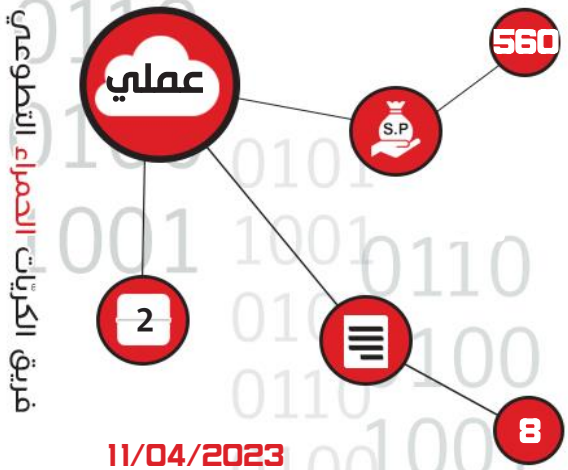


Parallel Virtual Machine (PVM)

م . عمار المصري

لمحة عن ال PVM :

- PVM من أقدم المكتبات التفرعية مبنية على C/C++ و Fortran ،و تدعم C/C++ و Fortran ،تشغل ضمن شبكات هجينة أي أنها محمولة على Linux و windows.
- من حيث نظام windows تعتمد على مكتبة wSock32 وهي مكتبة قديمة لم تعد موجودة في إصدارات Windows الحديثة لذلك يحتاج تشغيل PVM إلى نظام Windows 7 أو Windows XP.
- كبيئة تطوير ترتبط 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 .

PVM Communication (Send / Receive)

يتم التواصل بين ال tasks عن طريق عمليات الإرسال والاستلام.

عملية الإرسال:

1. تهيئة الرسالة: باستخدام أنماط معيارية معينة لتحقق عملية الارسال مثلاً التأكد من نمط البيانات المرسلة و قابلية فهمها لدى المستقبل المرسلة إليه.
2. تحزيم الرسالة: تحديد مواصفات الرسالة ، تحديد من المستقبل (اللصاقة).
3. إرسال الرسالة.

عملية الاستقبال:

1. استقبال الرسالة.
2. قراءة معلومات اللصاقة (Buffer) مثل معرفة حجمها , tag , اذا كان المستقبل المراد.
3. فك تحزيم الرسالة unpacking.

Blocking & Non Blocking Communication

Blocking :

عند الوصول إلى تعليمة الإرسال والاستقبال في الكود البرمجي يدخل البرنامج في حالة Block حتى تتم عملية الاستقبال أو الإرسال ومن ثم يتابع التنفيذ.

Non-blocking :

عند الوصول إلى تعليمة الإرسال يتم إرسال الرسالة ومتابعة تنفيذ الكود بغض النظر إذا تم استلامها أو لا ، وفي عملية الاستقبال عند الوصول إلى تعليمة الاستقبال يتم متابعة التنفيذ بغض النظر إذا تم أو لا ولكن هنا يجب الانتباه ألا تتم عملية قراءة معلومات اللصاقة طالما لم يتم استلام أي رسالة.

Parallel Task Registration

- أي برنامج يعمل ضمن النظام يعتبر OS Task وله Process ID.
- عند استدعاء أحد تابعي التسجيل (pvm_mytid() / pvm_parent() يتم الاتصال بـ local Deamon واستدعاء PVMD الذي يقوم بتسجيل البرنامج كـ PVM Task وإعطائه Task ID وحفظ معلومات ال (Task ID – Parent ID) للمهمة وبالتالي يعمل البرنامج كـ OS Task + PVM Task.
- عند استدعاء pvm_exit() يتم إنهاء المهم الحالية.
- ملاحظة: يتم استخراج توابع ال pvm ضمن المجال المحدد من تسجيل ال PVM Task بأحد تابعي التسجيل وحتى إنهاء التسجيلة ، وأي استدعاء خارج هذا الـ Scoop يتم إعطاء خطأ RunTime Error ، لأن كل تابع يحتاج أن يكون مرتبطاً بمهمة تتصل مع ال PVMD لتنفيذه.

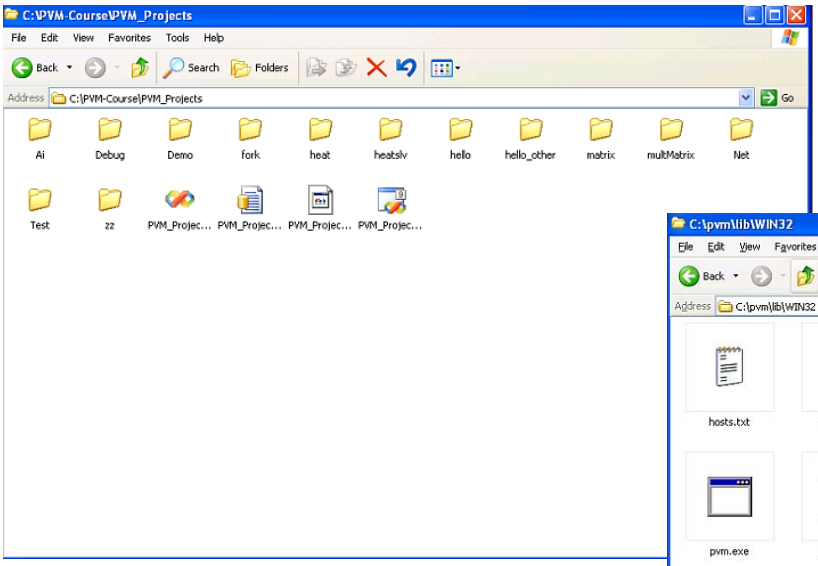
- Pvm_mytid() : يقوم بإرجاع id المهمة الحالية.
- Pvm_parent() : يقوم بإرجاع id الأب للمهمة الحالية.

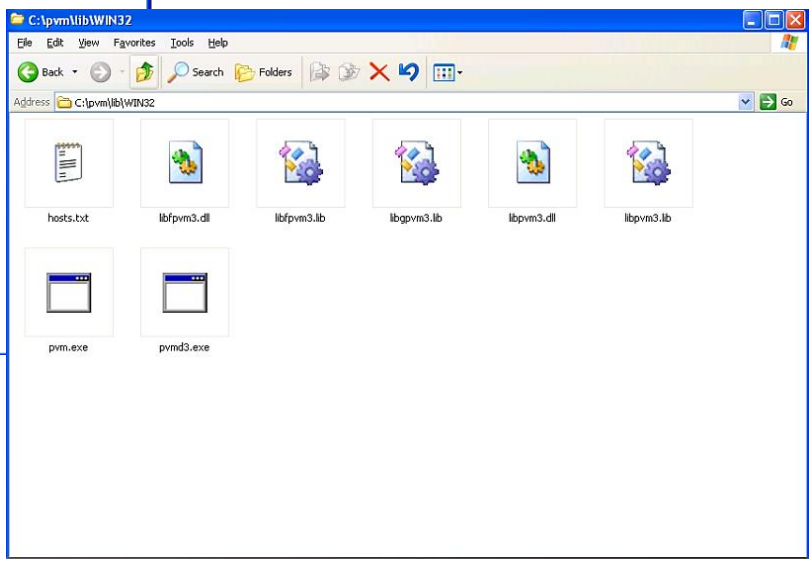
- في حال تم استدعاء pvm_parent() في مهمة الأب يقوم بإرجاع قيمة سالبة.
- لمهمة الأب يتم إنشاءها ضمن ال Terminal مباشرة.
- الاتصال بين الأب والابن على نفس الprocess ليس بحاجة إلى local Deamon لأن المواصفات الشبكية معروفة لدى الطرفين.
- إن المهام الأبناء تشغل في الذاكرة ولا ترتبط بالTerminal (أي ليس لها وحدة خرج) وبالتالي تقوم بإرسال النتائج إلى المهمة الأب للوصول إلى الناتج النهائي حيث أن :
- كل مهمة أب تملك ID's المهام الأبناء، ترد لها من خلال التابع pvm_spawn() .
- كل مهمة ابن تملك ID المهمة الأب لها.

التطبيق العملي

لدينا على القرص C :

1. كافة المشاريع التي سنعمل عليها متواجدة ضمن المجلد: C:\PVM-Course\PVM_Projects
2. المكتبات البرمجية الخاصة بال PVM موجودة ضمن المسار: C:\PVM-Course\PVM_Projects
3. الملفات التنفيذية للبرامج متواجدة ضمن المسار: C:\pvm\bin\WIN32

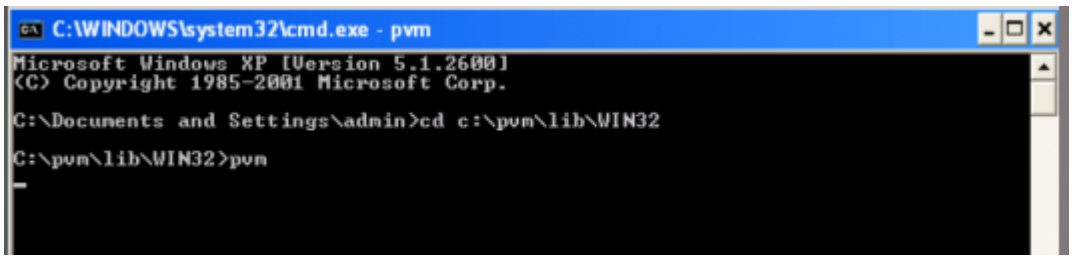




Step One (1)

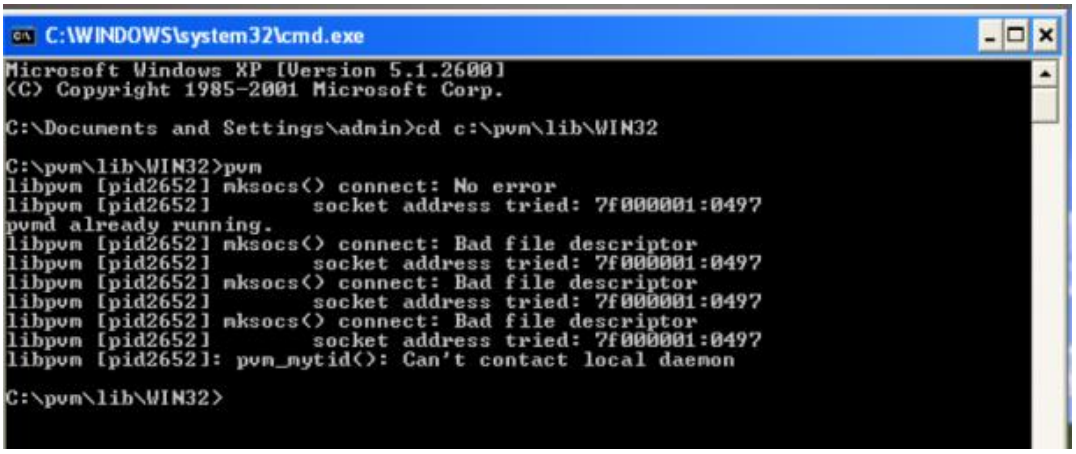
تشغيل ال PVM لتأمين جاهزية بيئة العمل التفرعية:

- عن طريف ال CMD يتم التوجه إلى المسار الخاص بال PVM : (C:\pvm\lib\win32)
- نقوم بتنفيذ الأمر pvm.exe

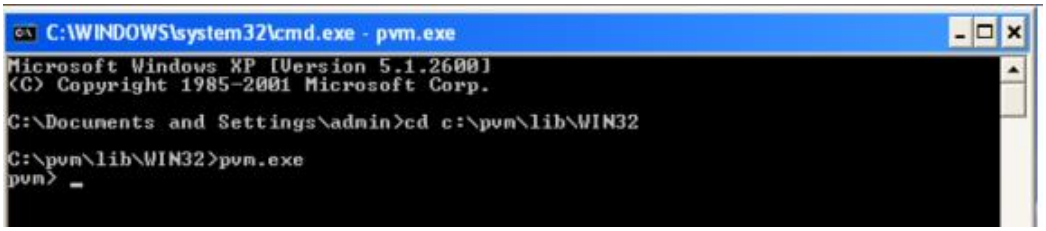


ونواجه حالتين عند التشغيل:

1. الحالة الأولى: أن يكون قد تم إغلاق ال 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 الpvm بشكل سليم.



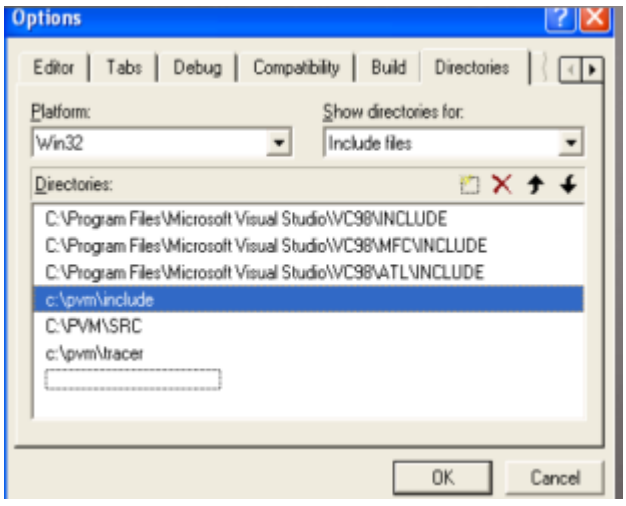
2. الحالة الثانية: أن يكون قد تم إغلاق ال pvm بشكل سليم عندها سيظهر لدينا مباشرة على الشاشة pvm> ولإغلاق ال PVM بشكل سليم نستعمل تعليمة halt.

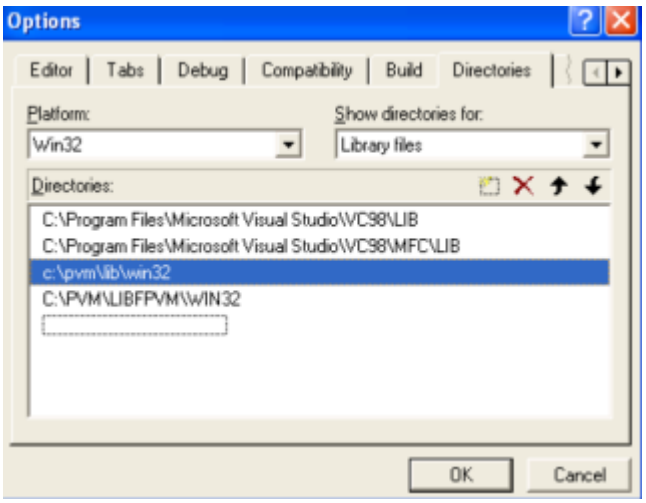


Step Two (2)

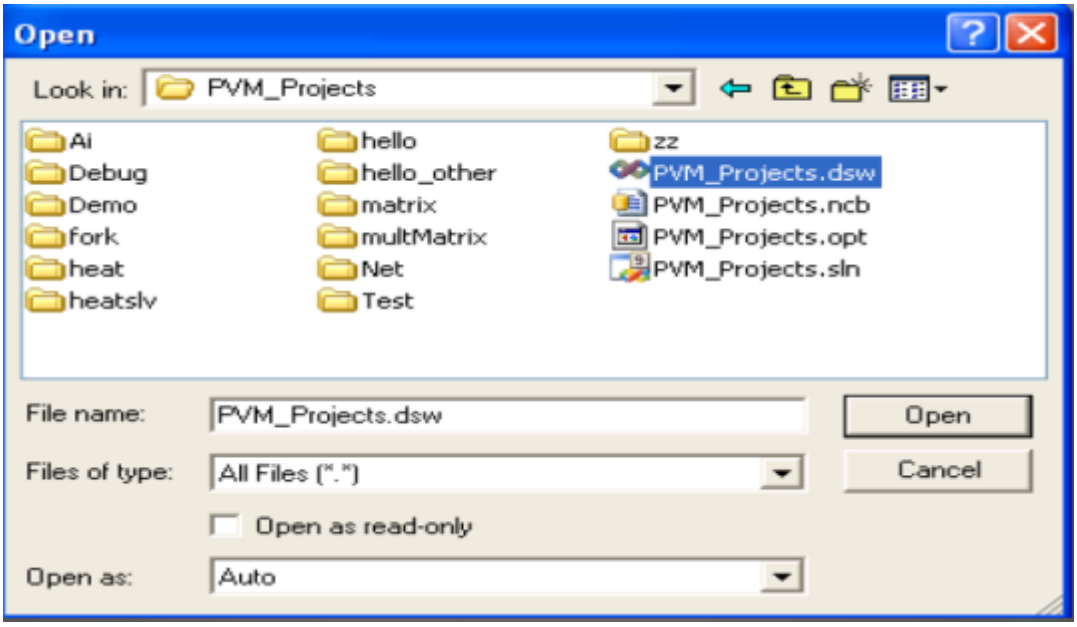
ربط ال PVM مع Microsoft Visual C++ 6.0

- 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 كافة المشاريع ضمن ال workspace.



The Code

لدينا برنامجين هما:

1) Hello : سيكون البرنامج الأب.
2) Hello_Other : سيمثل برنامج الابن.

Hello

```
1)  #include "pvm3.h"
2)  main()
    {
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 Task ID للمهمة الحالية (المهمة الأب).

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

1- الملف التنفيذي للابن المراد انشاء مهمة نسخة منه.
2- البارامتران الثاني والثالث نحتفظ بقيمتهما كما هي، حيث 0(int*)يعبر عن القيمة null.
3- البارامتر الرابع يعبر عن عدد المهام الأبناء التي نرغب بإنشائها.
4- البارامتر الخامس يعبر عن قيمة الـ Task ID للمهمة الابن، حيث ترد القيمة By Reference.

Cc:القيمة المُعادة من التابع ()pvm_spawn ، وتمثل العدد الحقيقي للمهام التي تم إنشاؤها بنجاح، وتكون القيمة سالبة في حال عدم نجاح التابع في إنشاء أي نسخة.

في السطر الثامن: يتم اختبار نجاح إنشاء مهمة hello_other واحدة، حيث إذا تم إنشاء الابن بنجاح سيقوم بنفس اللحظة بعمل run للابن hello_other.

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

1- Task_id: يحدد id مهمة معينة لاستلام الرسالة منها، القيمة 1- تمثل الاستلام من أي مهمة دون الاهتمام بالجهة المرسلة.
2- Msg_tag: يحدد tag معين للرسالة لاستقبالها، والقيمة 1- تعني قبول أي tag ، أي يتم الاستقبال دون الاهتمام بالـ tag الخاص بالرسالة.
3- التابع pvm_recv : يبقي الـ task قيد الانتظار إلى أن يستلم الرسالة المطلوبة، فعند استلام أول رسالة موافقة للشروط ستتوقف عملية الاستلام وينتقل للتعليمة التالية.

في السطر العاشر: التابع pvm_bufinfo ، وهو تابع قراءة معلومات اللصاقة، حيث بارامتراته هي:
البارامتر الأول: هو دخل عملية الاستقبال، حتى يعلم لصاقة أي رسالة مرسلة ستتم قراءتها (بحال استقبال أكثر من رسالة).

البارامتر الثاني: يعبر عن حجم الرسالة، والبارامتر الثالث يعبر عن tag الرسالة.
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 باللابن المنشأ، وطباعة محتوى الرسالة التي استلامها المخزن ضمن المتحول buf .

```
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); }
```

Hello_Other

في السطر الخامس: يتم تسجيل البرنامج كـ PVM Task عبر استدعاء التابع ()pvm_parent و الذي 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 البارامتر الثاني إلى الأول.

في السطر السابع: التابع gethostname يتم استخدامه لأنه يمكن أن نشغّل الأب على جهاز والابن على جهاز آخر ،ففي البارامتر الأول يقوم بتحديد مكان تواجد المؤشر، حيث سيقوم بتخزين اسم الجهاز المضيف بعد كلمة from وبطول أعظمي 64 محرف (تم تحديد الطول في البارامتر الثاني).

تتم عملية الإرسال وفق الخطوات الثلاث التالية:

- تتم تهيئة الإرسال عبر التابع ()pvm_initsend حيث PvmDataDefault هو نمط معياري للإرسال يستخدم لعمل encoding للبيانات بطريقة تناسب تبادل الرسائل لتقرأ ضمن أي نظام تشغيل. (في السطر الثامن)
- نستخدم التابع ()pvm_pkstr لتحزيم البيانات المُراد إرسالها إلى الطرف الآخر (المصفوفة buf). (في السطر التاسع)
- لدينا تابع إرسال الرسالة (int task_id,int msg_tag)pvm_send حيث:

1) Msg_tag:يحدد tag مميز للرسالة.
2) Task id : يحدد id المهمة المراد إرسال الرسالة لها. (في السطر العاشر)

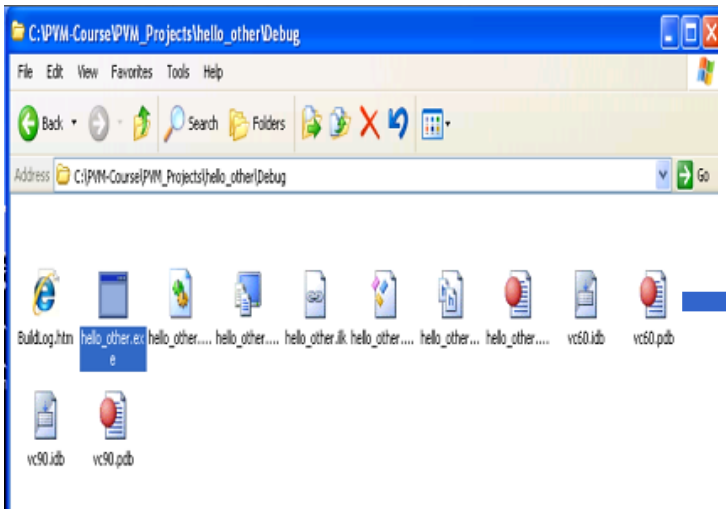
Execute The Code

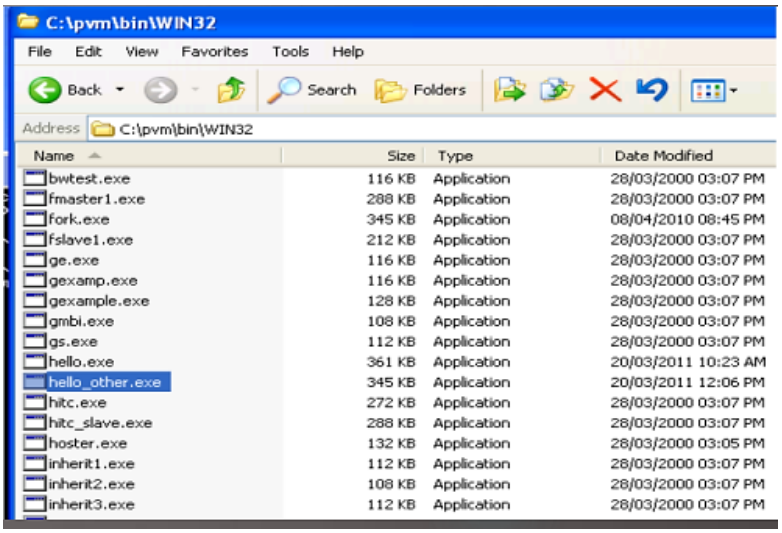
1. الخطوة الأولى:

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

2. الخطوة الثانية:

نقوم بالتوجه إلى المسار : C:\PVM-Courses\PVM_Projects\hello_other\Debug **ونقوم بنسخ الملف التنفيذي للابن** hello_other.exe **ونضعه ضمن المسار التالي:** C:\pvm\bin\win32



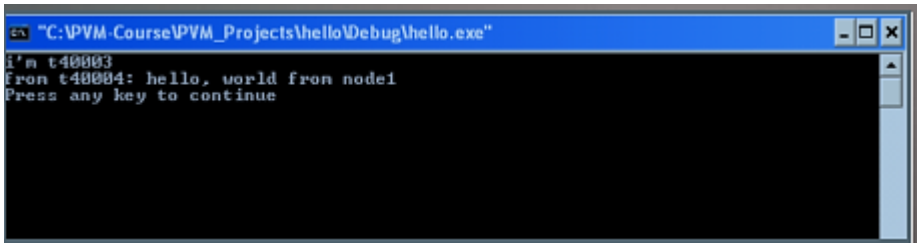


الهدف من الخطوة السابقة:

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

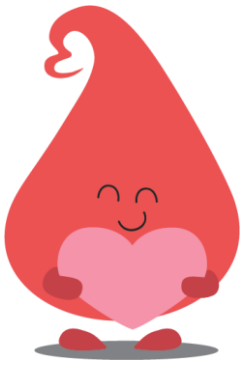
3. الخطوة الثالثة:

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) .



ملاحظات:

- من أجل كل مهمة ترسل أو تستقبل، هناك فقط buffer send active واحد و buffer receive active واحد في نفس اللحظة، حيث لا تتم عملية الإرسال بدون تهيئة الـ Send Buffer بـ init_send، وإلا يظهر خطأ PVM no active send buffer .
- 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 تُشغّل 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 الـ id الذي كانت تملكه في تسجيل مهمة جديدة، حيث يستمر الـ 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.



انتهت المحاضرة...