البرمجة التفرعية

المهندس عمار المصري

– الجلسة الرابعة –

- في المحاضرات السابقة كنا قد كتبنا برنامجين منفصلين واحد للأب $(hello)$ وواحد للابن $(hello\_other)$، كل منهما يقوم بعمله بشكل مستقل عن الآخر، أما في هذه المحاضرة فسنرى أنه بالإمكان ان نكتب الأب والابن في برنامج واحد ونتحكم متى سيكون أب ومتى سيكون ابن، وبالتالي تكون هيكلية البرنامج **if-else** والمتحولات المشتركة.
- أي ان الجزء البرمجي الخاص بالأب سيكون ضمن if، والجزء البرمجي الخاص بالابن سيكون ضمن else، والمتحولات التي نجدها بالطرفين تصبح مشتركة.
- فإذا أردنا تنفيذ $other\ hello$ & $hello$ في نفس البرنامج، فسنأخذ الأب وهو $hello$ ونضعه في شرط if، ونضع
- الابن $other\ hello$ في else.

```
Int sum
Float mm
If (     )
  {
    Father code
  }
Else
  {
     Sons code
  }
```

ملاحظات

- أي تابع من توابع الـ PVM عندا يقوم برد قيمة أصغر من الصفر (سالبة)، فهذا يدل على أن التابع فشل وهناك مشكلة في تنفيذ العملية.

- في البرنامج لن يتم فحص الأبناء التي تم انشاؤها.

FORK.C

```
int main(int argc, char* argv[])
{

    /* number of tasks to spawn, use 3 as the default */
    int ntask = 3;
    /* return code from pvm calls */
    int info;
    /* my task id */
    int mytid;
    /* my parents task id */
    int myparent;
    /* children task id array */
    int child[MAXNCHILD];
    int i, mydata, buf, len, tag, tid;
```

- بداية نقوم بتعريف مجموعة من المتحولات العامة أبرزها:
- ntask=3: ويعبر عن عدد المهام التي سنقوم بإنشائها.
- Info: متحول سنسند إليه عدد المهمات التي تم انشاؤها.
- Child: لتخزين الأبناء.

FORK.C

```
/* find out my task id number */
mytid = pvm_mytid();

/* check for error */
if (mytid < 0) {
    /* print out the error */
    pvm_perror(argv[0]);
    /* exit the program */
    return -1;
}
/* find my parent's task id number */
myparent = pvm_parent();

/* exit if there is some error other than PvmNoParent */
if ((myparent < 0) && (myparent != PvmNoParent)&&
                        (myparent != PvmParentNotSet))
{
        pvm_perror(argv[0]);
        pvm_exit ();
        return -1;
}
/* if i don't have a parent then i am the parent */
if (myparent == PvmNoParent || myparent == PvmParentNotSet)
{
    /*  find out how many tasks to spawn */
    if (argc == 2) ntask = atoi(argv[1]) ;

    /* make sure ntask is legal */
    if ((ntask < 1) || (ntask > MAXNCHILD))
    {
        pvm_exit();
        return 0;
    }
```

- ضمن الشروط السابقة:

- إن التابع **mytid = pvm_mytid()** هو تابع يرد قيمة الـ id للمهمة الحالية (سواءً كانت أب أو ابن)، فإن كانت القيمة الصادرة عن هذا التابع mytid<0، فهذا يدل على وجود خطأ في تسجيل الـ PVM Task وعندها يتم طباعة رسالة خطأ ومغادرة البرنامج.

- إن تابع **myparent = pvm_parent()** هو تابع يرد قيمة الـ id للأب الخاص بالمهمة الحالية، فإن كانت القيمة الصادرة عن هذا التابع myparent<0 فهذا يعني أن العملية فشلت ولا يوجد أب للمهمة الحالية (أي أنها المهمة الحالية تمثل الأب).

- لدينا المتحولات (**PvmParentNotSet**, **PvmNoParent**)، وهي متحولات معرفة مسبقاً ضمن الـ PVM، وتشير إلى وجود أب للمهمة الحالية أم لا، والقيمة الخاصة بهذه المتحولات هي id الأب.

- **اذا كانت قيمة المتحول myparent مساوية للقيمة PvmParentNotSet أو القيمة PvmNoParent، فهذا يعني أن المهمة الحالية لا تملك أب، وبالتالي فهي تمثل الأب، عندها ننتقل إلى تنفيذ الجزء البرمجي الخاص بالأب، وإلا فننتقل إلى تنفيذ الجزء الخاص بالابن.**

FATHER'S CODE

```
/* spawn the child tasks */
info = pvm_spawn("fork", (char**)0, PvmTaskDefault,
                        (char*)0,ntask, child);
/* print out the task ids */
for (i = 0; i < ntask; i++)
    if (child[i] < 0) /* print the error code in decimal*/
        printf(" %d", child[i]);
    else /* print the task id in hex */
        printf("t%x\t", child[i]);
putchar('\n');

/* make sure spawn succeeded */
if (info == 0) { pvm_exit(); return -1; }

/* only expect responses from those spawned correctly */
ntask = info;
```

- يقوم التابع **pvm_spawn** بإنشاء الأبناء، حيث اسم البرنامج الذي سينفذه هو ذات البرنامج fork، عدد المهام التي سيتم انشاؤها تساوي قيمة المتحول ntask أي سيتم انشاء ثلاث أبناء، child هي مصفوفة المؤشرات التي سيتم بها تخزين الـ id's للأبناء الذين تم انشاؤهم بنجاح.

- لدينا حلقة for نمر بها على عدد الأبناء، حيث عندما child[i]<0 فالابن لم يتم إنشاؤه، وسيطبع id بقيمة سالبة.

- اذا كانت قيمة المتحول info=0، وهو المتحول الذي يدل على عدد الأبناء التي تم انشاؤها بنجاح، فسيتم الخروج من البرنامج.

```
for (i = 0; i < ntask; i++)
{
    /* recv a message from any child process */
    buf = pvm_recv(-1, JOINTAG);
    if (buf < 0)
        pvm_perror("calling recv");
    info = pvm_bufinfo(buf, &len, &tag, &tid);
    if (info < 0)
        pvm_perror("calling pvm_bufinfo");
    info = pvm_upkint(&mydata, 1, 1);
    if (info < 0)
        pvm_perror("calling pvm_upkint");
    if (mydata != tid)
        printf("This should not happen!\n");
    printf("Length %d, Tag %d, Tid t%x\n", len, tag, tid);
}
pvm_exit();
return 0;
```

- سيتم استخدام حلقة for لاستقبال الرسائل من كافة الأبناء التي تم انشاؤها:

- التابع **pvm_recv()** يقوم باستقبال الرسالة من الابن، خرج التابع هي القيمة buf، فإذا كانت هذه القيمة سالبة (buf<0) عندها فإن عملية الاستقبال قد فشلت ويتم طباعة رسالة خطأ للمستخدم.

- التابع **pvm_bufinfo()** يقوم بقراءة معلومات الـ buffer، خرج التابع هو المتحول info، فإذا كانت هذه القيمة سالبة (info<0) عندها فإن عملية قراءة المحتويات قد فشلت ويتم طباعة رسالة خطأ للمستخدم.

- التابع **pvm_upkint()** يقوم بفك تحزيم الرسالة التي تم استقبالها، خرج التابع هو المتحول info، فإذا كانت هذه القيمة سالبة (info<0) عندها فإن عملية فك التحزيم قد فشلت ويتم طباعة رسالة خطأ للمستخدم.

- أخيراً يتم طباعة معلومات الرسالة من أجل كل ابن.

SON'S CODE

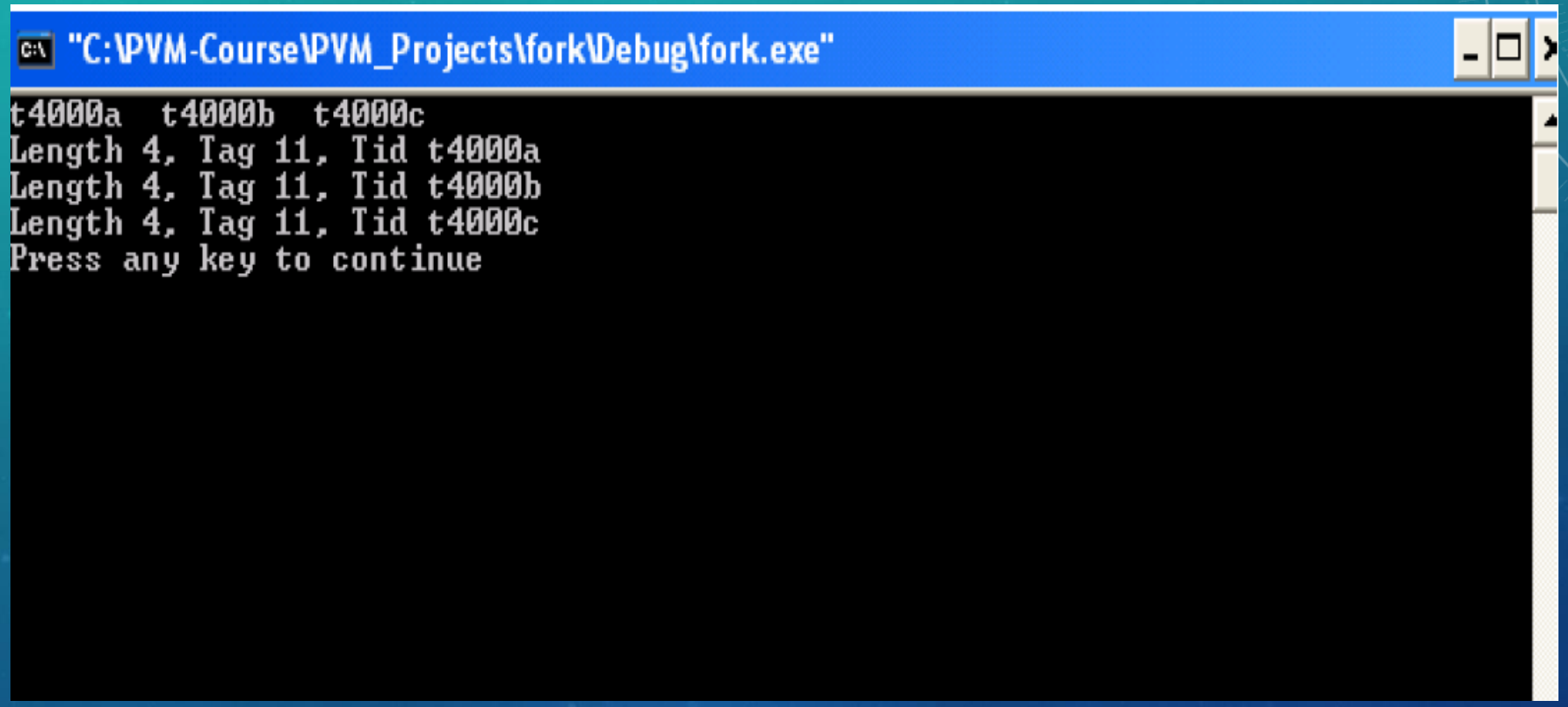
```
else
{
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&mytid,1,1);
    pvm_send(myparent, JOINTAG);
    pvm_exit();
    return 0;
}
```

- يقوم الابن بعملية الارسال:

- يتم بدايةً تهيئة عملية الارسال باستخدام التابع **pvm_initsend()**.

- التابع **pvm_pkint()** يقوم بتحزيم الرسالة المراد ارسالها إلى الأب (قيمة الـ id الخاص بالابن).

- يتم ارسال الرسالة باستخدام التابع **pvm_send()**، حيث يتم تحديد الـ id الخاص بالأب، وقيمة tag الرسالة.


"C:\PVM-Course\PVM_Projects\fork\Debug\fork.exe"
t4000a t4000b t4000c
Length 4, Tag 11, Tid t4000a
Length 4, Tag 11, Tid t4000b
Length 4, Tag 11, Tid t4000c
Press any key to continue


つづく
TO BE CONTINUED