Оперативни Системи

Блокада

Вон. Проф. Д-р Димитар Трајанов
Вон. проф. Д-р Невена Ацковска
Доц. Д-р Боро Јакимовски

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Цел на предавањето

- Ресурси – модел на систем
- Појава на блокада
- Детекција
- Справување
  - Превенција
  - Одбегнување
  - Дозволи и реагирај
  - Не прави ништо

Модел на систем – ресурси на систем

- Системот се состои од конечен број ресурси дистрибуирани меѓу процеси кои се натпреваруваат.
- Ресурсите се делат на неколку типови, од кои секој има одреден број на идентични инстанци
  - Меморискиот простор, CPU циклусите, датотеките и I/O уредите (како принтери) се пример за типови ресурси.
  - Ако системот има 2 CPU, тогаш ресурсот од тип *CPU* има две инстанци. Слично, ресурсот од тип *принтер* може да има 5 инстанци
- Ако процесот бара инстанца од ресурсот, тогаш било која инстанца од ресурсот ќе го задоволи барањето.
  - Во спротивно, не се работи за ист ресурс

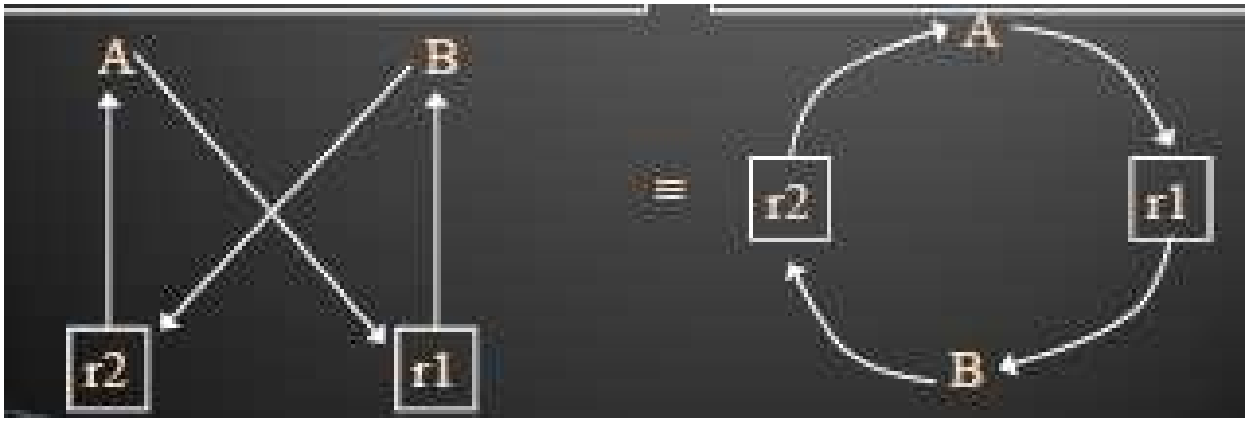
За проблемот

- Ситуација кога 2 или повеќе процеси се чекаат меѓусебно неограничено
- Множество процеси се блокирани кога некој од нив чека за настан што може да биде предизвикан само од некој друг процес од множеството процеси
- Се случува често кога се користат взаемни исклучувања (работа со критични секции),
  - **Пример**:
  - Процесот А го има печатачот, а бара датотека или нејзин слог
  - Процесот В ја има датотеката или слогот, а го бара печатачот

Релевантен пример во бази на податоци

Proces A
lock(r2);
…
lock(r1); / се обидува

Proces B
lock(r1);
…
lock(r2); / се обидува



Блокадата е лоша

Бидејќи:

- Таа го спречува процесот да „напредува“
- Блокадата бара интервенција за да биде прекината
- Таа ја намалува искористеноста на ресурсите
- Сите процеси чекаат
  - Ниту еден нема да направи настан кој би разбудил некој друг процес
  - Обично се чека на ослободување на ресурс, зафатен од друг блокиран процес

- Ниту еден од процесите НЕ
  - работи
  - ослободува ресурс
  - може да биде разбуден

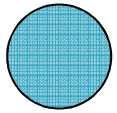
- Сите процеси бесконечно чекаат

Граф на алокација на ресурси

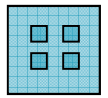
- Множество темиња V и множество лаци E
- V е партиционирано во два вида:
  - $P = \{P_1, P_2, \ldots, P_n\}$, множество што ги содржи сите процеси во системот
  - $R = \{R_1, R_2, \ldots, R_m\}$, множество што ги содржи сите ресурси во системот
- лак на барања – насочен лак $P_i \rightarrow R_j$
- лак на доделување – насочен лак $R_j \rightarrow P_i$

Граф на алокација на ресурси (2)

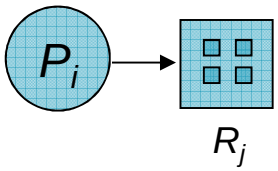
- Процес



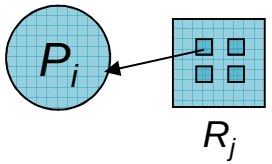
- Ресурс од 4 инстанци



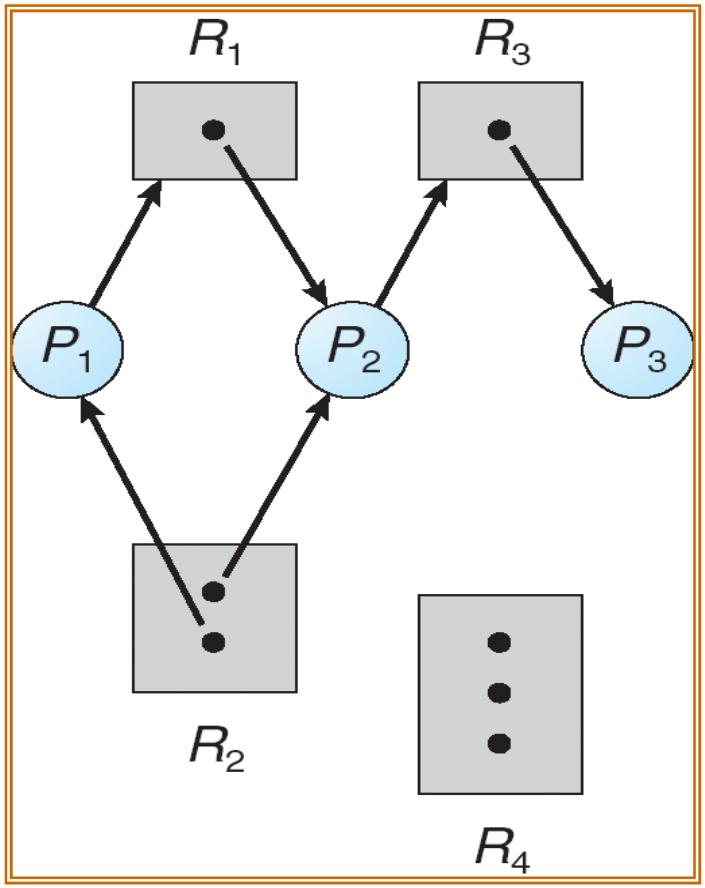
- P_i бара една инстанца на R_j



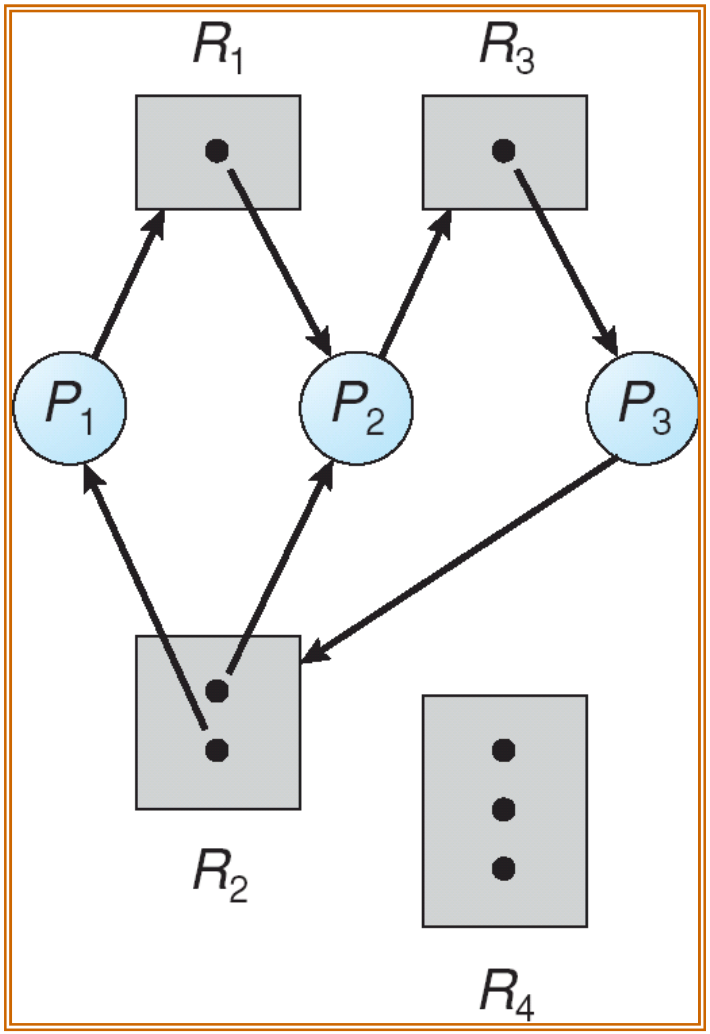
- P_i држи инстанца на R_j



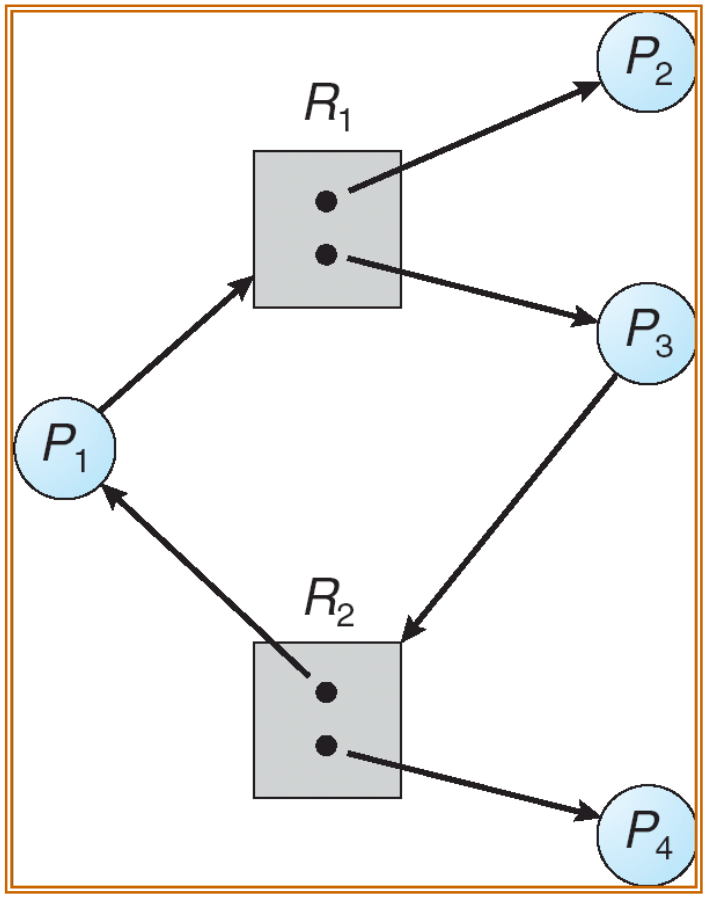
Пример на граф на алокација на ресурси



Граф со блокада



Граф со циклус, но без блокада



Основни факти

- ако графот нема циклуси $\Rightarrow$ нема блокада
- ако графот има циклус $\Rightarrow$
  - ако има само по една инстанца од ресурсот, тогаш има блокада
  - ако има повеќе инстанци од ресурсот, има можност за блокада

Блокада се случува кога 4 услови се точни истовремено:

- Взаемно исклучување: само еден процес во еден момент може да го користи ресурсот
- Држи и чекај: процес кој држи барем еден ресурс чека да добие дополнителни ресурси кои ги држат други процеси
- Нема испразнување: ресурсот може да биде ослободен само ако процесот го дозволи тоа, по комплетирање на неговата задача
- Кружно чекање: постои множество $\{P_0, P_1, \ldots, P_n\}$ чекачки процеси такви што P_0 чека за ресурс кој го држи P_1, P_1 чека за ресурс кој го држи P_2, ..., P_{n-1} чека за ресурс кој го држи P_n, и P_n чека за ресурс кој го држи P_0.

Решенија

- Во општ случај, постојат 4 стратегии за решавање на блокадите:
  - Превенција, реализирај ги ситуациите на конкурентност во ОС така што блокада да не може да се случи
  - Одбегнување, предвиди блокада и одбегни ја
  - Дозволи блокада (согледај ја) и реагирај
  - Не прави ништо (најчесто се користи во реалноста)

1. Превенција (како да се избегне блокадата)

Услов на взаемно исклучување (1)

- Ако се избегне ексклузивно доделување ресурс на некој процес – НЕМА блокада...
- Не е задолжително за деливите ресурси, но исклучување мора да има кај што ресурсите не можат да се делат
- Користено во јадрата на ОС

Услов за hold и wait

- Мора да се гарантира дека кога процесите бараат ресурс, тие веќе не држат друг ресурс.
  - Мора процесите да бараат и да им се доделат сите ресурси пред да започне извршувањето,
  - или да му се доделат ресурси на процесот кога не држи ниту еден ресурс.

  - Слаба искористеност на ресурсите, можно е изгладнување

Услов за одземање ресурси (испразнување)

- Ако процесот кој држи некој ресурс побара друг што не може да му биде доделен веднаш, тогаш ги ослободува сите ресурси кои во моментот ги држел
- Одземените ресурси му се додаваат на листа ресурси за кои процесот чека
- Процесот ќе биде рестартиран само кога ќе може да ги добие повторно своите стари ресурси, како и новите кои му требаат
- Ова е најлош услов за превенција на блокади:
  - Ако на процес му е доделен на пр. принтер и се наоѓа во среде работа, насилно одземање на принтерот затоа што моментално не е слободен исто така потребниот плотер е лошо, па и невозможно!

Услов за кружно чекање

- Правило: само еден ресурс може да се додели на процес. Кога ќе му треба втор ресурс, да го ослободи првиот.
  - Неприфатливо: процес кој копира голема датотека од лента на принтер!
- Правило: наметни линеарно подредување на ресурсите
- Присили ги процесите да ги бараат ресурсите во дадениот редослед
  - процес што држи некој ресурс, не може да побара ресурс со понизок реден број, се додека не го ослободи својот ресурс
- Постои сценарио во кое сите процеси завршуваат блокадата никогаш нема да се случи
- Но:
  - Нефлексибилно
  - Не може да се прилагоди за големи програми/системи

2. Одбегнување

- Кај најголемиот број системи ресурсите се побаруваат еден по еден
- ОС треба да одлучи дали е сигурно да се додели некој ресурс
- Прашање: Дали постои алгоритам со кој секогаш може да се избегнуваат блокади?
- ДА...
  ... Но за да се реализира, потребно е да се знаат некои работи однапред (кои и колку ресурси се потребни по процес!)

Модел за одбегнување

- Потребно е системот да има некое претходно знаење
  - Наједноставен и најкорисен модел е секој процес да декларира *максимален број* на ресурси од секој тип што може да ги побара.
  - Алгоритмот за одбегнување блокада динамички го прегледува графот за алокација на ресурси за да осигура дека нема да се случи кружно чекање.
  - *Состојба* на доделување ресурси се дефинира преку бројот на слободни и алоцирани ресурси и максималните барања на процесот

Сигурна состојба

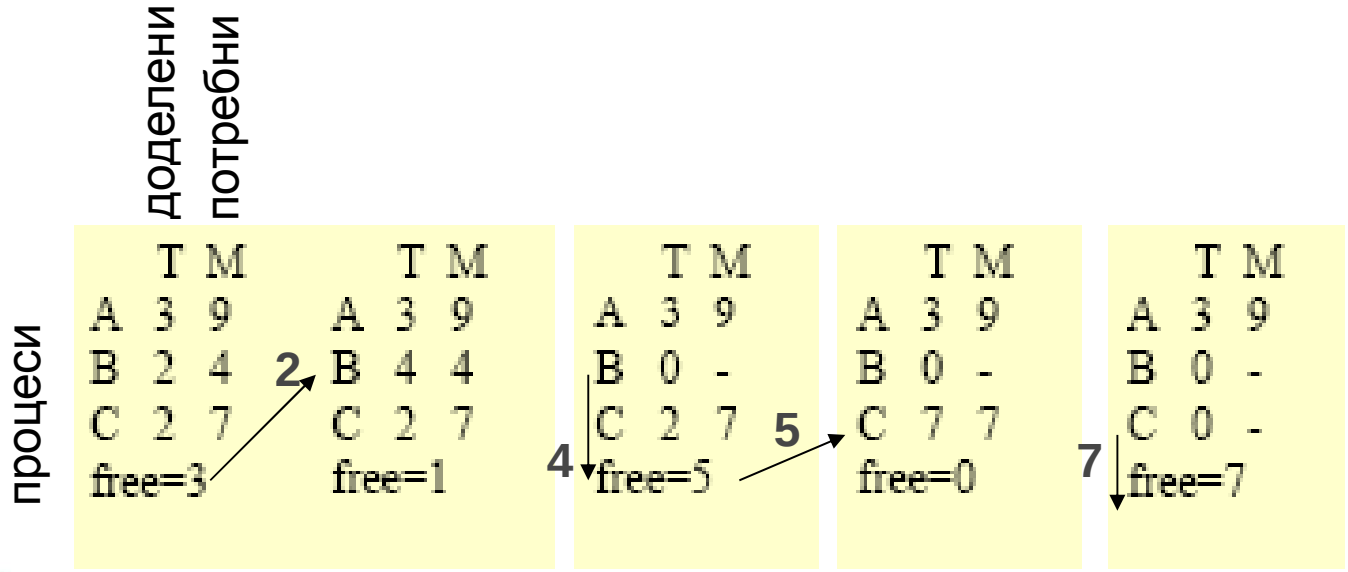
- Кога на процесот му треба слободен ресурс, системот мора да одлучи дали неговото доделување ќе го остави системот во сигурна состојба

- Системот е во сигурна состојба ако постои секвенца $<P_1, P_2, \ldots, P_n>$ на СИТЕ процеси во системот така да за секој P_i, ресурсите кои P_i може сеуште да ги побара можат да бидат задоволени од тековно слободните ресурси + ресурсите држени од сите P_j, со $j < i$.
- Ова значи:
  - ако потребите за ресурси на P_i не можат да се остварат веднаш, тогаш P_i ќе чека додека другите P_j завршат.
  - кога P_j завршил, P_i може да ги добие потребните ресурси, да се изврши, да ги врати алоцираните ресурси и за терминира.
  - кога P_i терминирал, P_{i+1} може да ги добие бараните ресурси, итн.

Пример за сигурна состојба

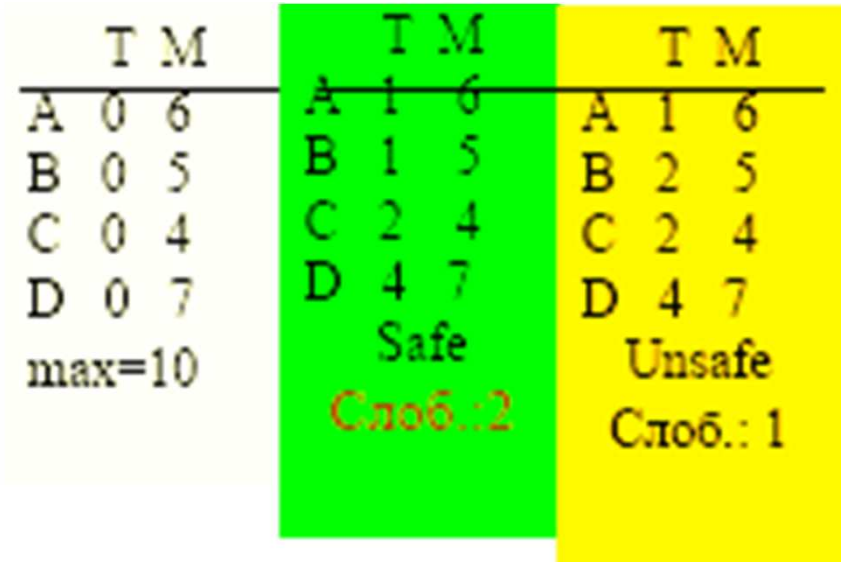
- Состојба се нарекува сигурна, ако таа не е блокирана и ако постои начин да се задоволат сите потреби за ресурси кај тековните процеси, активирајќи ги процесите по некој редослед

- Пр1. За еден ресурс, со 10 инстанци



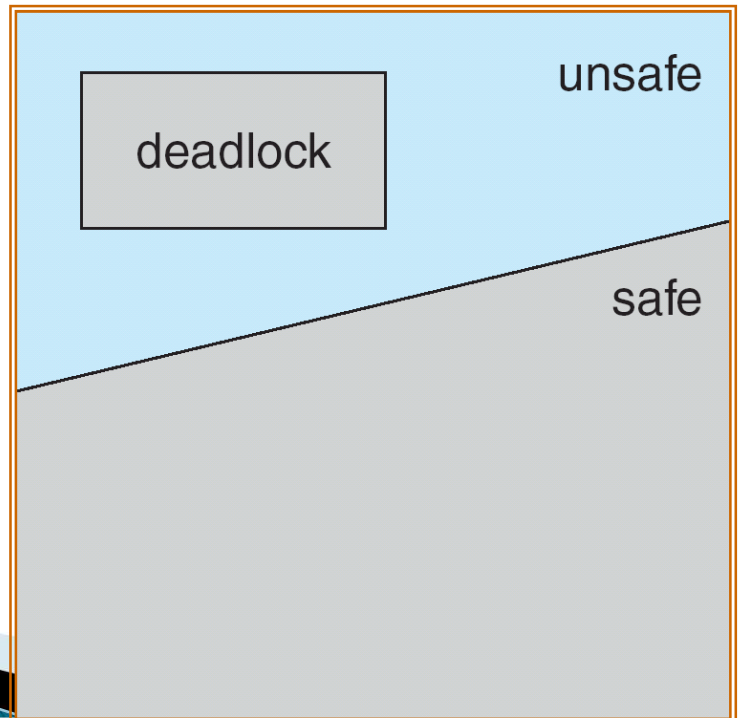
Пример за несигурна состојба

- Систем со еден тип ресурси
- Вкупно 10 инстанци
  - Втората ситуација **е сигурна**,
  - Третата **не е сигурна**



Сигурна, несигурна и блокирана состојба

- ако системот е во сигурна состојба ⇒ нема блокада
- ако системот е во несигурна состоја ⇒ можност за блокада
- одбегнување ⇒ осигурај дека системот нема никојпат да влезе во несигурна состојба



Алгоритми за одбегнување

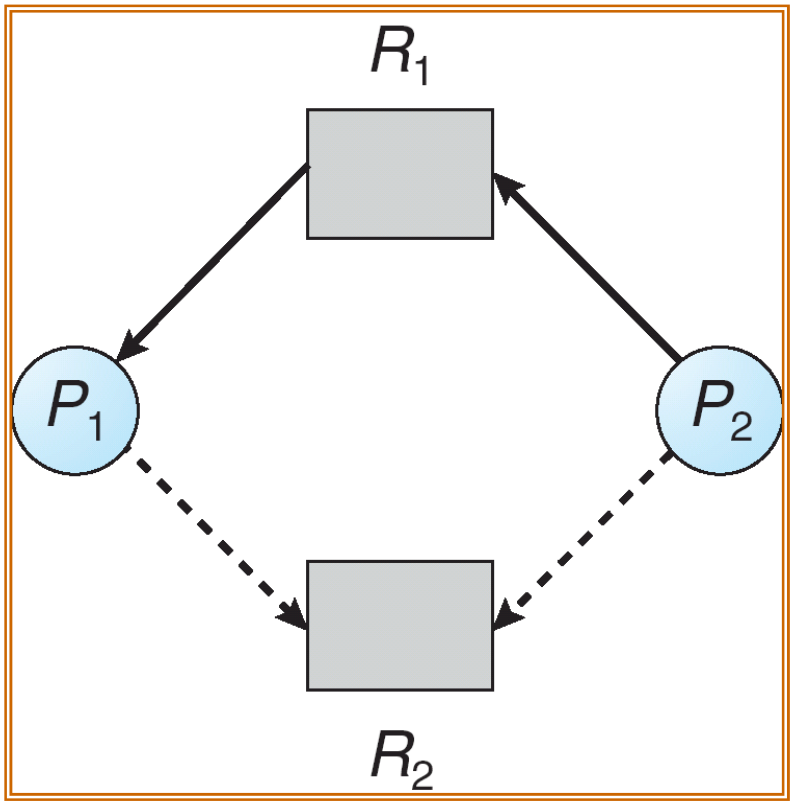
- Ако имаме по една инстанца од секој ресурс може да користиме граф за алокација на ресурси

- Ако имаме повеќе инстанци од некој тип ресурс користиме Банкаров алгоритам

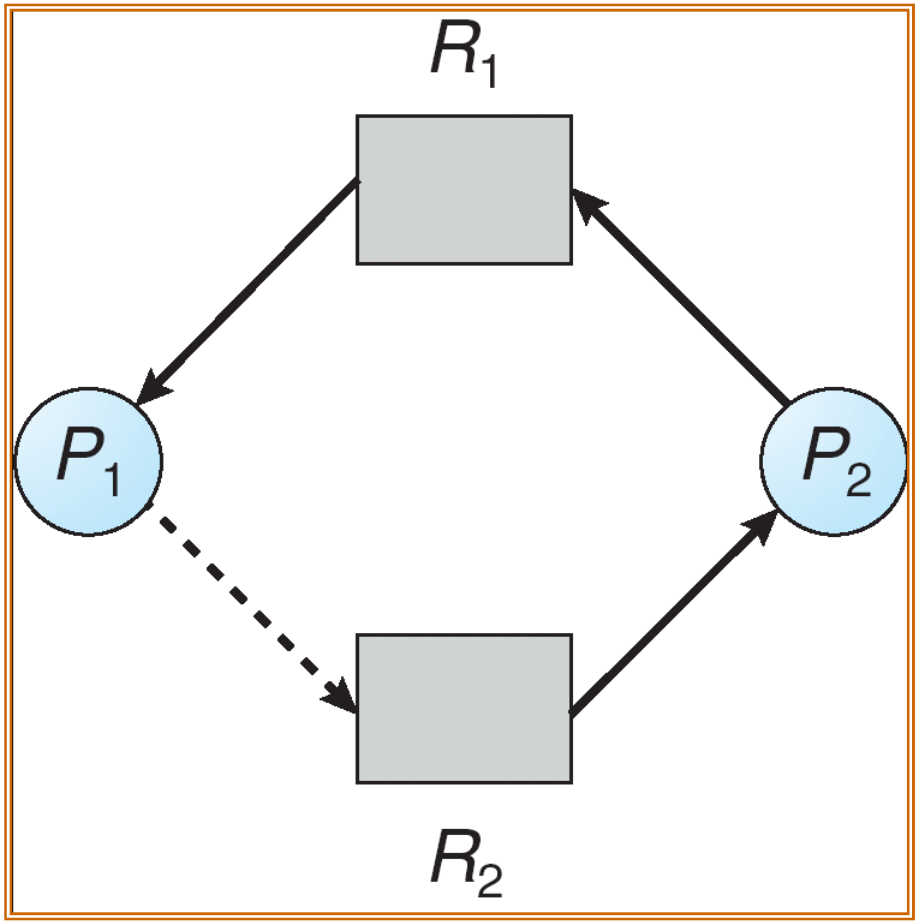
Користење алокациски граф

- *Поднеси барање* за лакот $P_i \rightarrow R_j$ што значи дека процесот P_i може да го побара ресурсот R_j; презентирано со испрекинати линии.
- Лакот за поднесено барање се трансформира во лак за барање кога процесот ќе го побара ресурсот
- Лакот за барање се претвора во лак за доделување кога ресурсот му е доделен на процесот
- Кога ресурсот е ослободен од процесот, лакот за доделување се враќа во лак за поднесување барање
- Ресурсите мора *однапред* да бидат побарани во системот

Пример – поднесување барање за ресурс



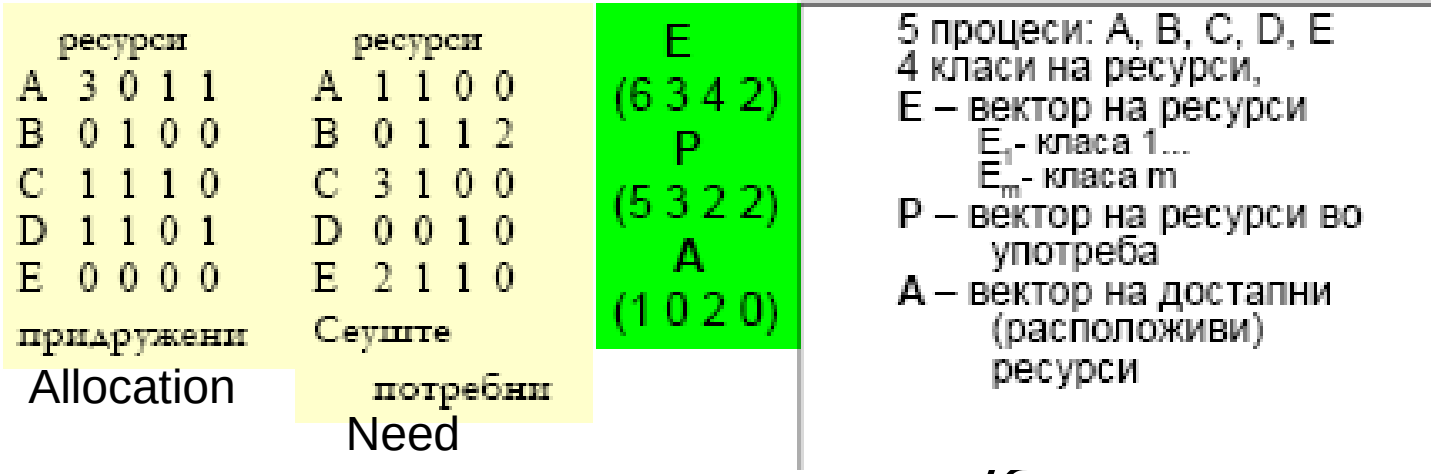
Несигурна состојба во графот за алокација ресурси



Алгоритмот за алокација на ресурси

- Претпостави дека процесот P_i бара ресурс R_j
- Барањето ќе биде удоволено само ако конвертирањето на лак за барање во лак за доделување **не** резултира во формирање циклус во графот за алокација на ресурси

Повеќе инстанци од ист тип



- Алгоритам за проверка *дали состојбата е сигурна*
- Да се најде редот во втората матрица чии елементи се помали или еднакви на A (...редот на процесот D)
  - Ако нема таков, следува блокада!
  - Ако има, изврши го тој процес и врати ги неговите ресурси на A
- Повторувај го чекорот додека не се маркирани сите процеси
  - Ако сите се извршат, состојбата била сигурна
  - Ако има блокада, состојбата НЕ била сигурна

Банкаров алгоритам – формален метод

- Повеќе инстанци од ист тип
- Секој процес мора однапред да знае колку максимум инстанци од секој ресурс му требаат
- Кога процесот бара ресурс, можеби ќе мора да чека
- Кога процесот ќе ги добие сите ресурси што му требаат, мора да ги врати за конечно време

Структури податоци за банкаровиот алгоритам

Нека n = број на процеси и m = број на типови ресурси

- *Available*: Вектор со должина m. Ако *available[j]* = k, тогаш постојат k слободни инстанци на ресурсот од тип R_j
- *Max*: n x m матрица. Ако *Max* [*i,j*] = k, тогаш на процесот P_i може да му требаат најмногу k инстанци од ресурсот од тип R_j.
- *Allocation*: n x m матрица. Ако Allocation[*i,j*] = k тогаш на P_i се тековно алоцирани k инстанци од R_j.
- *Need*: n x m матрица. Ако *Need*[*i,j*] = k, тогаш на P_i може да му требаат уште k инстанци од R_j за да се заврши

Need [*i,j*] = *Max*[*i,j*] – *Allocation* [*i,j*]

Алгоритам за барање ресурси за процесот P_i

Request = вектор на побарувања за процесот P_i. Ако $Request_i[j] = k$ тогаш процесот P_i бара k инстанци од ресурсот R_j.

1. Ако $Request_i \leq Need_i$ оди на чекор 2. Инаку, пријави грешка бидејќи процесот ги достигна максималните барања
2. Ако $Request_i \leq Available$, оди на чекор 3. Инаку P_i мора да чека, бидејќи ресурсите не му се достапни.
3. Замисли дека му ги додели бараните ресурси на P_i модифицирајќи ја состојбата според:

$$Available = Available - Request;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

- *ако е сигурна ⇒ додели ресурси на Pi.*
- *ако е несигурна ⇒ Pi мора да чека, врати ја состојбата од пред да ги направиш измените*

Пример за банкаров алгоритам

- 5 процеси P_0 до P_4;
  3 типови ресурси:
  A (10 инстанци), B (5 инстанци), и C (7 инстанци).
- Во време T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Пример за банкаров алгоритам (2)

- Содржината на матрицата *Need* е *Max* – *Allocation*.

	Need	*Available*
	A B C	*A B C*
P_0	7 4 3	3 3 2
P_1	1 2 2	
P_2	6 0 0	
P_3	0 1 1	
P_4	4 3 1	

- Системот е во сигурна состојба бидејќи секвенцата $< P_1, P_3, P_4, P_2, P_0 >$ го задоволува условот за сигурност

Корисност?

- Теоретски убаво, но практично бескорисно
  - Како однапред да го дознаеме бројот на ресурси по процес?
  - Бројот на активни процеси **динамички се менува**, нови корисници се вклучуваат и исклучуваат!
  - Некој ресурс може да биде недостапен (расипување и сл.)!

- Во практиката, многу малку постоечки системи го користат банкаровиот алгоритам за избегнување блокади!

3. Дозволи, детектирај и реагирај

- Дозволи ја блокадата
- Анализирај ја ситуацијата
- Одбери процес – жртва и отстрани го
- Направи надоместување (recovery)

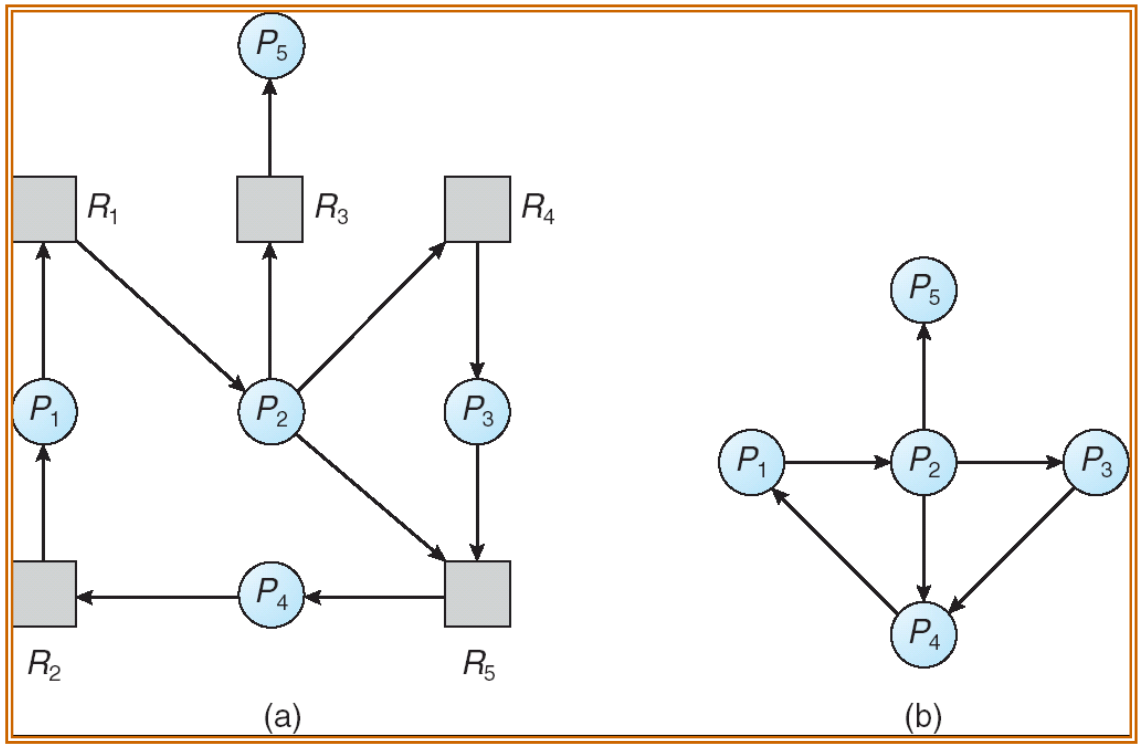
- Прашања:
  - Како да препознаеме дека се случува блокада?
  - Потребен е формален алгоритам (детектирање циклуси во ориентиран граф, матричен алгоритам)
- Како да ја одбереме жртвата?
  - Да се препознае кој од процесите се блокира...

Една инстанца од секој ресурс

- Направи *wait-for* граф на чекање
  - Јазлите се процеси.
  - $P_i \rightarrow P_j$ ако P_i го чека P_j.
- Периодично стартувај алгоритам кој бара циклуси низ графот. Ако има циклус, има блокада.
- Алгоритам кој детектира циклус во граф има комплексност од n^2 операции, каде n е бројот на лакови во графот

Граф на алокација и кореспондирачки граф на чекање

- Пример:



граф на алокација на ресурси

кореспондирачки wait-for граф

Неколку инстанци од ист ресурс

- *Available*: Вектор со должина *m* означува колкав број слободни ресурси има од секој тип
- *Allocation*: *n* x *m* матрица која дефинира колкав број ресурси од ист тип тековно се алоцирани на секој процес
- *Request*: *n* x *m* матрица која ги означува тековните барања на секој процес. Ако *Request* $[i_j] = k$, тогаш процесот P_i бара уште *k* инстанци од ресурсот од тип R_j

Алгоритам за детекција

1. Нека *Work* и *Finish* се вектори со должина *m* и *n*, соодветно. Иницијализирај:
   a) *Work* = *Available*
   b) за *i* = 1,2, …, *n*,
      ако $Allocation_i \neq 0$,
         тогаш
            Finish[i] = false;
         инаку,
            Finish[i] = *true*.
2. Најди индекс *i* така да и двата услова важат:
   (a) *Finish*[*i*] == *false*
   (b) $Request_i \leq Work$

   Ако нема таков *i*, оди на чекор4.

Алгоритам за детекција (2)

3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   оди на чекор 2.

4. Ако $Finish[i] ==$ false, за некое i, $1 \leq i \leq n$, тогаш системот е во блокада. Уште повеќе, ако $Finish[i] == false$, тогаш P_i е блокиран.

На алгоритамов му требаат $O(m \times n^2)$ операции да детектира дали системот е во блокада

Пример за алгоритам за детекција

- Пет процеси P_0 до P_4; три типови ресурси A (7 инстанци), B (2 инстанци), и C (6 инстанци).
- Во време T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	**0 1 0**
P_1	2 0 0	2 0 2	**3 1 3**
P_2	3 0 3	0 0 0	**5 2 4**
P_3	2 1 1	1 0 0	**7 2 4**
P_4	0 0 2	0 0 2	**7 2 6**

- Секвенцата $<P_0, P_2, P_3, P_1, P_4>$ ќе резултира со $Finish[i] =$ true за сите i.

Пример (2)

- P_2 бара дополнителна инстанца од типот C.

	Request		
	A	*B*	*C*
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Состојба на системот?
  - Може да се добијат ресурсите кои ги држи P_0, но нема доволно ресурси да се исполнат барањата на другите процеси.
  - Блокада, која се состои од процесите P_1, P_2, P_3, и P_4.

Користење на алгоритам за детекција

- Кога и колку често се вклучува зависи од:
  - Колку често се случува блокада?
  - Колку процеси треба да се вратат назад (roll back)?
    - по еден за секој откачен циклус
- Ако алгоритамот за детекција се вклучува произволно, може да има повеќе циклуси во графот и да не можеме да одлучиме кој од нив ја создал блокадата

Што да се направи?

- Привремено одземи ресурси на некој процес, па после да му се врати (може при пакетни обработки, зависно е од ресурсот и ретко е можно)
- Врати некој процес наназад (се запишуваат на диск состојби на процеси од кои места тие може да се реактивираат)
- Убиј еден или повеќе процеси (што се во блокада или не)
  - Добро е да се убие процес кој нема проблеми да се пушти повторно
  - (пр. компајлирање, но не ажурурање во база на податоци – ако на пр. Процесот додава 1 на некој запси и ако потоа тој се убие, при повторното стартување, ќе додаде уште 1, што е некоректно)

Повраток од блокада со терминирање процеси

- Прекини ги сите процеси во блокада
- Прекинувај еден по еден процес сè додека блокадата не е елиминирана
- По кој редослед да се одбере кој процес да се прекине?
  - Приоритет на процеси
  - Колку долго даден процес се извршувал и уште колку има до крајот
  - Ресурси кои процесот ги искористил
  - Ресурси кои сеуште му требаат на процесот
  - Колку процеси ќе треба да бидат терминирани
  - Дали процесот е интерактивен или пакетен (batch)?

Повраток од блокада со празнење ресурси

- Селектирање жртва – минимизирање цена
- Rollback – врати се во некоја претходна сигурна состојба, рестартирај го процесот оттаму
- Изгладнување – некој процес може секојпат да биде биран за жртва, па вклучи го бројот на враќање на процесот (rollback) како фактор за цена

4. Не прави ништо!

- Наједноставен пристап
- Алгоритам на ној (глава во песок)
  - Колку често се јавува блокадата?
  - Математичар наспроти инженер!

- Многу системи потенцијално може да страдаат од блокади
- Во UNIX
  - потенцијално страда од блокади што не се ни детектираат
  - Табела на процеси е конечна величина
  - Ако fork паѓа бидејќи табелата е полна, треба да се почека случајно време и да се обиде пак

Баланс – точност и удобност

- Во секој ОС табелите се конечни ресурси и тоа води кон проблеми
  - Табели на процеси, конци,
  - Отворени датотеки
  - Swap простор на дискот

- Веројатно секој корисник преферира повремена блокада отколку низа ограничувања во користењето на ресурсите баланс помеѓу точноста и удобноста

Заклучок

- Блокадите се потенцијален проблем за секој систем
- Може да се избегнуваат со водење сметка кои состојби се сигурни (постои низа од настани кои гарантираат дека сите процеси ќе завршат), а кои не се
- Банкаровиот алгоритам избегнува блокади со не дозволување на барањето што ќе го доведе системот во несигурна состојба
- Блокада се избегнува со нумерирање на сите ресурси, т.ш. сите процеси да ги побаруваат стриктно во растечки редослед
- Изгладнувањето може да се избегне со FCFS политика