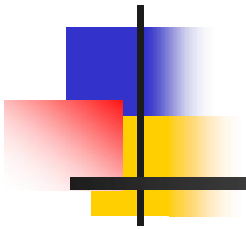
سیستم عامل

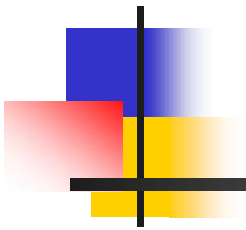


Operating Systems

جلسه دهم

مدرس: اسماعیل طغرایی

www.Teach.Toghraee.ir



همزمانی:

بن بست و گرسنگی

مباحث این فصل:

- اصول بن بست
- پیشگیری از بن بست
- اجتناب از بن بست
- کشف بن بست
- یک راهبرد مجتمع برای بن بست
- سؤالات دوره ای

بن بست:

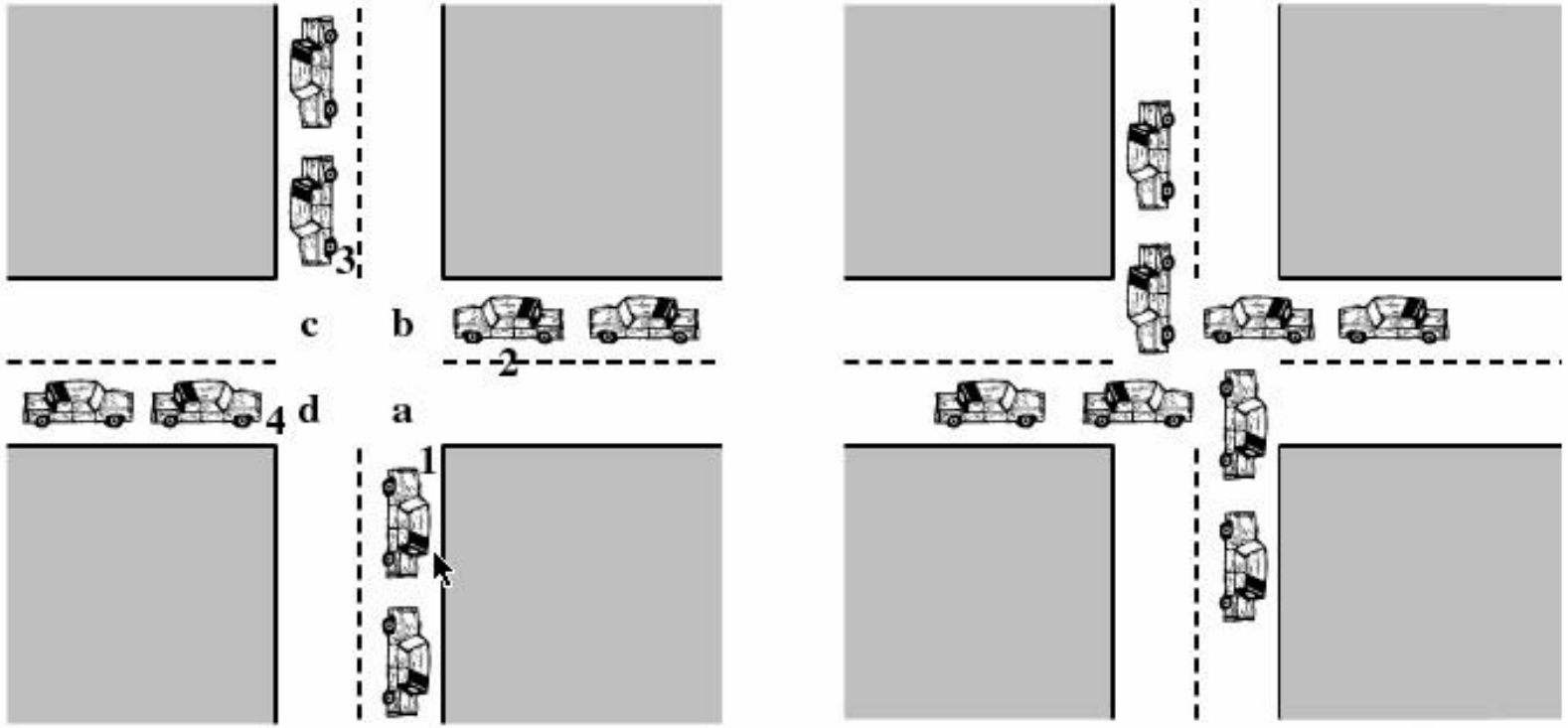
- بن بست را به صورت مسدود بودن دائمی مجموعه ای از فرایندها، که برای منابع سیستم رقابت میکنند یا با یکدیگر در ارتباط هستند. تعریف میکنند .

- برای بن بست راه حل کارآمدی وجود ندارد.

- تمام بن بست ها با نیازهای متضاد دو فرایند یا بیشتر، برای منابع همراه هستند.

مثالی از بن بست:

- یک مثال بن بست، ترافیک است.در رانندگی قانون این است که هر خودرو باید تسلیم خودروی سمت راست باشد.در این صورت در حالت زیر چه رخ میدهد؟



الف)امکان بن بست

ب) بن بست

مثالی از بن بست در فرایند ها:

- ساده ترین نوع بن بست زمانی اتفاق می افتد که ۲ فرایند مجزا و همزمان منتظر بدست آوردن منبعی هستند که در اختیار دیگریست و خود نیز منابعشان را تا اتمام فرایند آزاد نمیکنند.

Process P :

```
Get A
Get B
Release A
Release B
.
.
```

Process Q :

```
Get B
Get A
Release B
Release A
.
.
```

انواع منابع:

- منابع نقش تعیین کننده ای در بن بست دارند.
- منابع به دو دسته تقسیم میشوند:
  - منابع قابل استفاده مجدد
  - منابع مصرف شدنی یا غیر قابل استفاده مجدد

منابع قابل استفاده مجدد:

- منابعی هستند که در هر لحظه از زمان تنها توسط یک فرایند قابل استفاده اند و استفاده از آنها موجب به پایان رسیدن آنها نمیشود(بدون آسیب دیدن آزاد میشوند)
- فرایند ها منابع را بدست می آورند و سپس آنها را برای استفاده مجدد توسط سایر فرایند ها آزاد میکنند.
- پردازنده، کانالهای I/O، حافظه اصلی و ثانوی، دستگاه ها و ساختمان داده هایی مثل پرونده ها، پایگاه های داده و راهنماها از این دسته اند.
- بن بست زمانی رخ میدهد که یک فرایند منابع را نگه دارد و منبع دیگری درخواست کند.

مثالی از بن بست منابع قابل استفاده مجدد:

- فرض کنید فضای حافظه که میتواند به فرایند ها تخصیص یابد ۲۰۰ KB باشد.در این صورت ترتیب زیر موجب بروز بن بست میشود.

P1

. . .

Request 80 Kbytes;

. . .

Request 60 Kbytes;

P2

. . .

Request 70 Kbytes;

. . .

Request 80 Kbytes;

منابع مصرف شدنی:

- این منابع تولید میشوند و از بین میروند
- وقفه ها، علامتها، پیامها و اطلاعات بافر I/O از این نمونه اند.
- کشف بن بستهای حاصل از این منابع بسیار مشکل است و ممکن است ترکیب نادری از حوادث آنها را ایجاد کند.

مثالی از بن بست منابع قابل استفاده مجدد:

- در صورتیکه عمل Receive مسدود کننده باشد بن بست اتفاق می افتد.

P1

```
. . .
Receive(P2);
. . .
Send(P2, M1);
```

P2

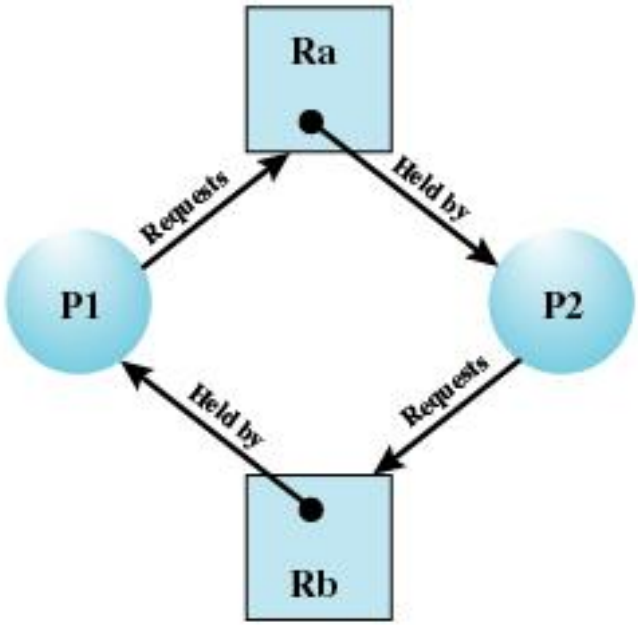
```
. . .
Receive(P1);
. . .
Send(P1, M2);
```

شرایط لازم برای ایجاد بن بست:

- **انحصار متقابل:**
  - در هر لحظه تنها یک فرایند میتواند از یک منبع استفاده کند.
- **نگهداشت و انتظار:**
  - هنگام درخواست منبع جدید فرایند منابع قبلی تخصیص یافته را آزاد نمیکند.
- **قبضه نکردن:**
  - منابع به زور قابل پسگیری نیستند.

شرایط لازم برای ایجاد بن بست:

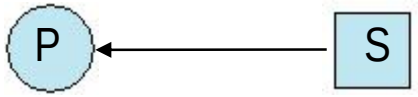
- انتظار مدور:
  - زنجیر بسته ای از فرایند ها وجود دارد، بطوریکه هر یک حداقل یک منبع مورد نیاز فرایند بعد در زنجیره را نگه دارد.



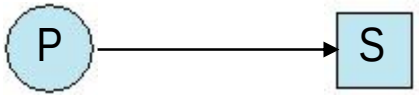
(c) Circular wait

گراف تخصیص منابع:

- گراف تخصیص منابع یک گراف جهت دار است که نحوه تخصیص منابع به فرایند ها را در هر لحظه از زمان نشان میدهد.
- برای تشخیص بن بست باید گراف تخصیص منابع را بعد از هر درخواست، هر تخصیص، یا هر ترخیص به روز کرد.
- در گراف ، منابع را با ▭ و فرایند را با ◯ نشان میدهیم.
- وجود چرخه در گراف بیانگر بروز بن بست است.



فرایند P منبع S را در اختیار دارد



فرایند P در انتظار منبع S است

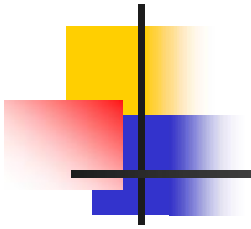
جلوگیری از بن بست:

- جلوگیری از بن بست با نقض کردن یکی از شرایط ۴ گانه لازم برای بن بست انجام میشود:

- **انحصار متقابل**: این شرط را نمیتوان رد کرد، چرا که بعضی از منابع ذاتاً انحصاری هستند. مثلاً یک چاپگر تنها میتواند به یک فرایند پاسخ دهد یا نوشتن بر روی یک بانک اطلاعاتی تنها توسط یک فرایند انجام میشود.

جلوگیری از بن بست:

- **نقض نگهداشت و انتظار**: میتوان فرایند ها را ملزم ساخت که اختصاص منابع تنها زمانی انجام شود که تمام منابع مورد نیاز فرایند آزاد باشد و حتی اگر یک منبع آماده نبود هیچ اختصاصی انجام نشود.
- معایب این روش:
  - انتظار طولانی فرایند برای تکمیل منابعش
  - بیکار ماندن یک منبع به مدت طولانی
  - عدم پیش بینی منابع مورد نیاز در آینده



جلوگیری از بن بست:

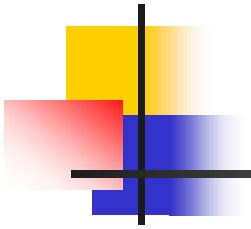
- **نقض شرط غیر قابل استفاده مجدد**
  - پاسخ به درخواست منبع جدید در صورت آزاد شدن منابع قبلی
  - قبضه کردن فرایند با اولویت پایینتر: زمانی که یک فرایند نیاز به منبعی دارد که فرایند دیگری آنرا نگه داشته است، سیستم عامل آنرا قبضه کرده و منابعش را آزاد میکند.
    - در فرایند های اولویت بندی شده استفاده میشود.
    - حالت منبع باید براحتی قابل ذخیره باشد تا بعدها بازیابی شود.

جلوگیری از بن بست:

- **نقض شرط انتظار مدور:** مرتب کردن منابع به صورت خطی:به هر منبع یک شاخص نسبت داده میشود. در این صورت فرایند میتواند ابتدا منبع Ri و سپس منبع Rj را درخواست کند اگر i<j
- معایب این روش:
  - کند شدن فرایندها
  - رد کردن غیر ضروری دسترسی به منابع

اجتناب از بن بست:

- در این استراتژی سعی در پیش بینی آینده داریم و تلاش میکنیم به دو صورت مانع از بروز بن بست شویم:
  - عدم شروع فرایندی که ممکن است درخواست هایش موجب بن بست شود.
  - عدم پاسخ به درخواست فرایندی برای منبع که این تخصیص موجب بن بست میشود.



اجتناب از بن بست:

- عدم شروع فرایند:
- سیستمی از n فرایند و m نوع مختلف از منابع را در نظر بگیرید. بردارها و ماتریسهای زیر را تعریف میکنیم:
  - آرایه R[i]
  - آرایه Available[i]
  - ماتریس دو بعدی Allocation[i][j]
  - آرایه max[i][j]
  - ماتریس دوبعدی Need[i][j]

اجتناب از بن بست:

- Resource={R1,R2,R3,...,Rm}
- Available={A1,A2,A3,...,Am}

R[i]= i تعداد منابع کل اولیه از منبع نوع

A[i]= i تعداد منابع آزاد از منبع نوع

- Max=

$$\begin{pmatrix} C11 & C12 & \dots & C1m \\ C21 & C22 & \dots & C2m \\ \vdots & \vdots & \vdots & \vdots \\ Cn1 & Cn2 & \dots & Cnm \end{pmatrix}$$

max[i][j]=k فرایند i حداکثر k تا منبع نوع j نیاز دارد

- Allocation=

$$\begin{pmatrix} A11 & A12 & \dots & A1m \\ A21 & A22 & \dots & A2m \\ \vdots & \vdots & \vdots & \vdots \\ An1 & An2 & \dots & Anm \end{pmatrix}$$

A[i][j]=k به فرایند i، k تا منبع نوع j تخصیص یافته

اجتناب از بن بست:

- ۳ نکته بدیهی است:
  - تعداد منابع یا تخصیص داده شده اند یا موجودند.
  - هیچ فرایندی نمیتواند بیش از کل منابع سیستم درخواست کند.

$$C_{ki} < R_i \quad \text{for all i , k}$$

  - تعداد فرایندهای اختصاص یافته نمیتواند بیش از حداکثر نیاز فرایند باشد.

$$A_{ki} < C_{ki} \quad \text{for all k , i}$$

- بنابراین برای اجتناب از بن بست فرایند را تنها درصورتی شروع میکنیم که:

$$R_i \geq C_{(n+1)i} + \sum_{k=1}^{n} C_{ki} \quad \text{for all i}$$

اجتناب از بن بست:

- عدم تخصیص منبع: به این راهبرد الگوریتم بانکداران نیز گفته میشود.
- حالت سیستم، تخصیص جاری منابع به فرایندها است، لذا حالت سیستم شامل دو بردار Available,Resource و دو ماتریس Max,Allocation میباشد.
- حالت مطمئن: حالتی است که در آن حداقل یک ترتیب از فرایندها وجود دارد که میتوانند اجرا و کامل شوند، بدون اینکه حالت بن بست ایجاد شود.

الگوریتم بانکداران:

- مقادیر مورد نیاز هر فرایند را بدست می آوریم.(Need)
- کمترین را در نظر میگیریم. اگر کمترین مقدار از مقادیر منبع باقیمانده بیشتر بود، نتیجه میگیریم که حالت ناامن است. در غیر این صورت فرایند با کمترین نیاز را حذف میکنیم و تعداد منابع تخصیص یافته آنرا را آزاد میکنیم و مراحل را دوباره تکرار میکنیم.

الگوریتم بانکداران:

- توجه : در حل مسائل بانکداران باید جداول و اطلاعات زیر را داشته باشیم:
  - جدول Allocation
  - جدول Max **یا** جدول Need
  - جدول Resource **یا** جدول Available
- سپس بر حسب ورودی باید جداول زیر را بدست آوریم:
  - جدول Allocation
  - جدول Need
  - Available

الگوریتم بانکداران:

- با توجه به جداول زیر بگویید آیا حالت امن است؟
  - این جدول بیان میکند تنها یک نوع منبع داریم.
  - فرایند A به ۶ منبع نیاز دارد و در حال حاضر ۱ منبع به آن اختصاص یافته است.(برای فرایندB نیز به همین صورت)
  - در حال حاظر از کل تعداد منابع تنها ۲ منبع آزاد است.(Available=2)

Availavle=2		
فرایند	Allocation	Max
A	1	6
B	1	5
M	2	4
S	4	7

الگوریتم بانکداران:

- محاسبه Need :

(Need=Max-Allocation)

Availavle=2			
فرایند	Allocation	Max	Need
A	1	6	6 5 1
B	1	5	5 4 1
M	2	4	4 2 2
S	4	7	7 3 4

الگوریتم بانکداران:

پیدا کردن کوچکترین مقدارNeed:

Availavle= 4			
فرایند	Allocation	Max	Need
A	1	6	5
B	1	5	4
M	2	4	2
S	4	7	3

2<2(Available)

Available=Available + 2

الگوریتم بانکداران:

پیدا کردن کوچکترین مقدار Need:

Availavle= 8			
فرایند	Allocation	Max	Need
A	1	6	5
B	1	5	4
M	2	4	2
S	4	7	3

3<4(Available)

Available=Available + 4

الگوریتم بانکداران:

پیدا کردن کوچکترین مقدارNeed:

Availavle= 9			
فرایند	Allocation	Max	Need
A	1	6	5
B	1	5	4
M	2	4	2
S	4	7	3

4<8(Available)

Available=Available + 1

الگوریتم بانکداران

پیدا کردن کوچکترین مقدارNeed:

Availavle= 10			
فرایند	Allocation	Max	Need
A	1	6	5
B	1	5	4
M	2	4	2
S	4	7	3

5<9(Available)

Available=Available + 1

الگوریتم بانکداران

- چون توانستیم تمام فرایند ها را حذف کنیم بنابراین حالت داده شده در صورت مسأله امن است.
- توجه داشته باشید که در این مسأله فرض شد که تنها یک نوع منبع وجود دارد . در مثال بعدی مسأله را با انواع مختلف منابع بررسی میکنیم.

الگوریتم بانکداران:

- در این حالت نیز همانند قبل عمل میکنیم. با این تفاوت که کمترین تعداد نیاز یک سطر است.
- بنابراین ابتدا مقدار نیاز را محاسبه میکنیم.
- تعداد نیاز های هر فرایند را جمع میزنیم و کمترین را انتخاب میکنیم.
- اگر سطر مربوط به کمترین مقدار از مقادیر منبع باقیمانده بیشتر بود، نتیجه میگیریم که حالت ناامن است. در غیر این صورت
- فرایند با کمترین نیاز را حذف میکنیم و تعداد منابع تخصیص یافته آنرا را آزاد میکنیم و مراحل را دوباره تکرار میکنیم.

الگوریتم بانکداران:

- مثال: آیا حالت زیر یک حالت امن است ؟

Allocation				
فرایند	R	S	T	U
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

MAX				
فرایند	R	S	T	U
A	4	1	1	1
B	0	2	1	2
C	4	2	1	0
D	1	1	1	1
E	2	1	1	0

Resource	
نوع	تعداد
R	6
S	3
T	4
U	2

الگوریتم بانکداران:

- مرحله اول :جدول Need را محاسبه میکنیم.

Allocation					Max				Need=max-Alloc				
فرایند	R	S	T	U	R	S	T	U	R	S	T	U	+
A	3	0	1	1	4	1	1	1	1	1	0	0	2
B	0	1	0	0	0	2	1	2	0	1	1	2	4
C	1	1	1	0	4	2	1	0	3	1	0	0	4
D	1	1	0	1	1	1	1	1	0	0	1	0	1
E	0	0	0	0	2	1	1	0	2	1	1	0	4

الگوریتم بانکداران:

- جدول Available را محاسبه میکنیم.

Allocation				
فرایند	R	S	T	U
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0
+	5	3	2	2

Resource	
نوع	تعداد
R	6
S	3
T	4
U	2

Available= Res[i] - A[+][i]	
نوع	تعداد
R	615
S	303
T	422
U	202

الگوریتم بانکداران:

- چون مقدار آن از منابع باقیمانده کمتر است سطر را حذف و منابع را آزاد میکنیم.
- با توجه به جدول Need کدام فرایند کمترین تعداد منابع را نیاز دارد؟
- مقدار این سطر را با منابع باقیمانده مقایسه میکنیم
- عملیات را برای بقیه سطر ها نیز انجام میدهیم.

Allocation					Need				
فرایند	R	S	T	U	R	S	T	U	+
A	3	0	1	1	1	1	0	0	2
B	0	1	0	0	0	1	1	2	4
C	1	1	1	0	3	1	0	0	4
D	1	1	0	1	0	0	1	0	1
E	0	0	0	0	2	1	1	0	4

Available	
نوع	تعداد
R	1
S	0
T	2
U	0

معایب الگوریتم بانکداران:

- الگوریتم بانکداران و اجتناب از بن بست نمیتواند به طور حتم بروز بن بست را پیشبینی کند. گاهی آنرا حدس میزند و گاهی آنرا اطمینان میدهد.
- معایب روش اجتناب از بن بست:
  - حداکثر منابع باید از قبل مشخص باشد.
  - فرایند های مورد نظر باید مستقل باشند (همگام سازی نشده باشند)
  - تعداد منابع تخصیصی باید ثابت باشد.
  - فرایندی که منبعی را در اختیار داشته باشد نمیتواند خارج گردد

کشف بن بست:

- روشی است که در آن اجازه برای انجام فرایند و هر تخصیص منبع داده میشود و سیستم عامل مرتباً وجود بن بست را بررسی میکند.
- در راهبرد کشف بن بست به صورت زیر عمل میشود:
  - قطع تمام فرایندهای در بن بست
  - برگشت هر یک از فرایند های در بن بست به نقاطی که از قبل برای بررسی تعیین شده اند و شروع مجدد تمام فرایند ها
  - قطع پیگیری فرایندهای در بن بست تا جایی که دیگر بن بست وجود نداشته باشد
  - قبضه کردن پی در پی منابع تا جایی که دیگر بن بست وجود نداشته باشد.

کشف بن بست:

- برای بند های ۳ و ۴ معیار انتخاب میتواند یکی از موارد زیر باشد:
  - کمترین وقت مصرفی از پردازنده تا کنون
  - کمترین خروجی تولید شده تا کنون
  - بیشترین زمان باقیمانده تخمینی
  - کمترین منابع تخصیص داده شده تا کنون
  - کمترین اولویت

راهبرد مجتمع برای بن بست:

- هر راهبرد بن بست مزایا و معایب مخصوص به خود را دارد شاید بهتر باشد از ترکیبی از راهبرد ها استفاده کنیم:
  - تقسیم بندی منابع در تعدادی گروه های مختلف
  - استفاده از راهبرد مرتب سازی خطی
  - در داخل یک گروه از منابع استفاده از بهترین الگوریتم برای آن گروه

خلاصه رویکرد های پیشگیری، اجتناب و کشف بن بست:

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates on-line handling	•Inherent preemption losses