Practical parallel programming lectures:

في البرمجة التسلسلية: يكون تنفيذ العمليات عن معالج وحيد واحدة تلو الاخرى
عن خلال وحدة زمنية واحدة يتم تنفيذ عملية وحيدة.

* عملية تنفيذ العمليات واحدة تلو الاخرى يجعل الوصول الى الموارد المحلية عملية بطيئة وصعبة + كلفة زيادة للاتصال الشبكي.

ال PVM تعمل على نظام Windows 7 او Windows XP.

توابع الارسال والاستقبال: المسؤولة عن التواصل بين ال tasks
PVM Communication.

(تجهيز)

خطوات عملية الارسال في PVM : - تهيئة - تحزيم - ارسال
= = الاستقبال في PVM : - استقبال - قراءة الطباعة
- فك تحزيم

* في عملية التهيئة يتم تحديد معلومات الرسالة (تحديد مثل طالبة box ومغلفة (اول ما طالبة) خطوة على المعلومات وتجهيز الطلبية بعدين تغلفوها (تحزيم) ثم ارسال
وهنة بدي ~~نستلمها~~ نستلمها (استقبال) او نقرأ معلومات الطلبية (فتح او طلب وهي تكون مغلفة) وبعدين نفك التحزيم (التغليف)

انواع التواصل: ⟵ Blocking : يتوقف التنفيذ عند الوصول الى تعليمة الارسال والاستقبال حتى يتم تنفيذ العملية

⟵ non Blocking : تتم متابعة تنفيذ الكود بغض النظر عن استلام الرسالة او عدم استلامها ~~...~~ عند وصول تعليمة الارسال الى

* في عملية الاستقبال تتم متابعة التنفيذ بغض

النظر إذا تم أولا المهمة لا يجب انه تم تمرير أي اضافة طالما يتم استلام اي رسالة.

لكل مهمة عندي 2 id ← task id ← pvm_Mytid()

← parent id ← pvm_parent()

عند استدعاء pvm_Mytid يتم الاتصال بـ localdaemon واستدعاء PVMD هو الذي يعطي الـ task id الخاص بها.

- استدعاء pvm_parent() في المهمة الاب يعطي قيمة سالبة (لانه ليس لها اب).
- المهمة الاب يتم انشاؤها في الـ Terminal اما المهام الابناء تم انشاؤها في الذاكرة (ليس لها وحدة خرج) ← ترسل النتائج المهمة الى الاب.
- pvm_spawn() ← تعمل اعادة task id للمهام الابناء.
- البرامج المستخدمة:
  - Hello : برنامج الاب
  - Hello other : عمل برنامج الابن.

* عند استدعاء pvm_Mytid يتم تسجيل المهمة كـ pvm task

cc = pvm_spawn ("Hello_other", (char **) 0, 0, "", 1, &tid)

اسم ملف الابن — ثوابت — عدد المهام

اذا الـ cc كانت قيمة سالبة ← في خطأ في انشاء اي نسخة — مصفوفة task id

pvm_recv : تابع استقبال الرسالة منه taskid / msgtag

لها الرسالة اللي استلمناها (في حال كانت -1 ← ما دهم بجهة الاستلام) / يحدد tag معينة للاستقبال والعكس -1 ← عكسه استقبال اي tag

• ال pvm_recv يبقى قيد الانتظار لحد ما توصل الرسالة ويتم استلامها

في برنامج الاب pvm_recv(-1, -1) عكس ان يستقبل من اي مكان وأي tag.

محصول مع تابع الاستقبال في تابع pvm_bufinfo لقراءة معلومات الرسالة

pvm_bufinfo (cc, (int*)0 (int*)0, & id)

null : تحمل حجم الرسالة / tag الرسالة

taskid : ال task التي ارسلت الرسالة.

pvmupkstr ← تابع فك الترميز.

pvm_exit

كل ما سبق هو في برنامج الاب.

في برنامج الابن : Hello other : (هنا تتم عملية الإرسال)

• ptid ← parent task id = pvm_parent

• pvm_initsend () لتهيئة الارسال

pvm_pkstr (buf) ترميز الرسالة

pvm_send (ptid -1) ال 1 هنا هو Msgtag.

• الطباعة خرج مهمة ما يتم ارسال هذا الخرج الى المهمة الاب (لانه تمثيل Terminal) ويتم الطلاع الخرج ضمنه.
(فوق «الاب» مكتوب: ابن)

• ال pvmd سيعرض اعطاء ID جديد عند تشغيل مهمة جديدة حتى لا يحصل تعارض بين المهام المنتهية والجديدة.

• ضرب المصفوفات تفرعياً:

* قاعدة ضرب المصفوفات: (1) يجب ان يكون عدد الاعمدة الاولى = عدد اسطر المصفوفة الثانية

(2) نضرب كل سطر من المصفوفة الاولى مع كل عمود من المصفوفة الثانية

⇐ لتمثيل هذا البرنامج بشكل تفرعي: نعطي كل ابن سطر وعمود كي يحسب الناتج ويرد الناتج للأب

⇐ في حال كان عندي من المصفوفة الاولى سطرين و 3 اعمدة = = الثانية 3 اسطر وعمودين

⇐ يكون لدينا 4 مهمات ابناء:

السطر الثاني مع العمود الثاني	السطر الثاني مع العمود الاول	السطر الاول مع العمود الثاني	السطر الاول مع العمود الاول

* في هذا النوع من المسائل سنحتاج الى ال tag كثيراً لأنه سيساعدني معرفة كل قيمة اين يجب ان توضع ضمن مصفوفة الناتج.

• هنا سنتعامل مع المصفوفة ~~الأحادية~~ الثنائية على انها مصفوفات احادية ولكن سيتم ارسالها بشكل صحيح (كل سطر على حدة) نستخدم القاعدة

$i * d2 + j$

(for ال ~~الداخلية~~ (من ال ←
كل الأعمدة
كل الأسطر

* في هذه الحالة سيكون عدد الابناء هو عدد النتائج التي سنحصل عليها (ناتج ضرب المصفوفة (أي عدد الأسطر * عدد الاعمدة التي ستقوم لضربه).

عندي 3 مصفوفات
A : الأولى : ابعادها n, m1
B : الثانية : = m_2, p
C : مصفوفة الناتج : ابعادها n, p

• عند ملف ال txt عندي ابعاد المصفوفات + القيم ضمن كل مصفوفة
⇐ نفتح الملف و نقرأ ابعاد المصفوفة الاولى و نحملها Malloc
و نحجزها بحجم size of (int)

A = (int*) malloc (n * m1 * size of (int));

وبعد ثم نقوم بعملية read

وبنفس الطريقة نقرأ ابعاد B وقبل تشكيلها نتأكد من شرط ضرب المصفوفات ، في حال تحقق الشرط نكمل من انشاء المصفوفة

• المصفوفة C هي ; C = (int*)Malloc (n * p * size of (int))

m : عدد عناصر المصفوفة الناتجة وهي $m = m_1 * m_2$

pr : عدد المهام : n*p (عدد أسطر الأولى * عدد أعمدة الثانية).

* سيتم إنشاء مصفوفة نحفظ فيها الـ tasks id الخاصة بالأبناء
• نسميها child id (عرفناها لأنو عندي أكثر من child)

pvm_mytid() ← لتشغيل المهمة كـ task

```
n_child = pvm_spawn ("hello_other", (char**)0,
          0, "", child ID)
if ( n_child == pr)          --> n * p   [عدد المهام]
  for ( i=0; i<n ;i++){
    for( j=0 ; j<p; j++) {
  pvm_initsend (pvmDataDefault);
  pvm_pkint ( &m, 1, 1);
  pvm_pkint( A+i*m, m, 1);
  pvm_pkint( B+j, m, p);
  pvm_send( child ID[i*p+j], i*p+j);
  }
  }
}         for( k=0; K<pr; K++)
 bufid = pvm_recv( -1, -1);
 info = pvm_bufinfo( bufid, &bytes, &tag, &tid);
 pvm_upkint ( &S, 1, 1);
   C[tag] = S;
   }
```

ضمن تابع الاب (ضمن ملف الـ Hello) يقوم الاب بعملية ارسال لكل سطر موجود من المصفوفة الاولى والثانية الى احد الابناء وكذلك يقوم باستقبال النتائج من المهام الابناء.

* وكذلك هو الامر في hello_other.pvm يقوم كل ابن باستقبال السطر والعمود الخاص به ويوجد ناتج ضربهما ومن ثم يعيد ارسال الناتج الى الاب.

* سيتم تشغيل 4 ملفات Hello other.exe حيث سيتم ارسال لكل مهمة سطر وعمود مع الـ tag المناسب وبعد اتمام عملية الجداء من قبل كل مهمة سيرسل الناتج الى الاب مع الـ tag نفسه.

* pvm_recv(-1, master) يقوم باستقبال رسالة من المهمة الاب أيا كانت قيمة الـ tag الخاصة بالرسالة ...

• pvm_groups: يستفاد من فكرة المجموعات في تبسيط عمليات الإرسال بين عدة مهام، ففي كل group يتم إضافة مجموعة من المهام
في الـ groups يكون لكل مهمة 2 id ← task id وهو لا يتغير
← group id وهو الـ id الموجودة ضمنه عند اتمام المهمة
ومغادرتها المجموعة يصبح الـ gid=0
ويتم اسناده لأول مهمة جديدة تدخل المجموعة

• لكل group لدينا Root task وهي المهمة المسؤولة عن كافة العمليات

* Collective operation :

[1] pvm_join group(char * group name)

↓

عكي ان تكون المهمة في اكثر من group ⇐ تكون لها اكثر من groupid خاصة بها في كل مجموعة.

من هنا ايضا تم انشاء ال group في حال كان غير موجود مسبقا.

[2] pvm_lvgroup (char * group name);

[3] pvm_getinst (char * groupname, task ID)

يعيد هذا التابع ال gid الخاص بـ task معينة ضمن group معينة

[4] pvm_gettid(char group name, GID)

يعيد هذا التابع ال task id الخاص بمهمة ما الموجودة ضمن مجموعة، نعطي اسم ال group مع ال gid الخاص بها ضمنه

أما ال pvm_mytid يعيد ال task id الخاص بمهمة

التوابع الخاصة بـ pvm : هي كل شي مع ال Barrier الاثنين نفس الحزمة.

↓ تستجيب

ال MultiCast لا تستجيب

* pvm_mcast (child ID, count, tag)

one to many: تحدد عدد الـ tasks اللي بدي يها تستلم الرسالة مع مصفوفة الـ IDs لتحديد الأم اللي رح ارسلها

* pvm_Bcast (group name, tag)

مو ما بدي الـ count, والـ Child id لأني رح ابعت للكل ← بس اسم الـ group مع الـ tag.

* تابع one to all ← تحكم الـ task اللي ارسل الرسالة

دكويه ضمن المجموعة ← ستقبله الرسالة

ولكن دكويه خارج المجموعة فلا يصله الرسالة.

* pvm_gather (   )

تابع تجميعي + وظيفته تجميع البيانات الموزعة بين المهام، ويتم ارسال نتائج

Block of data تمثل النتيجة ويتم ارسالها الى الـ Root لعملية استلام لجميع الكتل الواردة.

• الارسال مثال nonroot / الاستقبال في الـ Root.

مثال: * pvm_gather (NULL, send, count, datatype, tag, group, root).

تجميع: * pvm_gather (gather data, send, count, datatype, tag, group, root).