

البرمجة التفرعية

المهندس عمار المصري

– الجلسة الثانية –

Parallel Virtual Machine (PVM)

- ▶ PVM من أقدم المكتبات التفرعية مبنية على `c\c++`، تدعم `c\c++` و Fortran، تشغل ضمن شبكات هجينة أي أنها محمولة على Linux و Windows.
- ▶ من حيث نظام Windows، تعتمد PVM على مكتبة wSock32، وهي مكتبة قديمة لم تعد موجودة في إصدارات Windows الحديثة، لذلك يحتاج تشغيل PVM إلى نظام Widows 7 أو Windows XP.
- ▶ كبيئة تطوير تربط PVM مع Visual Studio 6.0 الذي يوفر C Compiler الذي تحتاجه PVM ويتوافق مع مكتبة wSock32 .
- ▶ أما في Linux يكفي العمل على editor كون Linux تملك افتراضياً Standard C/C++ Compiler.

Parallel Task Registration

▶ أي برنامج عند تنفيذه يعمل كـ OS Task ويعطى Process ID.

▶ يتم تسجيل البرنامج كـ PVM Task عند استدعاء أحد تابعي التسجيل `pvm_mytid()` أو `pvm_parent()`، ليتم عندها الاتصال بالـ Local Deamon.

▶ عند الاتصال يقوم الـ PVMD بتسجيل البرنامج الحالي كـ PVM Task وإعطائه Task ID وحفظ معلومات الـ (Task ID – Parent ID) للمهمة، عندها يعمل البرنامج كـ OS Task & PVM Task.

Parallel Task Registration

- ▶ إنهاء الـ PVM Task يتم من خلال التابع `pvm_exit()`، عند استدعائه يلغي الـ PVMD تسجيل المهمة الحالية كـ PVM Task.
- ▶ يمكن استخدام توابع الـ PVM ضمن المجال المحدد من تسجيل الـ PVM Task بأحد تابعي التسجيل وحتى إنهاء تسجيلها بـ `pvm_exit()`.
- ▶ إن استدعاء توابع الـ PVM خارج هذا المجال يعطي Runtime Error لأن كل تابع يحتاج أن يكون مرتبطاً بمهمة تتصل مع الـ PVMD لتنفيذه.

Parent/Child Relation

- ▶ الـ PVMD يحفظ العلاقات بين المهام عند تسجيلها، أي من أجل كل مهمة يقوم بتسجيل الـ ID الخاص بها و ID المهمة المنشأة لها (Parent ID).
- ▶ المهمة الابن: هي كل مهمة تنشأ برمجياً عبر التابع `pvm_spawn()`، وتحصل على id المهمة المنشأة لها عبر التابع `pvm_parent()`.
- ▶ المهمة الأب: كل مهمة تنشأ مباشرة على الـ Terminal، ليس لها مهمة أب، فعند استدعاء تابع `pvm_parent()` لها يرد قيمة سالبة.
- ▶ الاتصال بين الآباء والأبناء مباشر ولا يحتاج لـ Local Deamon لأن المواصفات الشبكية معروفة لدى الطرفين لإنشاء اتصال مباشر، حيث أن:
- ▶ كل مهمة أب تملك ID's المهام الأبناء لها، ترد لها من خلال التابع `pvm_spawn()`.
- ▶ كل مهمة ابن تملك id المهمة الأب لها.
- ▶ إن المهام الأبناء تشغل في الذاكرة ولا ترتبط بـ Terminal (أي ليس لها وحدة خرج مباشرة).

PVM Communication (Send/Receive):

▶ تتم عملية الإرسال وفق ثلاث خطوات:

▶ تهيئة الرسالة.

▶ تحزيم الرسالة Packing.

▶ إرسال الرسالة.

▶ تتم عملية الاستقبال وفق ثلاث خطوات:

▶ استقبال الرسالة.

▶ قراءة معلومات اللصاقة (Buffer).

▶ فك تحزيم الرسالة Unpacking.

PVM Communication (Send/Receive):

- ▶ **Blocking Communication:**
 - ▶ Operations will wait until a communication has Completed in its local Process Before Continuing.
- ▶ **Non-Blocking Communication:**
 - ▶ It will Initiate a Communication Without waiting for that Communication to be Completed.

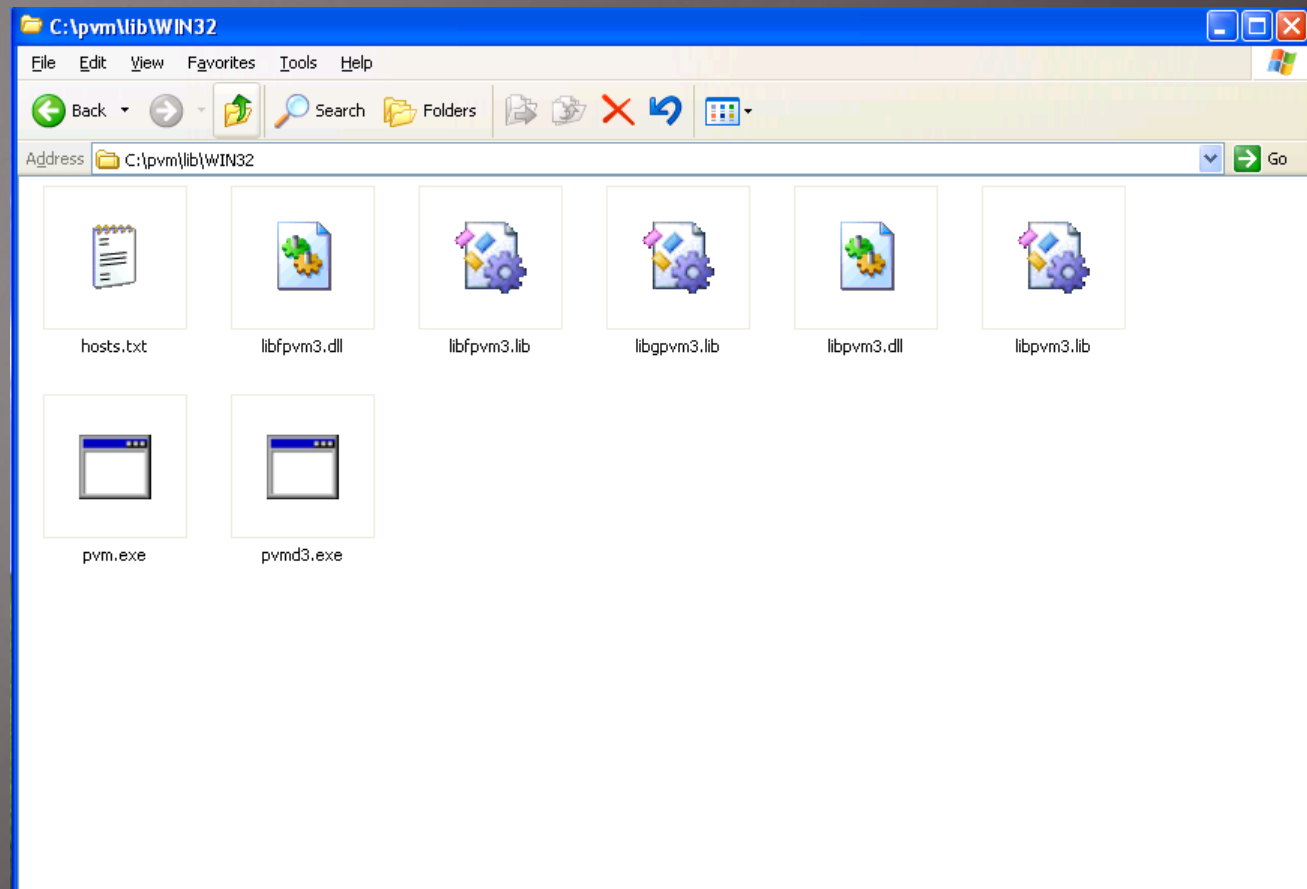
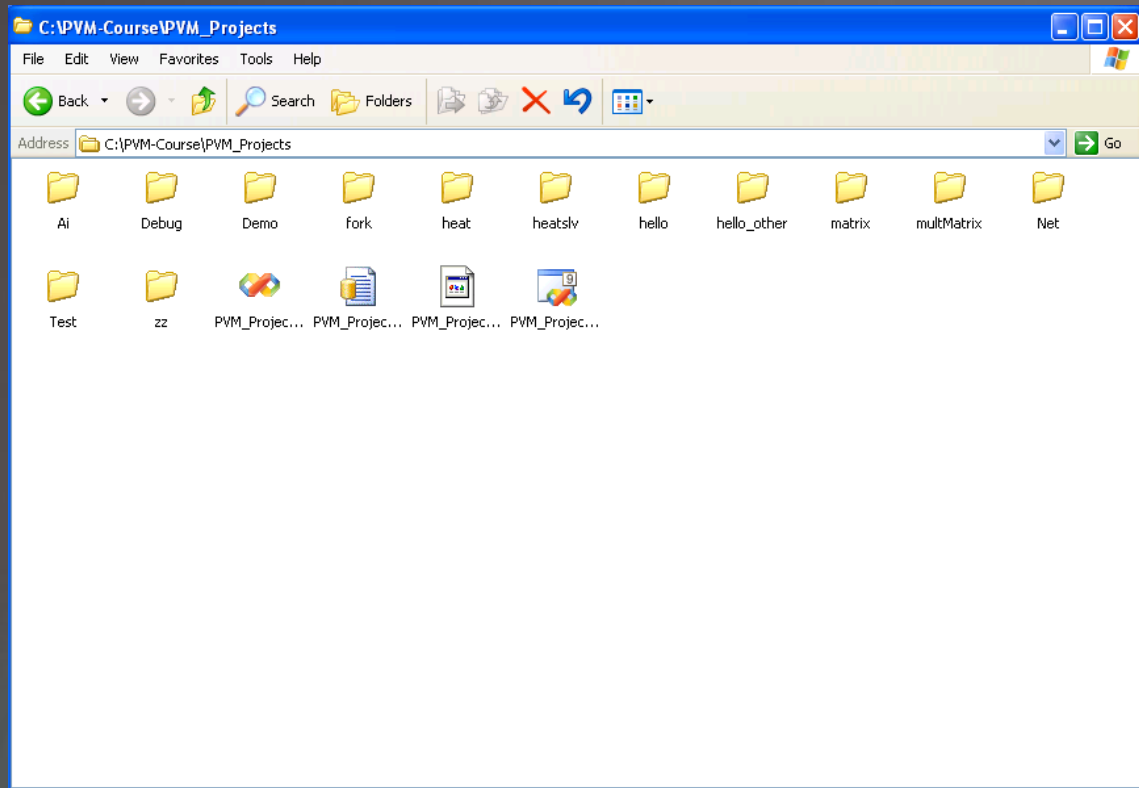
PVM Files

► لدينا ضمن القرص C:

► كافة المشاريع التي سنعمل عليها متواجدة ضمن المجلد
C:\PVM-Course\PVM_Projects

► المكتبات البرمجية الخاصة بالـ PVM موجودة ضمن المسار
C:\pvm\lib\WIN32

► الملفات التنفيذية للبرامج متواجدة ضمن المسار
C:\pvm\bin\WIN32

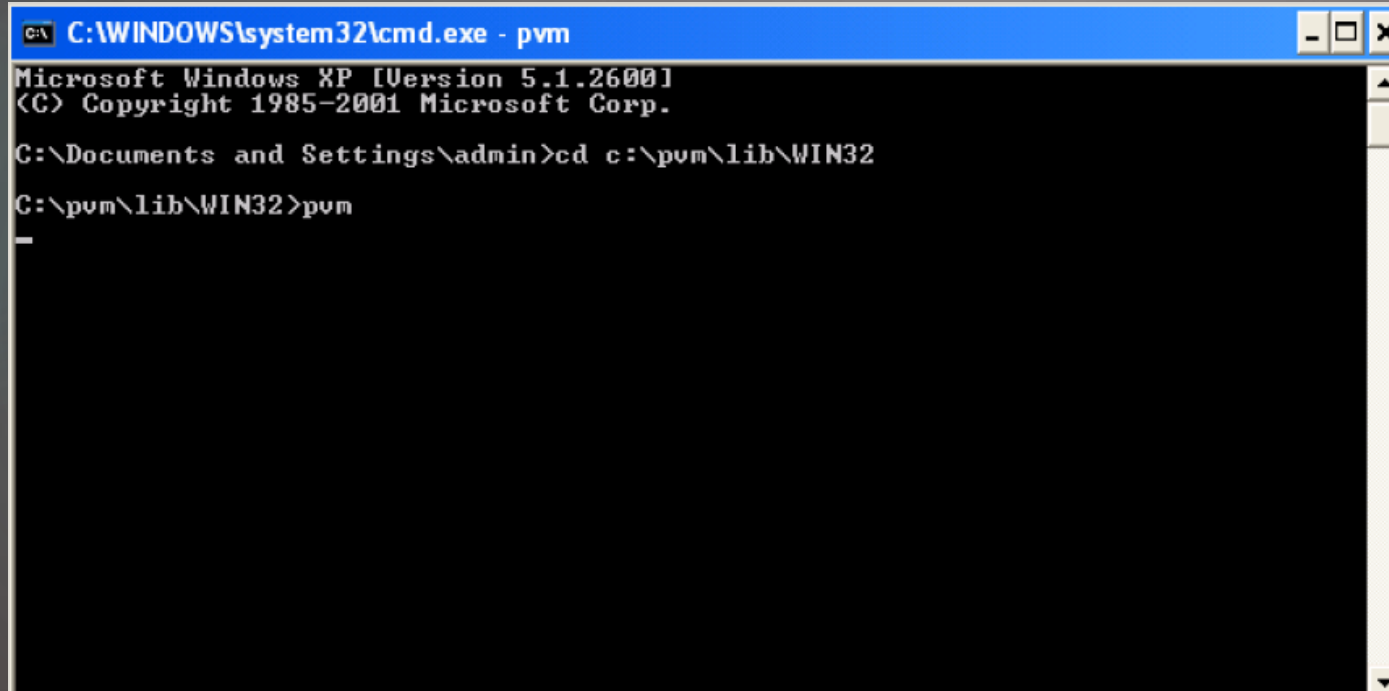


STEP ONE

▶ تشغيل الـ PVM لتأمين جاهزية بيئة العمل التفرعية:

▶ عن طريق الـ CMD، يتم التوجه إلى المسار الخاص بالـ PVM
(c:\pvm\lib\win32)

▶ نقوم بتنفيذ الأمر pvm.exe، ونواجه حالتين عند التشغيل:



```
C:\WINDOWS\system32\cmd.exe - pvm
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\admin>cd c:\pvm\lib\WIN32
C:\pvm\lib\WIN32>pvm
_
```

الحالة الأولى: ▶

▶ أن يكون قد تم اغلاق الـ PVM بشكل غير نظامي، وفي هذه الحالة لن يعمل وستظهر الرسالة `pvm already running`، ولحل هذه المشكلة نتوجه إلى مجلد `pvm_Temp` (المتواجد على سطح المكتب) ونقوم بحذف كافة الملفات المتواجدة بداخله، ثم نقوم بالتشغيل مرة أخرى عندها تعمل الـ PVM بشكل سليم.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

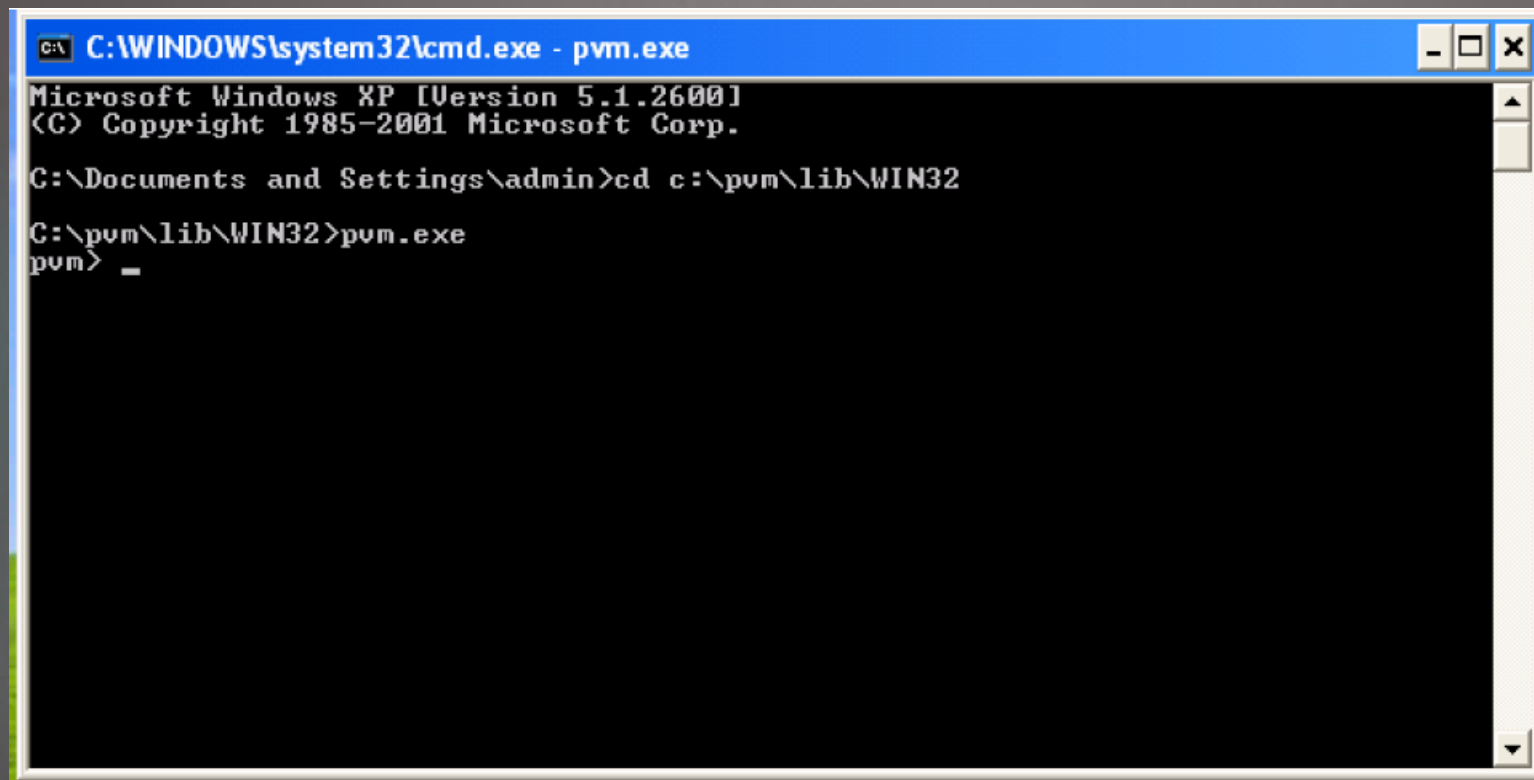
C:\Documents and Settings\admin>cd c:\pvm\lib\WIN32

C:\pvm\lib\WIN32>pvm
libpvm [pid2652] mksocs() connect: No error
libpvm [pid2652] socket address tried: 7f000001:0497
pvm already running.
libpvm [pid2652] mksocs() connect: Bad file descriptor
libpvm [pid2652] socket address tried: 7f000001:0497
libpvm [pid2652] mksocs() connect: Bad file descriptor
libpvm [pid2652] socket address tried: 7f000001:0497
libpvm [pid2652] mksocs() connect: Bad file descriptor
libpvm [pid2652] socket address tried: 7f000001:0497
libpvm [pid2652]: pvm_mytid(): Can't contact local daemon

C:\pvm\lib\WIN32>
```

الحالة الثانية: ▶

▶ أن يكون قد تم إغلاق الـ PVM بشكل سليم، عندها سيظهر لدينا مباشرةً على الشاشة `pvm>`، ولإغلاق الـ PVM بشكل سليم نستعمل التعليمة `.halt`.



```
C:\WINDOWS\system32\cmd.exe - pvm.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\admin>cd c:\pvm\lib\WIN32

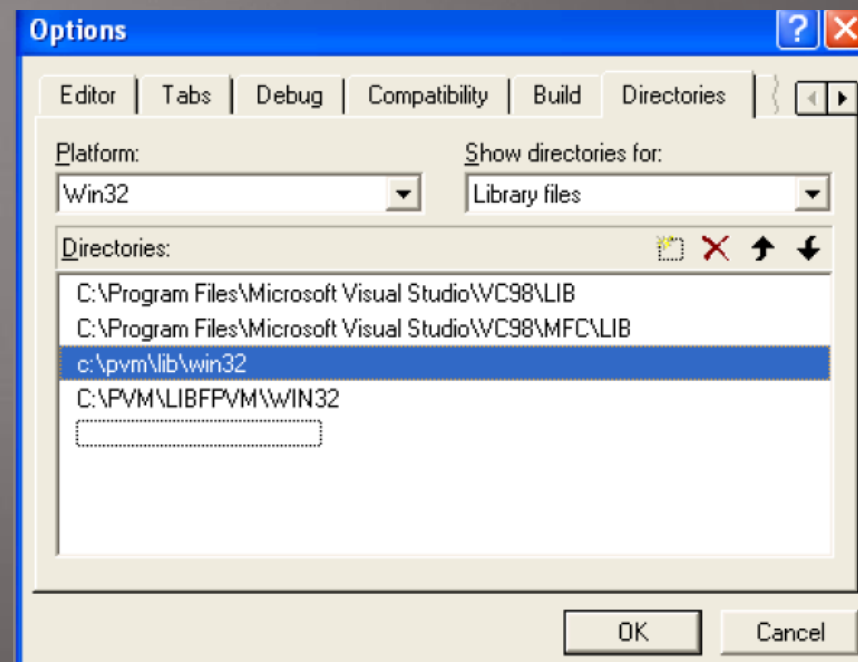
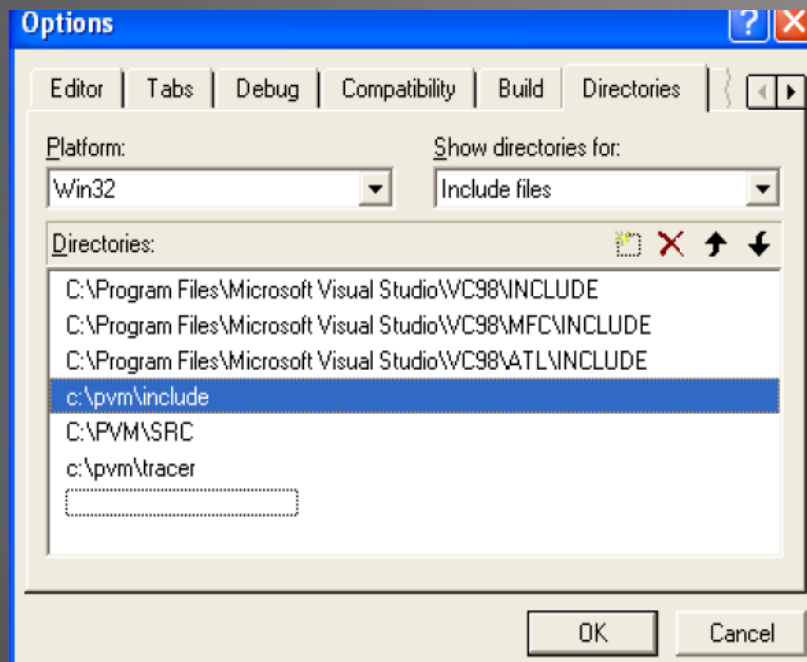
C:\pvm\lib\WIN32>pvm.exe
pvm> _
```

STEP TWO

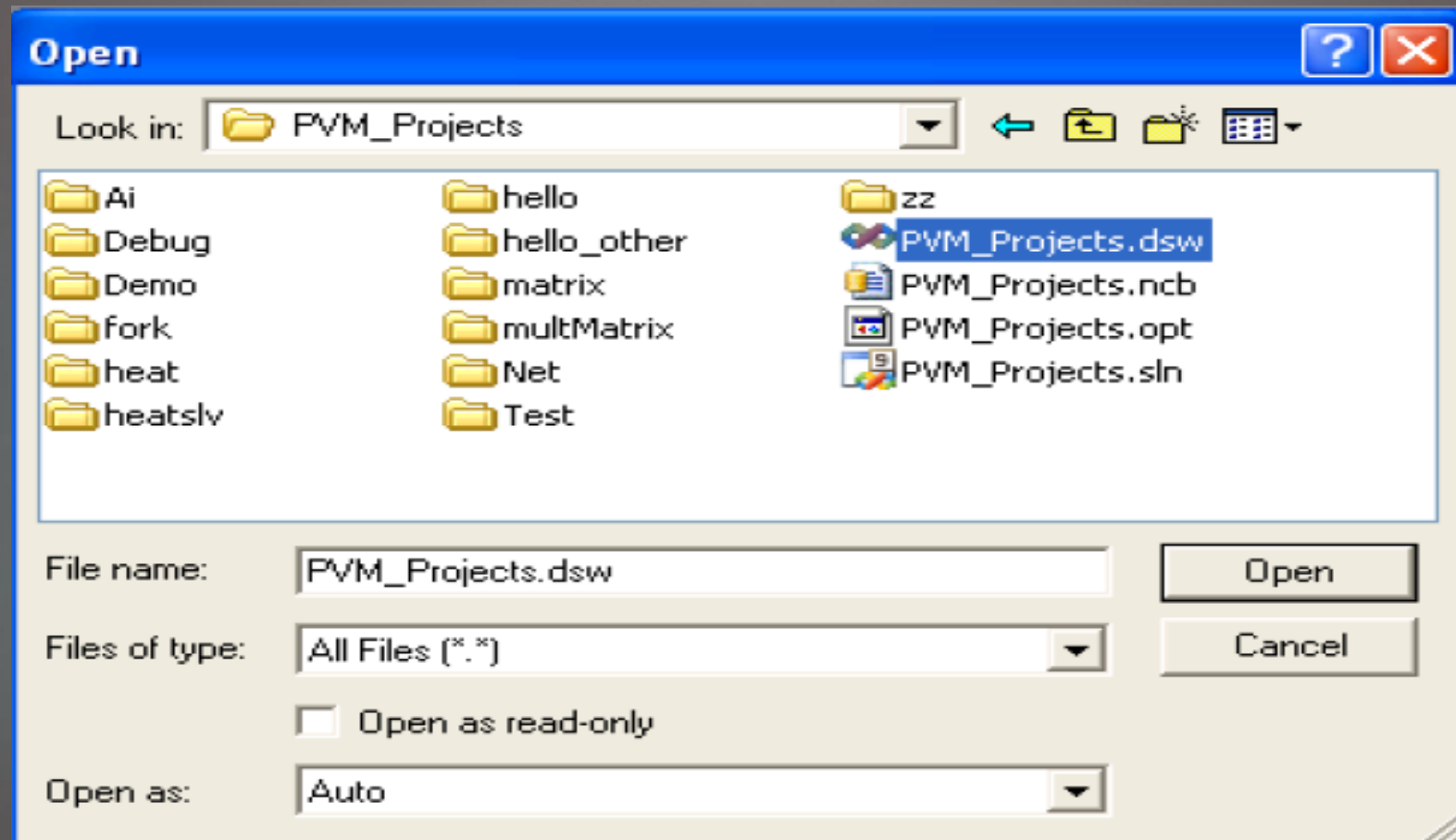
➤ ربط الـ PVM مع Microsoft Visual C++ 6.0:

➤ من خيار Tools ثم Options، عند فتح النافذة نختار Directories،
ثم نقوم بتحديد مسار c:\pvm\include من خيار include files.

➤ ونكرر نفس العملية لأجل الـ Library Files عبر تحديد المسار
c:\pvm\lib\win32



► للقيام بفتح المشاريع نعرض كافة الملفات ضمن المجلد `pvm_projects` ونختار الملف ذو اللاحقة `dsw` من أجل عرض كافة المشاريع ضمن الـ `workspace`.



The Code:

▶ لدينا برنامجان هما:

▶ **Hello:** وسيكون البرنامج الأب.

▶ **Hello_Other:** وسيمثل برنامج الابن.

Hello

```
1)  #include "pvm3.h"
2)  main()
    {
3)
4)      int cc, tid;
5)      char buf[100];
6)      printf("i'm t%x\n", pvm_mytid());
7)      cc = pvm_spawn("hello_other", (char**)0, 0, "", 1, &tid);
8)      if (cc == 1)
        {
9)          cc = pvm_recv(-1, -1);
10)         pvm_bufinfo(cc, (int*)0, (int*)0, &tid);
11)         pvm_upkstr(buf);
12)         printf("from t%x: %s\n", tid, buf);
        }
13)     else
14)         Printf("can't start hello_other\n");
15)     pvm_exit();
16)     exit(0); }
```


► في السطر السادس يتم تسجيل البرنامج كـ PVM Task عبر استدعاء التابع `pvm_mytid()` والذي يقوم بطباعة قيمة الـ Task ID للمهمة الحالية (المهمة الأب).

► في السطر السابع التابع `pvm_spawn()` هو المسؤول عن انشاء الأبناء، والبارامترات الخاصة بهذا التابع هي:

(1) الملف التنفيذي للابن المراد انشاء مهمة نسخة منه.

(2) البارامتران الثاني والثالث نحتفظ بقيمتها كما هي، حيث `0(int*)` يعبر عن القيمة `null`.

(3) البارامتر الخامس يعبر عن عدد المهام الأبناء التي نرغب بإنشائها.

(4) البارامتر السادس يعبر عن قيمة الـ Task ID للمهمة الابن، حيث ترد القيمة `By Reference`.

(5) `Cc`: القيمة المعادة من التابع `pvm_spawn()`، وتمثل العدد الحقيقي للمهام التي تم انشاؤها بنجاح، وتكون القيمة سالبة في حال عدم نجاح التابع في انشاء أي نسخة.



► في السطر الثامن يتم اختبار نجاح انشاء مهمة `hello_other` واحدة، حيث اذا تم انشاء الابن بنجاح سيقوم بنفس اللحظة بعمل `run` للابن `hello_other`.

► في السطر التاسع: تابع استلام الرسائل `pvm_recv(int task_id, int msg_tag):`

► `Task_id`: يحدد `id` مهمة معينة لاستلام الرسالة منها، القيمة 1- تمثل الاستلام من أي مهمة دون الاهتمام بالجهة المرسل.

► `Msg_tag`: يحدد `tag` معين للرسالة لاستقبالها، والقيمة 1- تعني قبول أي `tag`، أي يتم الاستقبال دون الاهتمام بالـ `tag` الخاص بالرسالة.

► التابع `pvm_recv` يبقى الـ `task` قيد الانتظار إلى أن يستلم الرسالة المطلوبة، فعند استلام اول رسالة موافقة للشروط ستتوقف عملية الاستلام وينتقل للتعليمة التالية.

► في السطر العاشر التابع `pvm_bufinfo`، وهو تابع قراءة معلومات اللصاقة، حيث بارامتراته هي:

(1) الأول هو دخل عملية الاستقبال، حتى يعلم لصاقة أي رسالة مرسلة ستتم قراءتها (بحال استقبال أكثر من رسالة).

(2) الثاني يعبر عن حجم الرسالة، والثالث يعبر عن `tag` الرسالة.

(3) البارامتر الرابع يحدد `id` المهمة التي أرسلت الرسالة.

► في السطر الحادي عشر التابع `pvm_upkstr()` وهو التابع المسؤول عن فك تحزيم الرسالة.

► في السطر الثاني عشر يتم طباعة قيمة الـ `id` الخاص بالابن المنشأ، وطباعة محتوى الرسالة التي تم استلامها والمخزن ضمن المتحول `buf`.

Hello_Other

```
1)  #include "pvm3.h"
2)  main() {
3)    int ptid;
4)    char buf[100];
5)    ptid = pvm_parent();
6)    strcpy( buf, "hello , world from ");
7)    gethostname( buf + strlen(buf) , 64 );
8)    pvm_initsend( PvmDataDefault );
9)    pvm_pkstr( buf );
10)   pvm_send(ptid, 1);
11)   pvm_exit();
12)   exit(0); }
```



► في السطر الخامس يتم تسجيل البرنامج كـ PVM Task عبر استدعاء التابع `pvm_parent()` والذي يقوم بإعادة قيمة الـ `id` المهمة الأب وتخزينها في المتحول `.ptid`.

► في السطر السادس التابع `strcpy()` هو تابع لنسخ سلسلة محارف، حيث يقوم بنسخ محتويات الـ `baramtr` الثاني وتخزينها ضمن المصفوفة `.buf`.

► في السطر السابع التابع `gethostname` يتم استخدامه لأنه يمكن أن نشغل الأب على جهاز والابن على جهاز آخر، ففي الـ `baramtr` الأول يقوم بتحديد مكان تواجد المؤشر، حيث يقوم بتخزين اسم الجهاز المضيف بطول أعظمي 64 محرف (تم تحديد الطول في الـ `baramtr` الثاني).



► تتم عملية الارسال وفق الخطوات الثلاث التالية:

► في السطر الثامن تتم تهيئة الارسال عبر التابع `pvm_itsend()`، حيث `PvmDataDefault` هو نمط معياري للإرسال يستخدم لعمل `encoding` للبيانات بطريقة تناسب تبادل الرسائل لتقرأ ضمن أي نظام تشغيل.

► نستخدم التابع `pvm_pkstr()` في السطر التاسع لتحزيم البيانات المراد إرسالها إلى الطرف الآخر (المصفوفة `buf`).

► في السطر العاشر لدينا تابع إرسال الرسالة `pvm_send(int task_id, int msg_tag)` حيث:

► `Task_id`: يحدد `id` المهمة المراد إرسال الرسالة لها.

► `Msg_tag`: يحدد `tag` مميز للرسالة.

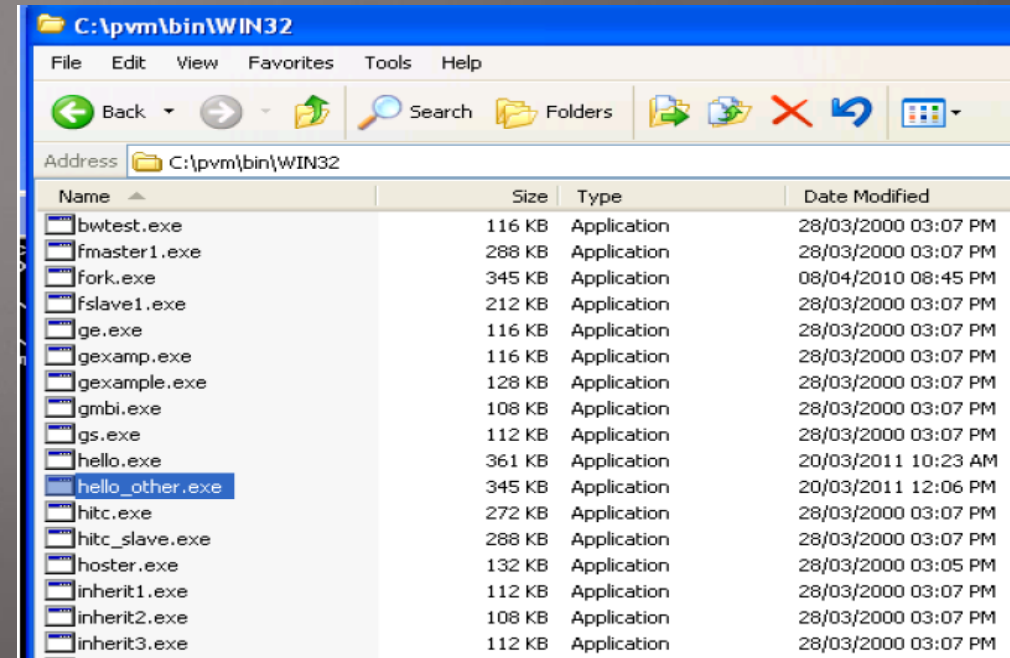
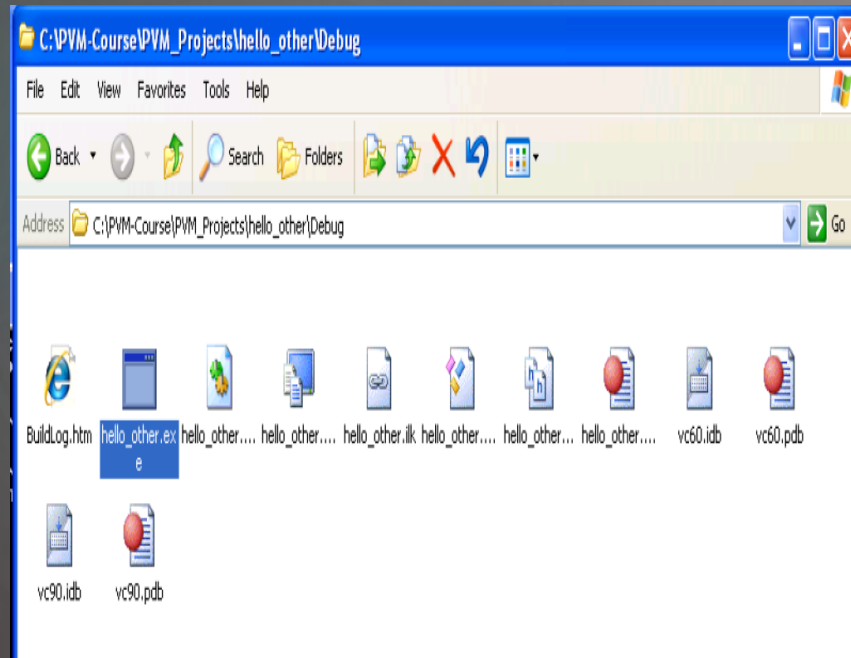
Execute The Code

▶ الخطوة الأولى:

▶ نقوم بعمل Build للابن hello_other وذلك عبر الضغط على ملف البرنامج واختيار Build.

▶ الخطوة الثانية:

▶ نقوم بالتوجه إلى المسار c:\PVM-Course\PVM Projects\hello_other\Debug ونقوم بنسخ الملف التنفيذي للابن hello_other.exe ونضعه ضمن المسار التالي c:\pvm\bin\win32.

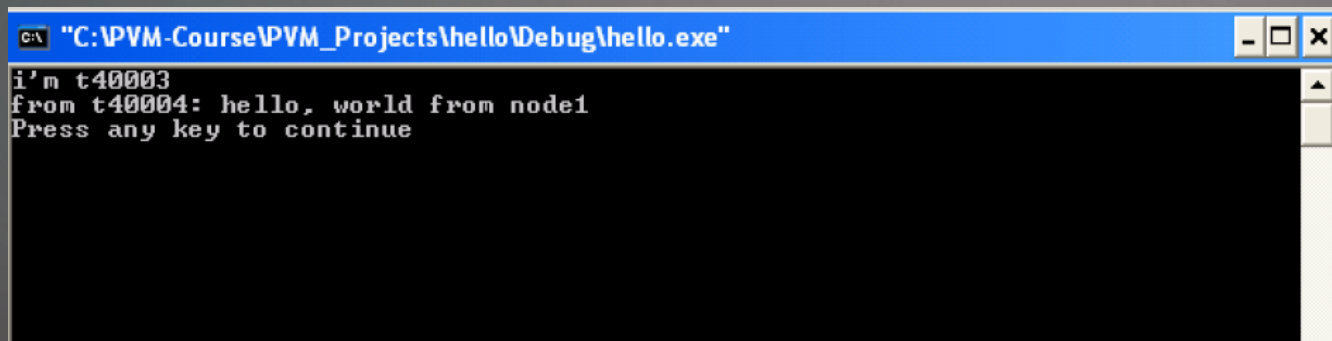


الهدف من الخطوة السابقة: ▶

▶ عندما يتم عمل spawn بغرض انشاء الـ hello_other، عندها يتم البحث عن الملف التنفيذي للابن ضمن مسار متفق عليه عوضاً عن البحث عنه ضمن كامل الجهاز.

الخطوة الثالثة: ▶

▶ نقوم بتنفيذ المشروع الأب hello عن طريق الضغط عليه بالزر اليميني وتعيينه كالمشروع الرئيسي (Set as Active Project)، ثم نقوم بعمل Execute للمشروع (Ctrl+F5).



```
C:\PVM-Course\PVM_Projects\hello\Debug\hello.exe
i'm t40003
from t40004: hello, world from node1
Press any key to continue
```


Notes:

- ▶ عند استخدام التابع `pvm_recv`:
تحفظ الرسالة الواردة ضمن `System Buffer` مخصص (Receive Buffer)، وتكون القيمة `cc` المعادة من التابع هي عنوان ال `buffer` في الذاكرة.
- ▶ المهام الأبناء تُشغل في الذاكرة ولا ترتبط ب `terminal`، أي ليس لها واجهة خرج مباشرة، ولطباعة خرج مهمة ابن يتم إرسال هذا الخرج للمهمة الأب التي تملك `terminal` لإظهار الخرج ضمنه.
- ▶ عند إنهاء تسجيل مهمة ما لا يعاد استخدام ال `id` الذي كانت تملكه في تسجيل مهمة جديدة، حيث يستمر ال `PVMD` في إعطاء قيم جديدة للمهام التي يسجلها حتى نهاية مجال الأرقام المتاح، وذلك لتجنب حدوث أخطاء تداخل بين مهام مرتبطة بالمهمة المنتهية والمهمة الجديدة.

TO BE
CONTINUED