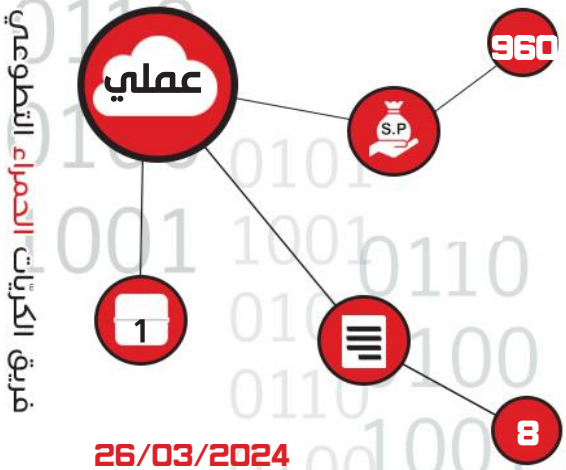


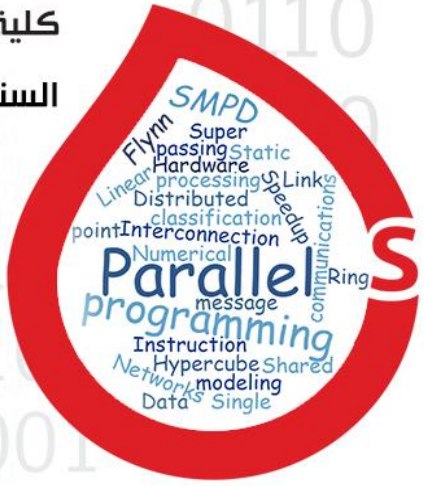
كلية الهندسة المعلوماتية

السنة الرابعة

البرمجة التفرعية

vs

البرمجة التسلسلية



م. عمار المصري

26/03/2024

RB Informatics;

البرمجة التفرعية

بسم الله الرحمن الرحيم

في السنوات الدراسية السابقة وكل ما تعلمناه في مجال البرمجة كان عبارة عن برمجة تسلسلية، في هذه المادة سوف نتعلم ما هي البرمجة التفرعية وكيفية كتابة البرامج التفرعيّة باستخدام PVM

محاور المحاضرة:

- الفرق بين البرمجة التفرعية والتسلسلية
- Flynn's Classification
- Thread
- Parallel virtual machine

البرمجة التسلسلية Serial Computing

- في السنوات السابقة عندما تعلمنا لغات البرمجة C++ Javag وقمنا بكتابة برنامج بلغة معينة فهو عبارة عن مجموعة من التعليمات البرمجية التسلسلية ضمن الـ main حيث يتم تنفيذ تعليمة تلو الأخرى على معالج وحيد بالتالي يكون مسار التنفيذ وحيد نعبر عنه بالتابع ()main (خلال لحظة زمنية معينة يتم تنفيذ تعليمات برمجية معينة).

ولكن نواجه مشاكل اثناء التنفيذ:

أي برنامج لدينا بالـ System له resources على مستوى CPU وعلى مستوى memory

بالتالي هذا البرنامج بحاجة لقراءة البيانات والمعطيات الموجودة بالذاكرة RAM في بعض الأحيان يكون لدينا Data موجودة بالـ Cache memory وبعضها موجود بالـ Hard disk

حيث كلفة الوصول إلى القرص الصلب زمن بطيء جداً يستهلك الكثير من الموارد والزمن والتنفيذ بالتالي نواجه مشكلة:

- كلفة الوصول إلى الموارد المحلية **مثل الذاكرة:**

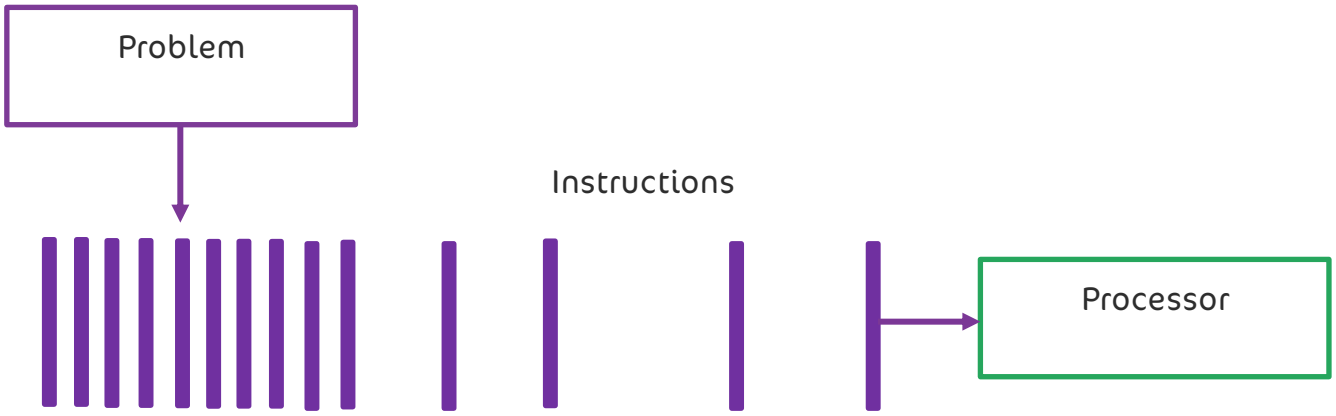
زمن الوصول إلى القرص الصلب بطيء جداً بالمقارنة مع زمن التنفيذ CPU وايضاً نواجه مشكلة عند الاتصال على مستوى الشبكة حيث يستهلك وقت طويل وله كلفة زمنية عالية

- كلفة الاتصال الشبكي (الموارد الخارجية):

الزمن اللازم لتحقيق الاتصال مع المورد البعيد (جهاز أخر) طويل نسبياً ضمن البرامج التفرعيّة

Serial Computing

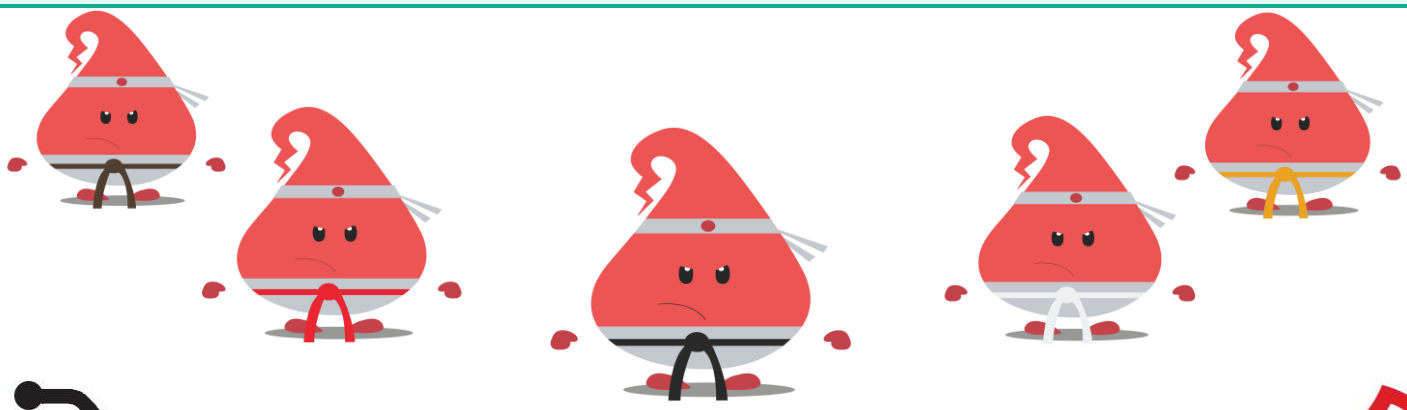
- آلية تسلسلية في تنفيذ البرامج تتمثل بمسار تنفيذ واحد يمثله التابع main، حيث يتم تنفيذ التعليمات الواحدة تلو الأخرى.
- يتم تنفيذ التعليمات ضمن البرنامج على معالج وحيد، حيث يتم تنفيذ تعليمة وحيدة خلال أي لحظة زمنية.
- نواجه أثناء التنفيذ تعليمات مكلفة زمنياً إما تحتاج الوصول للموارد المحلية (الذاكرة)، يتمثل زمن التأخير في الزمن اللازم للوصول إلى القرص الصلب البطيء جداً بالمقارنة مع زمن تنفيذ الـ CPU.
- أو تعليمات تحتاج الوصول للموارد الخارجية (الاتصال الشبكي)، يتمثل زمن التأخير في الزمن اللازم لتحقيق الاتصال مع المورد البعيد.



بالتالي مقيّدون ببرنامج يتم تنفيذه على معالج واحد ومقيّدون ببرنامج له تعليمات برمجيّة مكلفة زمنياً سواءً عند الاتصال بالموارد المحلية (كالذاكرة والقرص الصلب) أو الخارجية (على مستوى الشبكة).

ملاحظات:

- بسبب ظهور برامج بخوارزميات ذات تعقيدات كبيرة وتطلّب حسابات ضخمة وبعض البرامج تطلب معالجة requests على التوازي ظهر مفهوم البرمجة التفرعيّة وهي:
- تقسيم البرنامج إلى أجزاء منفصلة يمكن حلّها بشكل متزامن حيث كل جزء يتكون من تعليمات متسلسلة يتم تنفيذها بمسار مختلف على معالجات مختلفة.
- بالتالي البرمجة التفرعية هي الاستخدام المتزامن لموارد متعددة لحلّ مشكلة حسابية ضخمة. حيث يتم تنفيذ البرنامج في وقت أقل باستخدام موارد حوسبة متعددة مقارنة بمورد حساب واحد.



البرمجة التفرعية Parallel Programming

- هل من الضروري اذا تم تنفيذ برنامج بشل تفرعي يكون أسهل وأسرع من البرنامج التسلسلي؟

فرضاً لو لدينا tool برمجية معينة قادرة على تنفيذ برنامج تفرعي بشكل تسلسلي ففي بعض البرامج يكون الخرج الخاص فيها أسرع إذا تم تنفيذها بشكل تسلسلي لكن في حالة البرامج التي فيها تعليمات حسابية معقدة يتم تنفيذها حصراً بشكل تفرعي...

عملية البرمجة التفرعيّة

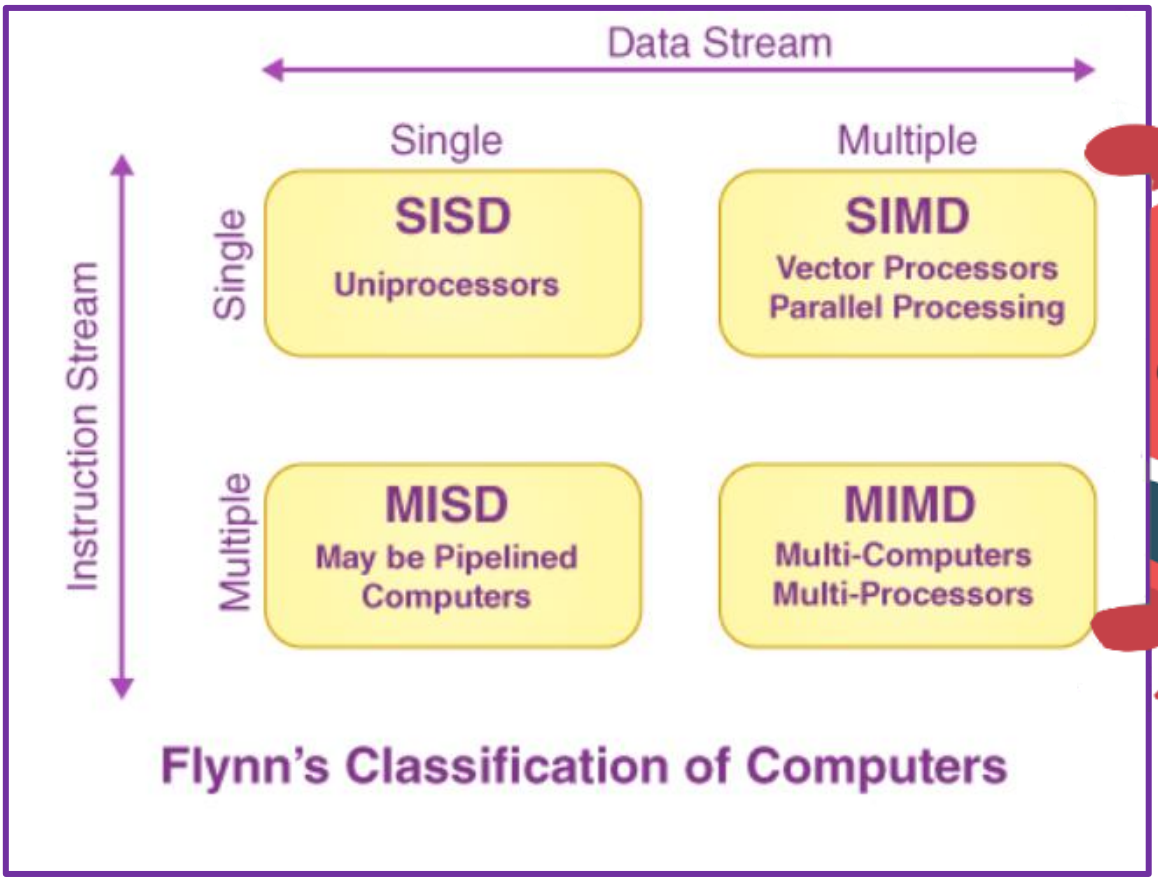
- هي عملية توافق ما بين Code الذي يتم كتابته أياً كانت اللغة البرمجية المكتوبة وما بين Physical resources الموجودة على الجهاز

فمثلاً يجب أن يكون لدينا معالجات متعددة عند تنفيذ برامج تفرعية (تكون قادرة على تنفيذها).

كيف يتم تصنيف الأجهزة اذا كانت قابلة للتعامل مع البرمجة التفرعية أم لا؟!

Flynn's Classification

تقسيم الأجهزة حسب إمكانية تعاملها مع الحوسبة التفرعية بناءً على عدد التعليمات البرمجية التي سيتم تنفيذها والمعطيات التي سيتم تنفيذ التعليمات عليها



- Single Instruction, Single Data stream (SISD):
يعبّر هذا النموذج عن الأجهزة التي تملك معالج وحيد حيث يتم تنفيذ عملية برمجية واحدة على دخل وحيد، في الـ SISD تم معالجة تعليمات البرنامج عبر آلية تسلسلية والأجهزة التي تندرج ضمن هذا النموذج تعرف عادةً بالأجهزة التسلسلية.
- Single Instruction Multiple data stream (SIMD):
يعبر هذا النموذج عن الأجهزة التي تملك عدة معالجات حيث يتم تنفيذ نفس العملية على معالجات مختلفة ولكن على اختلاف المعطيات التي سيتم تنفيذ التعليمات عليها (مناسبة لأغراض " Scientific Computing") نظراً لأنها تتضمن الكثير من العمليات التي تتعامل مع المتجهات والمصفوفات على سبيل المثال تعد SIMD مفيدة عند تطبيق ذات خوارزمية العمل على مجموعة متعددة من المعطيات.
- Multiple Instruction Single Data Stream (MISD):
يعبر هذا النموذج عن الأجهزة التي تمتلك عدة معالجات حيث يتم تنفيذ عمليات مختلفة على معالجات مختلفة ولكن جميع هذه العمليات يتم تنفيذها على نفس مجموعة المعطيات
يمكن استخدام هذا النموذج في خوارزميات فك التشفير ولكن بحاجة لعمليات Hashing والتشفير وحساب التوقيع الرقمي لنفس المعطيات وبنفس اللحظة
- Multiple Instruction Multiple Data stream (MIMD):
يعبر هذا النموذج عن الأجهزة التي تملك عدة معالجات حيث كل معالج يقوم بعمليات مختلفة على معطيات مختلفة حيث ستعمل كافة المعالجات بشكل غير متزامن (يمثّل أجهزة الحواسيب الشخصية Multicore).
- إن أبسط شكل من أشكال التفرعية هو Thread

Thread

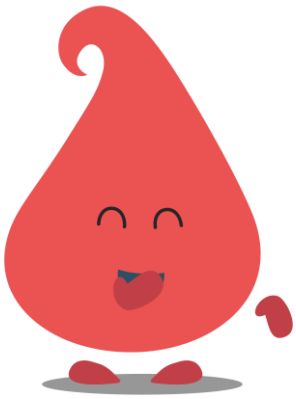
هو إمكانية تشغيل مهمة معينة على أكثر من مسار تنفيذ Single process, multi execution path

- الـ process **يحتوي على** thread **واحدة على الأقل** main
(نطلق مسمى الـ process **على البرنامج قيد التنفيذ المحمل في الذاكرة)**
- **البرمجة التسلسلية يقتصر التنفيذ فيها على** main thread
- **البرمجة التفرعية تقوم بإنشاء** extra threads **ضمن الـ** main thread

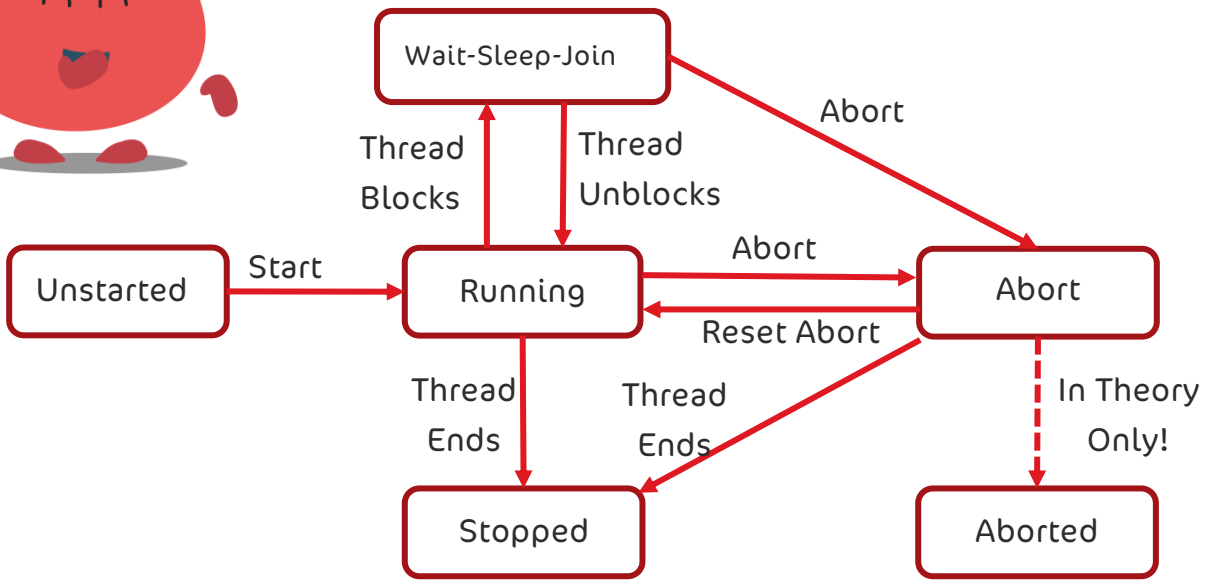
ملاحظات:

- تشترك الـ threads بالـ memory الخاص بالـ process أو يمكن أن نطلق عليها Address space حيث لكل process موجودة لدينا عالـ system يتم حجز Address space ويتم مشاركته من قبل threads
- يمكن تشغيل الـ threads على multicore مختلفة
- يمكن تنفيذ الـ threads بآلية تسلسلية ويمكن تنفيذه بآلية تفرعية

Thread States:



1. Unstarted: يتم خلق object ضمن الذاكرة دون حجز resources
2. Ready: تخصيص ذاكرة ويتم ذلك عند استدعاء تابع ;() Start
3. Run: يقوم المجدول بإدخاله إلى CPU Cycle
4. Block / not run: I/O – sleep – wait
5. Dead: انتهاء الـ thread



- ملاحظات:
  - Sleep: خروج Thread 1 من CPU Cycle ودخول Thread 2 فيصبح Thread 1 في حالة sleep حتى انتهاء تنفيذ thread 2.
  - Wait: هو انتظار thread 1 إشارة من Thread أخرى لاستئناف تنفيذه وإلا يبقى في حالة انتظار.
- سلبيات Thread:
  1. انشاء الـ thread يضع عبئاً على النظام من حيث الذاكرة وموارد CPU.
  2. Shared resources قد تؤدي إلى عدم تناسق البيانات أو مشكلات في مزامنة Threads.
  3. مزامنة الـ Thread تعد حملاً إضافياً على المطوّر.
  4. صعوبة في إدارة الـ Code من حيث تصحيح الأخطاء أو كتابة الـ Code (صعوبة في Testing وDebugging).

ملاحظة:

- آلية اختيار thread للدخول في CPU Cycle أو مسار التنفيذ هي عملية عشوائية وتتم من قبل المجدول بينما يتدخلّ المبرمج في اختيار Thread معينة للدخول في مسار التنفيذ عند وضع Priority بآلية معينة.

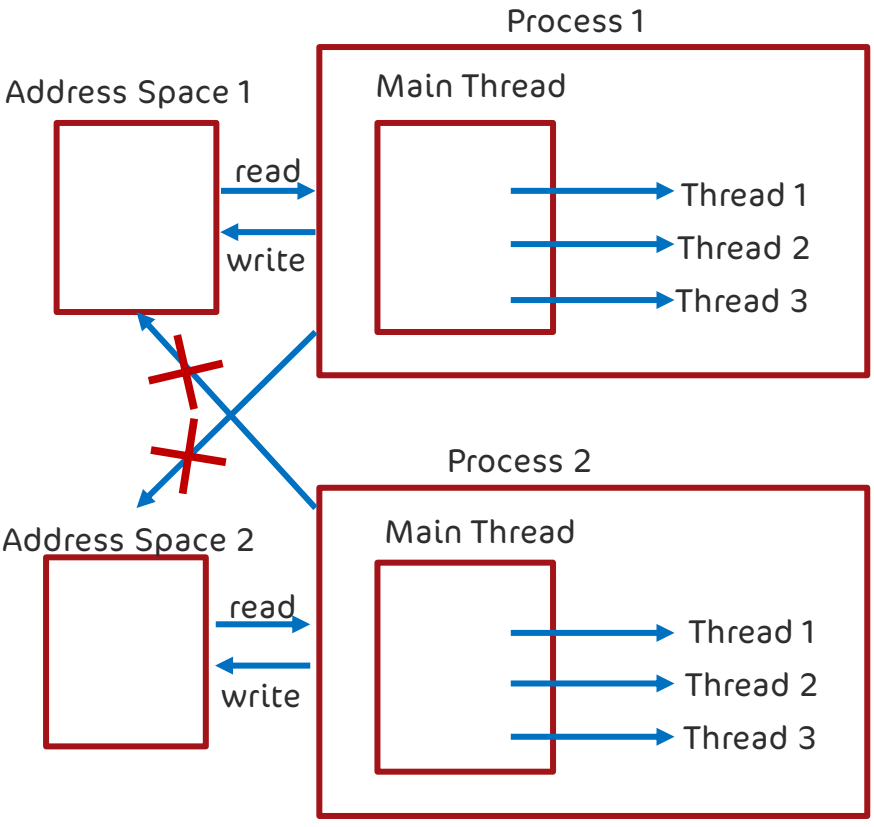
- الآن كيف يمكننا السماح لـ Process 1 بأن تتخاطب مع Process 2 أخرى؟؟

Process Memory:

- من أجل كل Process تقوم بيئة التنفيذ بحجز جزء من الذاكرة RAM كفضاء عناوين (Address Space) خاص بكل Process
- بالتالي تشترك جميع الـ threads ضمن الـ process بالـ Address Space ويتاح لها الوصول المباشر إليه
- يمنع نظام التشغيل أي Process من الوصول لفضاء عناوين (Address Space) لـ process أخرى

Connection Between Processes

- **Pipeline:** تمثل إحدى الحلول لتحقيق اتصال بين برنامجين لكن مشكلتها تتمثل باختصارها الربط بين البرامج التي تعمل ضمن جهاز واحد.
- لإنشاء اتصال بين برنامجين سواءً على جهازين مستقلين أو نظامي تشغيل مختلفين فالحل يتمثل بـ Protocol الاتصال المعياري الشبكي TCP / IP
- التعامل المباشر مع Protocol TCP / IP صعب لذلك تم انشاء Protocols خدمية مبنية عليه توفّر سهولة التعامل من خلال تعليمات مبسطة تحقق الاستخدام الضمني للـ TCP / IP منها الـ protocols:
HTTP, FTP, SMTP, POP3



- HTTP Protocol
مخصص لصفحات الـ HTML
- FTP
مخصص لتبادل الملفات
- SMTP
مخصص لرسائل البريد الإلكتروني
- POP3
مخصص لتخزين رسائل البريد الإلكتروني

Parallel Virtual Machine (PVM)

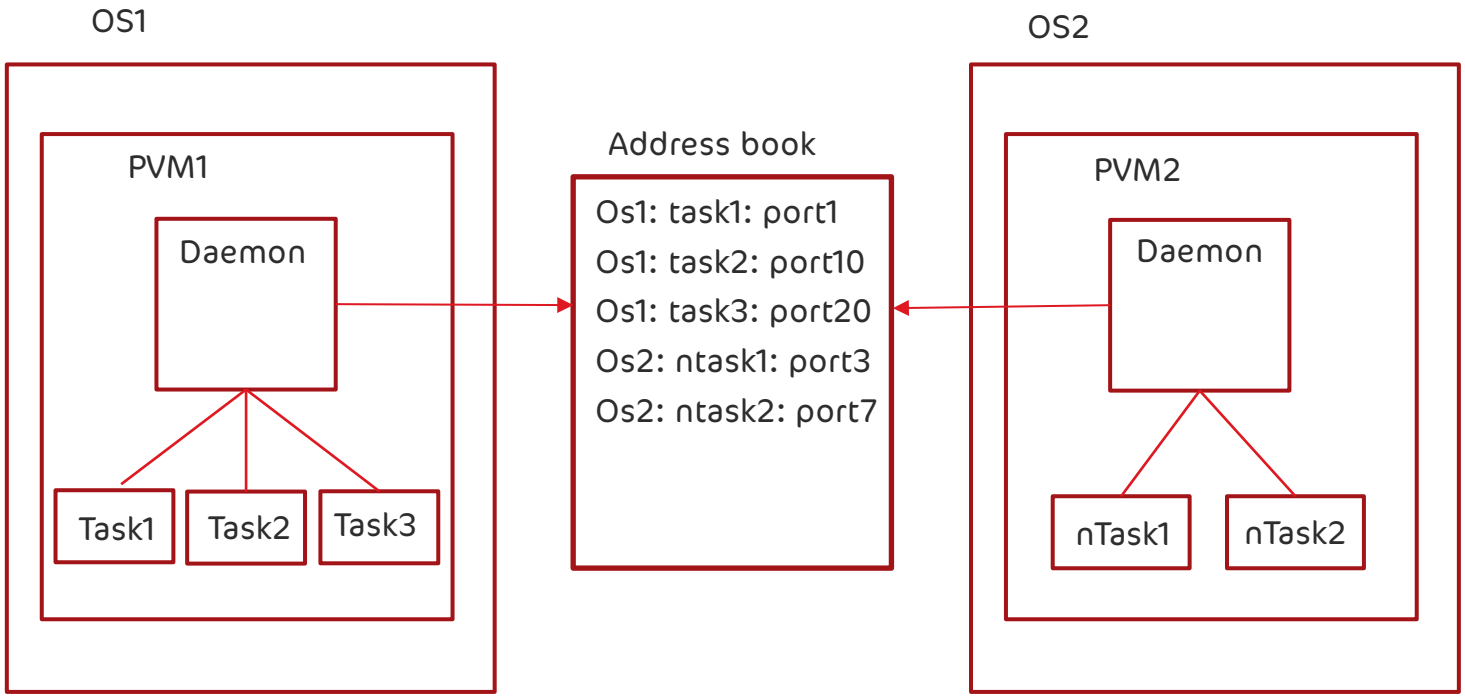
- مكتبة مبنية على TCP/ IP مخصصة لتبادل البيانات من خلال Message Passing بين Two Process.
- يعمل PVM وفق نموذج Client / server متمثلاً PVMD / PVM.
- PVM Client: يمثّل البرنامج المنشأ من قبل المبرمج باستخدام أوامر PVM ليتولى بعدها PVMD مهمة تنفيذها بالتالي على كل جهاز يرغب بالتخاطب الشبكي هناك Deamon يجب أن تكون قيد العمل.

ملاحظة:

- PVMD (Parallel Virtual Machine Daemon)
  Daemon **عبارة عن** Background service **تعمل ك** server **ضمن أي** system **(ويتم استخدامها ضمن** Linux)
- **من خلال** local Daemon **يتم مخاطبة الـ** Processes **على الـ** PC
- **تقوم** PVM **بإنشاء** PVM Task

- كيف أسمح لـ PVM Task على جهاز بالتخاطب مع PVM Task على جهاز آخر؟

Connection Between PVM Tasks



- يملك الـ Local Daemon لكل جهاز على الشبكة دفتر عناوين (Address Book) يحفظ المواصفات الشبكية لجميع الـ PVM Tasks المنشأة على كافة أجهزة الشبكة.

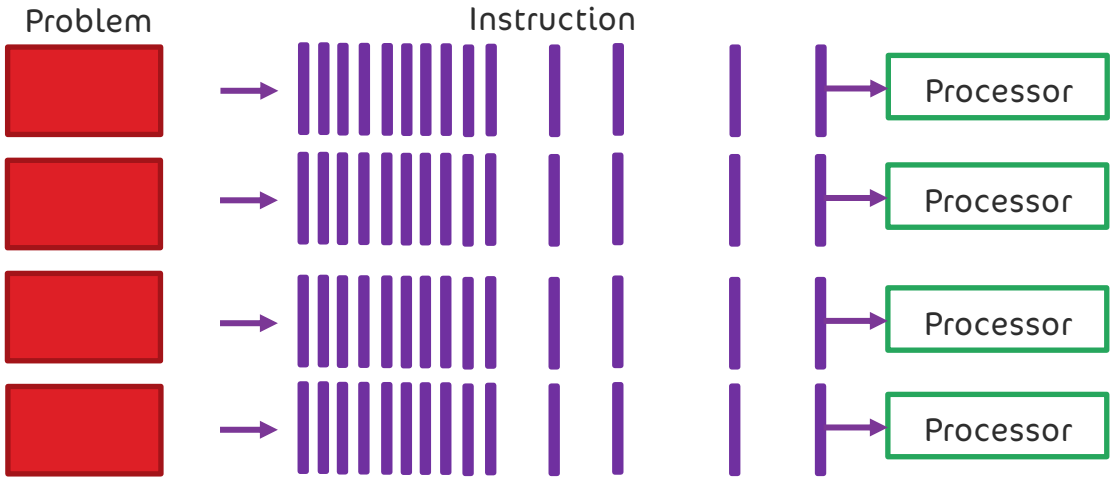
- المواصفات الشبكية ضمن الـ Address Book تتمثل باسم الـ Task واسم الجهاز المنشئ لها ورقم الـ port الذي تعمل عليه (حيث نميز بين أنواع الخدمات الشبكية من خلال الـ port Number)
- عندما تُنشأ أحد أجهزة الشبكة Task جديدة، يضيف الـ local Daemon الخاص بالجهاز المواصفات الشبكية للـ Task الجديدة إلى الـ Address Book الخاص به، ويرسل Broadcast بالتحديث إلى كل Daemon موجود على الشبكة.
- لتحقيق الاتصال بين 2 Tasks منشأتين على جهازين مختلفين يلزم معرفة المواصفات الشبكية للـ Task المراد الاتصال بها.
- أولاً تتخاطب الـ Task1 مع الـ Local Daemon فترسل له اسم الـ Task المراد الاتصال بها "N Task 1" ليعيد لها الـ Local Daemon المواصفات الشبكية للـ N task1 المسجلة ضمن الـ Address Space لديه، تقوم بعدها Task1 بأجراء اتصال مباشر مع N Task 1

ملاحظات:

- لكل جهاز عندنا على مستوى network له Local Daemon.
- كل service تعمل لدينا على مستوى network يتم حجز port لها بشكل عشوائي (عبارة عن رقم من 1024 وما فوق لأنه كل ما تحت 1024 محجوز.)
- عند تطبيق العملية السابقة على نفس الجهاز لسنا بحاجة إلى Local Daemon.

Parallel Computing

- يتم تقسيم البرنامج إلى أجزاء منفصلة يمكن حلها بشكل متزامن، حيث كل جزء يتم تقسيمه إلى مجموعة من التعليمات المتسلسلة.
- يتم وضع كل جزء من التعليمات ضمن مسار تنفيذ منفصل ليتم تنفيذها على معالجات مختلفة، حيث يتم تنفيذ البرنامج في وقت أقل باستخدام موارد حوسبة متعددة مقارنة بمورد حساب واحد.



The End...