الاتصال بين العمليات

الاتصالات بين العمليات(IPC) intercrosses communication

يمكن للعملية أن تتصل بعملية أخري لعدد من الأسباب منها :

1. تمرير معلومات إلي عملية أخري
2. التأكد من أن عمليتين أو أكثر لن تقف احدهما في طريق الاخري.
3. التأكد من التسلسل المنطقي للعمليات .

يجب منع أن تقف احدي المعمليات في طريق الاخري مثلا تحاول عدد من العمليات الحصول علي مورد محدد أو الحصول علي الميغابايت الأخير في الذاكرة. تأمين التسلسل الصحيح للعمليات مثلا إذا كانت العملية A تقوم بتوليد أعداد عشوائية ووضعها في مصفوفة بينما تقوم العملية B بإيجاد فراغ المهمة A انظر إلي البرنامج التالي

S0: cin>>a >> b>> c>>d;

S1 : x = a + b;

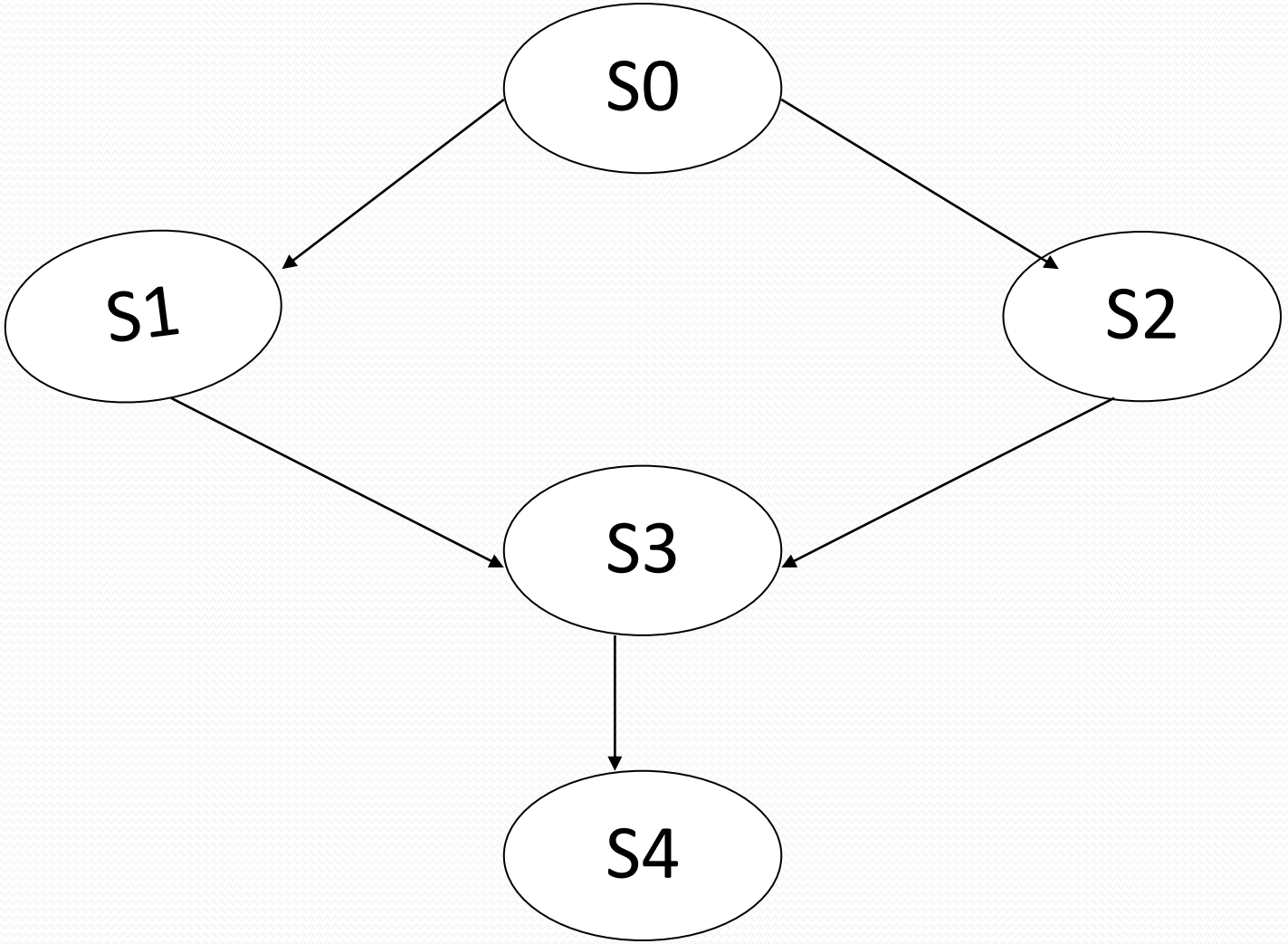
S2 : y = c + d;

S3 : z = x + y;

S4 : w = z + 10;

من الواضح انه لا يمكن حساب قيمه z قبل حساب قيمتي x,y وكذلك w لا يمكن أن تحسب قبل إيجاد قيمه z ولكن يمكن تنفيذ الجملة S1 والجملة S2 متزامنتين حيث إنهما لا تعتمدان علي بعض.

يمكن تمثيل هذه الأسبقية من المخطط التالي

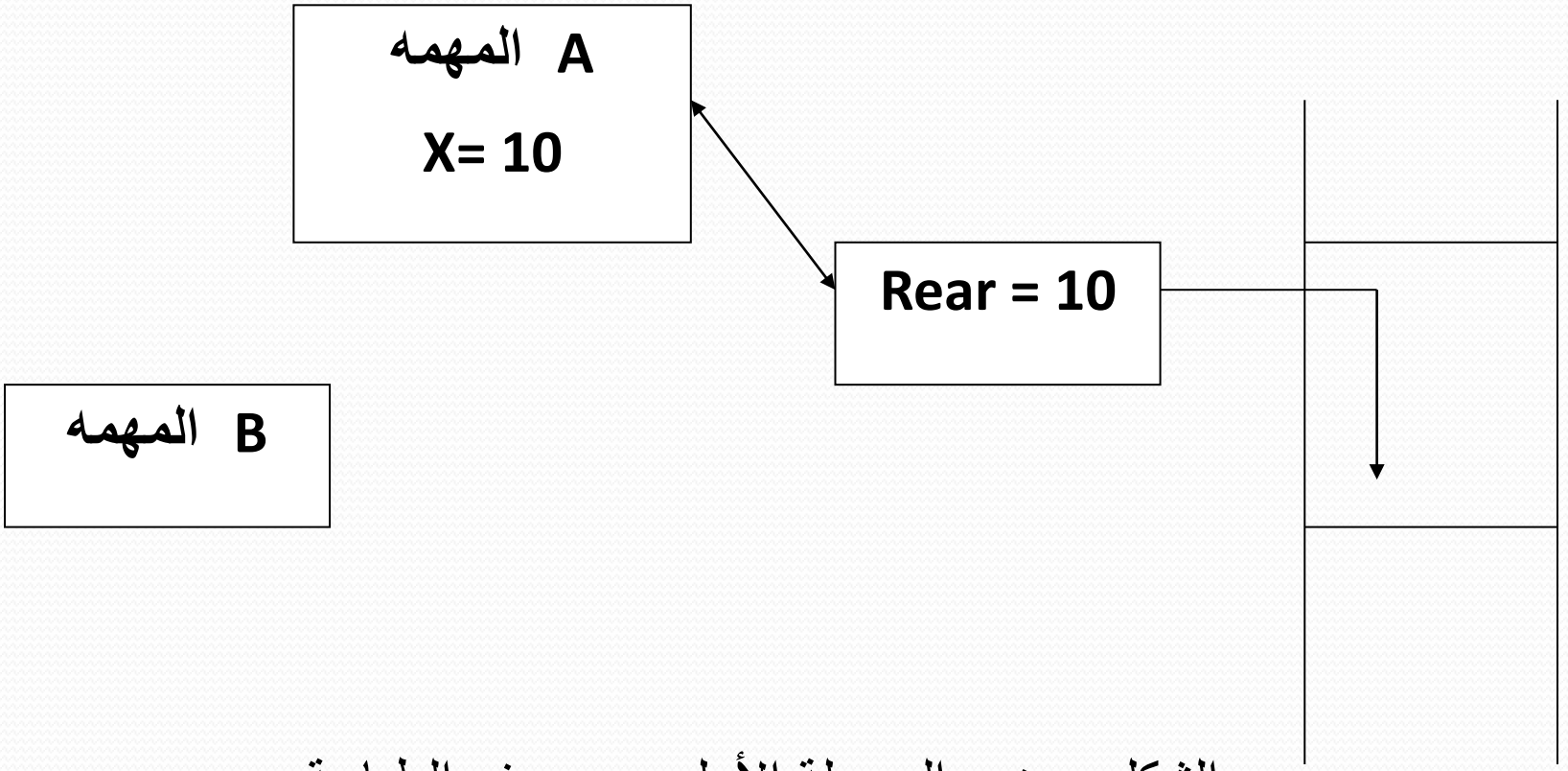

S0
S1
S2
S3
S4

تتيح عملية الاتصال بين العمليات التشارك في منطقة التخزين سواء كان علي الذاكرة أو الملفات من اجل القراءة أو الكتابة ولكن تبقي النتيجة متعلقة بمن يبدأ العمل ومتي . وينتج من هذه الحالات ما يسمي بحالات السباق (Race condition) وابسط مثال لذلك هو عملية إدارة طابعة لعدد من الملفات المراد طباعتها والواردة من عده جهات (مهمات) يتم تبديل العمليات وفقا لزمن محدد تقوم الطابعة بطباعة الملف الموجود في مقدمه الصف من خلال المؤشر front بينما يتم إضافة ملف جديد لصف الطباعة من خلال مؤخرة الصف أي المؤشر rear .

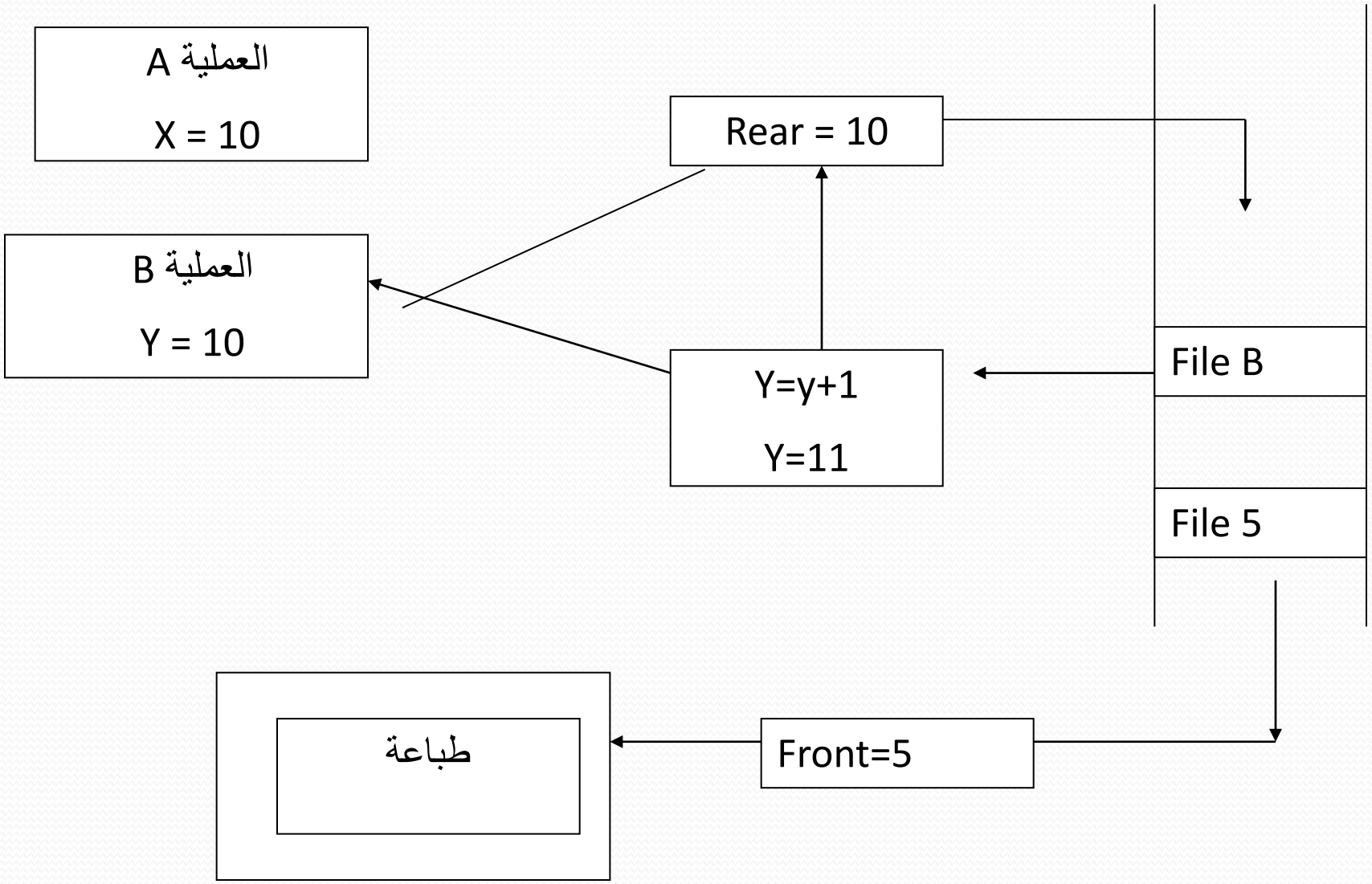
عندما تقوم عملية بإرسال ملف للطباعة تقوم بقراءة المؤشر rear وتخزنه في متغير محلي ثم تقوم بإرسال الملف ومعه قيم المتغير وهو مكان الملف في الصف

إذا أرادت العملية A إرسال ملف تقوم بقراءة المؤشر rear الذي يساوي مثلا 10 وتقوم بتخزينه في المتغير المحلي الخاص بها مثلا x أي أن قيمة x تساوي 10 . وبعد ذلك مباشرة يقرر المعالج انتهاء فترة عمل العملية A وينتقل لتنفيذ العملية B . تقوم العملية B بقراءة المؤشر rear الذي يساوي 10 ويتم تخزينه في متغير محلي خاص y ثم يتم إرسال الملف إلي الصف. يستقبل برنامج الطباعة الملف ومعه المتغير الخاص بموقعه مثلا y وهو 10 وبعد ذلك نريد تزيد قيمة y بواحد أي تصبح y=11 ويضع هذه القيمة في المؤشر rearبعد فترة تعود العملية B باستئناف العمل بناء علي برمجة المعالج وتواصل من نفس النقطة التي توقفت عندها .وتقوم بإرسال الملف ومعه قيمه المتغير x التي تساوي 10 ويقوم برنامج الطباعة بوضع الملف في الموقع 10 ويقوم بزيادة المتغير بمقدار 1 ليصبح 11 ويضعه في المؤشر rear .

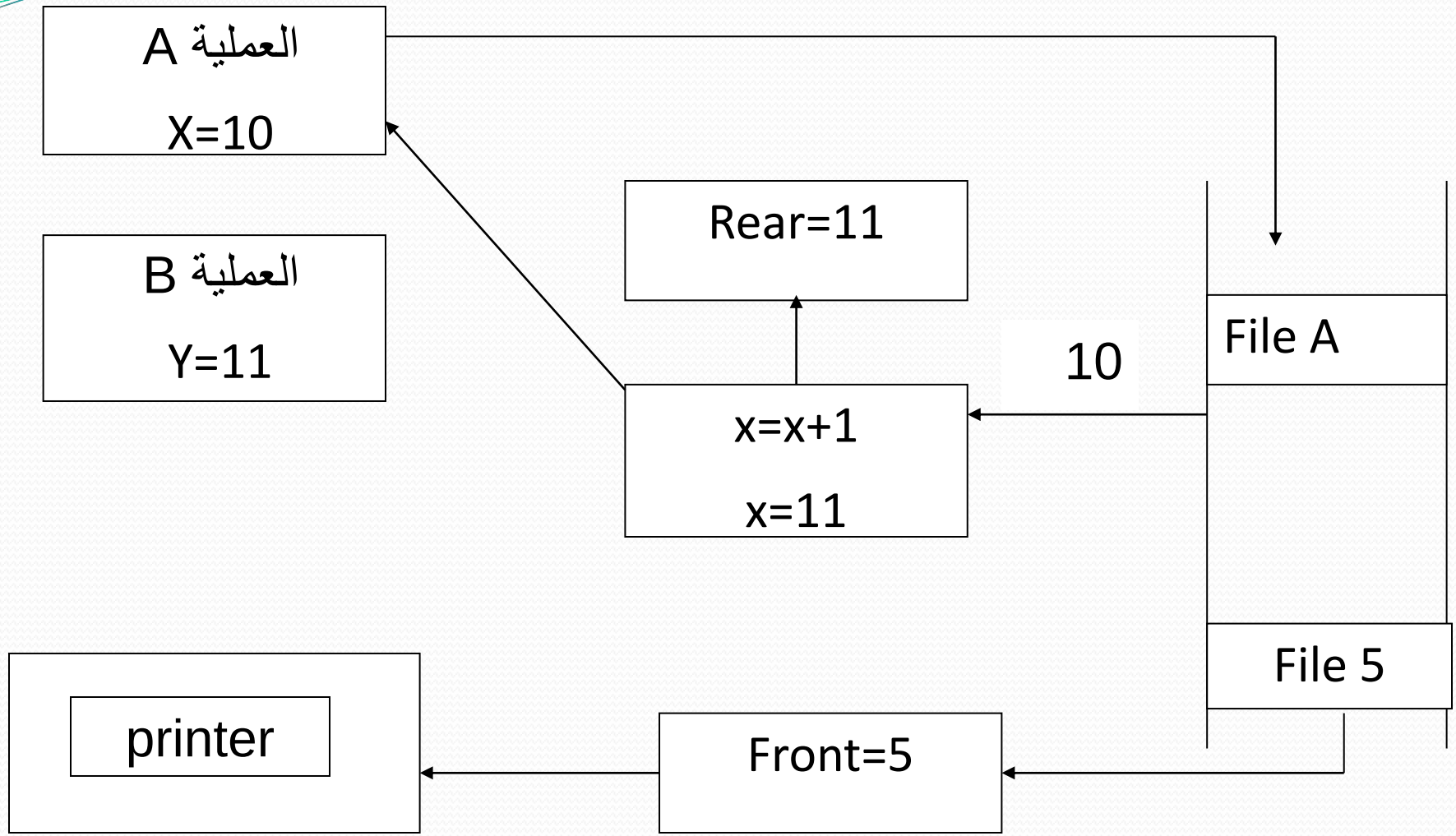
لا يلاحظ برنامج الطباعة أي خلل وسوف يكون صف الطباعة في هذه اللحظة متناسقا بعدما تم مسح الملف الخاص بالعملية B



الشكل يوضح المرحلة الأولي من صف الطباعة



الشكل يوضح المرحلة الثانية من صف الطباعة



الشكل يوضح المرحلة الثالثة من صف الطباعة

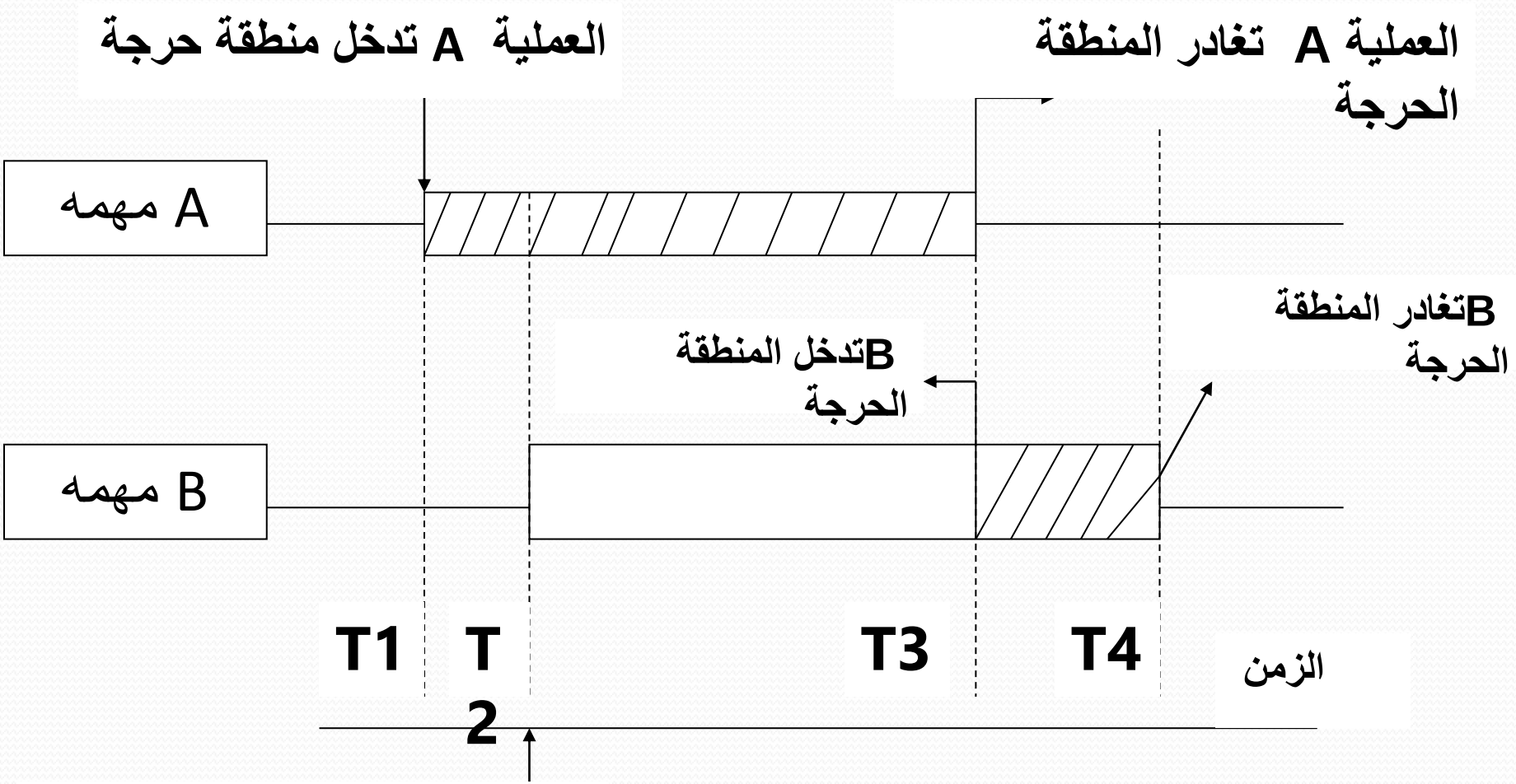
هكذا تم مسح الملف file B ولن تحصل العملية Bعلي طباعة الملف الخاص بها.

يمكن حل المشكلة السابقة وكافه المشاكل التي تتعلق بالمشاركة في منطقة تخزين واحده بالإقصاء أو الاستثناء المتبادل (mutual exclusion)

الإقصاء المتبادل (mutual exclusion) :

يعرف علي انه الطريقة التي تمنع أو ستقصي أي عملية من العمل إذا كانت هنالك عملية تعمل علي منطقة تخزين مشتركة .

تسمي منطقة التخزين المشترك بالمنطقة الحرجة(critical section) والشكل التالي يبين فكرة الاستثناء المتبادل باستخدام المناطق الحرجة .



الشكل يوضح الاستثناء المتبادل باستخدام المنطقة الحرجة

نحتاج إلي بعض المتطلبات لكي يعمل حل المنطقة الحرجة بصورة صحيحة وبكفاءة عالية وهذه المتطلبات هي :

1. عدم وجود عمليتين في مناطقهما الحرجة في نفس الوقت
2. لا يمكن افتراض أي فرضيات عن سرعات العمليات بالنسبة لبعضهما البعض وكذلك عدد المعالجات.
3. لا تستطيع العمليات التي تعمل خارج منطقتها الحرجة أن تعمل علي منع العمليات الاخري من الدخول إلي مناطقها الحرجة الخاصة بها.
4. يجب أن تكون هنالك فترة محدده لدخول العمليات الاخري لمناطقها الحرجة أي لا يوجد انتظار ابدي للعملية للدخول إلي منطقتها الحرجة.

هنالك عده طرق تحقق عملية الاستثناء المتبادل هي

1. الطريقة الأولي : تعطيل المقاطعات

عندما تدخل عملية منطقتها الحرجة تقوم بتعطيل جميع المقاطعات وهي بالتالي تقوم بتعديل الذاكرة المشتركة دون الخوف من دخول أي عملية أخري إلي هذه المنطقة وذلك لأنه معلوم لدينا أن المعالج ينقل التنفيذ من عملية إلي أخري من خلال مقاطعة الساعة التي تم إيقافها . عندما تخرج العملية من منطقتها الحرجة تقوم بتفعيل المقاطعات .

الطريقة الثانية : الانتظار المشغول busy waiting

جعل العمليات في هذه الطريقة تشترك في متغير واحد يسمي turn يأخذ القيمة 0 او 1 .إذا كانت turn =i فان المهمة Ai هي التي تعمل في المنطقة الحرجة الخاصة بها (المثال يوضح)

```
Do
{
     While (turn != i)   no-opration;
                Critical_section();
              Turn = 1-I;
            Remander_section();
} while(ture);
```

يلاحظ أن العمليات تدخل مناطقها الحرجة بالتناوب حيث تدخل العملية Ai المنطقة الحرجة أولا فإذا كانت العملية الثانية A_{1-i} مستعدة لدخول منطقتها الحرجة فإنها تجبر علي الانتظار وبعد انتهاء العملية الأولي Ai يجب أن تدخل العملية A_{1-I} منطقتها حتى أذا كانت غير مستعدة.

الطريقة الثالثة flag:

يتم تعديل مشكلة الطريقة الثانية من خلال استخدام متغير flag[i] صحيحة إذا كانت العملية Ai داخل منطقتها الحرجة ،وبعد الخروج من منطقتها تعدل قيمة flag[i] إلي falseوالشكل يبين الشفرة الخاصة العملية Ai من خلال هذه الطريقة.

```
Do
{
     Flag[i] = true ;
    While(flag[1-i]) no-opration;
    Critical_section();
     Flag[i] = false;
    Remainder_section();
}while (true);
```

الطريقة الرابعة Peterson:

تقوم فكرة (طريقة) Peterson بدمج الطريقة الثانية والثالثة معا لتحقيق الحل الصحيح لمشكلة المنطقة الحرجة وذلك بتحقق الشروط الأربعة الخاصة بالحل الصحيح والأكثر كفاءة لمشكلة المنطقة الحرجة كما ذكرنا سابقا تشترك كل العمليات في متغيرين هما:

```
Boolean flag[z];
 Int   turn;
```

حيث تأخذ turn القيمة 0 أو 1 بينما تأخذ flag[i] و flag[o] القيمة الأولية false. والشكل يبين الشفرة الخاصة بالعملية Ai من خلال طريقة Peterson

```
Do
   {
         Flag[i] = true;
        Turn = 1-i;
         While(flag[1-i]&&turn=1-i) no-operation;
           Critical_section();
           Flag[i]=false;
         Remainder_section();
}while(true);
```

يمكننا أيضا التغلب علي مشكلة المنطقة الحرجة من خلال معدات الحاسوب (hardware) إذ تحتوي العديد من المعالجات علي التعليمة test and set lock(TSL) حيث يقوم المعالج الذي ينفذ التعليمة TSL بقفل ممر الذاكرة لمنع المعالجات الاخري من الوصول إلي الذاكرة إلي إن ينتهي. والشكل التالي يوضح الشفرة البرمجية للتعليمة TSL

```
Boolean  TSL(Boolean  target)
    {
            TSL = target;
          target = true;
     }
```

```
Var
        J: 0..n-1;
       Key  Boolean;
Repeat
     Waiting[i]:=true;
     Key:=true;
     While waiting[i] and key do
       Key := TSL(lock);
      Waiting[i]:=flase;
      Critical_section();
       J:=(i+1)mod n;
```

```
While (j ≠ i) and (not waiting[i])
do
      J:=(j+1)mod n;
     If j=I then lock :=false;
         Else waiting[i] := false;
    Remainder_section();
Until false;
```

الشفرة الخاصة بالعملية A_i حسب TSL