسل لسلسلة مهام مخصصه فقط لاستقبال الإشارات.

على سبيل المثال, الإشارات من نوع Synchronous Signals لابد أن ترسل إلى الجزء الذي سبب ظهـــــــور الإشارة وليس لباقي أجزاء العملية.
أغلب أنظمة اليونكس تسمح للـــــ thread بـــــــأن يقوم باستقبال إشارات معينة ويحجب إشارات أخرى. ولهـــــــذا في بعـــــض الأحيـــــــــان يتـــــم إرســـــــال Asynchronous Signal فقط للأجزاء التي لم تحجبها . وبما انه لابد من معالجة الإشارة فإنه يتم تسليمها لأول thread لم يقم بعملية الحجب.
في أنظمة الويندوز, يتم معالجة الإشارات عن طريق الـــــــ APCs (Asynchronous procedure calls).
حيث تقوم هذه الخاصية بالسماح للـــــــ user thread بتحديد (function) ليتم مناداتها عندما يستلم جزء معين التنبيه لظهور حدث.ويتم التسليم لجزء معين بدلا من التسليم لكل العملية.

مثال على استخدام الإشارة[47]:

عندما يحدث للجهاز حالة تعليق فإن ctrl+z أوctrl+c ستوقف التعليق وتعيد العمل للجهاز مرة أخرى.
أما عند عملي في الخلفية فإن ctrl+z لن تنفع لذلك سوف أكتب kill
وإذا كنت أعمل في الــ pash فيجب أن اعرف رقم الــ pash حتى أعمله kill وأكتب : kill ثم رقم الpash -->(kill 9) .)

Thread pool:

في السابق قبل عمل ال (pool) كان كل عميل جديد ننشأ له سلسلة مهام جديدة مثل عندما يكون لدينا مائه عميل حيث ننشأ له مائه من سلسلة المهام وهذا العمليه كانت لها العديد من العيوب .
وهـــــــــي:
1- تستهلك وقت لإنشاء سلسلة المهام (cpu time) .
2 - تأخذ مســـــاحه من العملية .

لذلك تم إيجاد (pool) وهي إيجاد عدد محدد من سلسلة المهام عندما نبدأ العملية وننقلها إلى (pool) وعندما يتم طلب خدمه فإن سلسلة المهام تأتي من الــ (pool)إذا كانت ممكنه وتمر لخدمة الطلب , وعندما تكمل عملها تعود إلى الــ (pool) لتنتظر طلب جديد .

محاسن استخدام الــ(pool) :

1- أن طلب الخدمة من سلسلة مهام موجودة أسرع من انتظـــــار إنشاء الخدمة سلسلة مهام جديدة.
2-يوجد في الــ(POOL) عدد محدد من سلسلة المهام منشأه في نقطة ما .. أفضل من إنشاء عدد لا نهائي من السلسلة المهام.

---

[47]منى حكمي , حليمه حكمي ، قطر الندى السماعيل

طرق تحديد عدد الأجزاء المسموح بها:

يتحكم بها عدة أشياء منها عدد الـــــــ(CPUs) المتوفر في النظام , حجم الذاكرة وعدد الطلبات المتوقعة من المستخدم.
وهناك أنظمة ذكية والتي تقوم بدورها بتحديد العدد المطلوب تبعا للاستخدام وهذا ينتج (pool) أصغر وبالطبع يقوم بتوفير جزء من الذاكرة المستخدمة عندما يكون الضغط قليل على النظام.

بالنسبة للبرامج المتعددة الأجزاء هنا مشكلة التواصل بين قلب النظام والـــــــ(thread library) والتي تتطلب من النظام بالقيام بتحديد عدد من أجزاء قلب النظام للقيام بعملية الارتباط.

هناك عدد من أنظمة التشغيل تقوم بوضع بيانات تتوسط بين أجزاء المستخدم و أجزاء قلب النظام. هذه البيانات والتي تعرف بالـــــ Lightweight processأو LWP عبارة عن معالج خيالي والذي يقوم البرنامج الذي يكون تحت التشغيل بجدولة جزء المستخدم عليه .
وبهذا كل LWP تكون مرتبطة مع جزء من قلب النظام.

قد يتطلب برنامج معين اي عدد من الـــــLWP لكي يتم تشغيله بطريقة صحيحة.

على سبيل المثال:

في حالة وجود برنامج يعمل على معالج واحد , أذن لن يتم تشغيل أكثر من جزء في المرة الواحدة ولن يحتاج إلى أكثر من LWP واحد.

Scheduler Activation[48]:

طريقة للتواصل بين الـــUser thread library وقلب النظام

طريقة عمله:

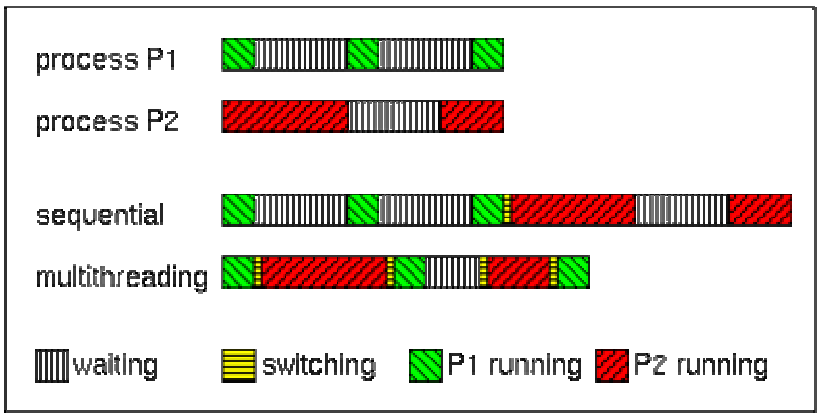
يقوم قلب النظام بتزويد البرنامج بمجموعة من الـــــLWP وعندها يستطيع البرنامج بالقيام بجدولة أجزاء المستخدم على المعالج الخيالي .

Upcall: هي الحالة المعروفة عندما يقوم قلب النظام بإعلام برنامج معين بظهور حدث جديد.

طريقة عمل هذه الخاصية:

تتم معالجة التنبيهات القادمة من قلب النظام من قبل Upcall handler والتي لابد من أن تعمل على المعالج الخيالي.

مثال على حدوث عملية Upcall: عندما يقوم قلب النظام بالتنبيه أن هناك جزء معين سوف يتم إيقافه , في هذه الحالة سيتم تعيين معالج خيالي أخر لهذا البرنامج.



[48] رويدا العندس

context switch and multi-threading[49]:

عند استخدام multi-threading فان TCB يعتبر جزء من Context switch , صف الانتظار والصف الجاهز المستعد يحويان مؤشرات تشير إلي TCB مبدل السياق وهو المسؤول عن عمل نسخة لحالة وحدة المعالجة المركزية من وإلى TCB .

كيفية عمل مبدل السياق:

مبدل السياق بين تشعبين ينتمون لعملية واحدة : في هذه الحالة لا نحتاج لتغيير فضاء العناوين .

مبدل السياق بين تشعبين ينتمون لعمليتين مختلفتين : سوف نحتاج لتغيير فضاء العناوين .

بعض أسباب استخدام المهام المتعددة (threads) في تصميم نظام التشغيل[50]:

1- تقاسم الموارد : حيث أن مهام البرنامج الواحد تتشارك في أجزاء عده كالذاكرة .

2- توفير الوقت : حيث يتم عمل عدد من المهام في نفس الوقت .

مثل : عندما نكتب ببرنامج الـــword في وقت تشغيله يصبح هو process وعندما نكتب فيه تكون هناك عمليات تعمل بنفس وقت عمليه الكتابة

مثل تنفيذ save every two min & spell check هنا عمل شيئين in one process.

3-العملية(process) مع المهام المتعددة تجعل الخادم(server) يقوم بعمله بفاعلية أكبر.

4- بما أن المهام المتعددة تتشارك في البيانات(data) فإنها لا تحتاج (interprocess communication).

5- عند إيقافه أو إنهاءه يأخذ وقت أقل من processes .

6- لا حاجة للاستعانة بـــ kernal أو إخباره بأي عملية تتم عن طريق thread لأنه لا يعلم بوجودها .

---

[49] منى الماجد

[50] بدور دهش الدهش , إيمان البلالي ، أفنان البحيري

المهام المتعددة رخيصة وذلك لأن[51]:

1- تقلل التكاليف من حجز الذاكرة حيث تكون مشتركه لعدد من المهام.
2- نحتاج فقط إلى(stack) و أماكن تخزينية في (register) لذا فإن المهام المتعدده رخيصة الإنشاء.
3- المهام المتعددة تستخدم جزء بسيط من مصادر نظام التشغيل عند عملها حيث لا تحتاج إلى(address space) جديد ولا بيانات عامة (data global) ولا (program code).
4- التبديل سريع عندما نعمل مع المهام المتعددة وذلك لأننا نحتاج فقط إلى تخزين أو إعادة تخزين (pc program counter) , sp (stack pointer) , register.

[51]إيمان البلالي