KONFLIKTET E PROCESEVE (DEADLOCKS)

KAPITULLI 7

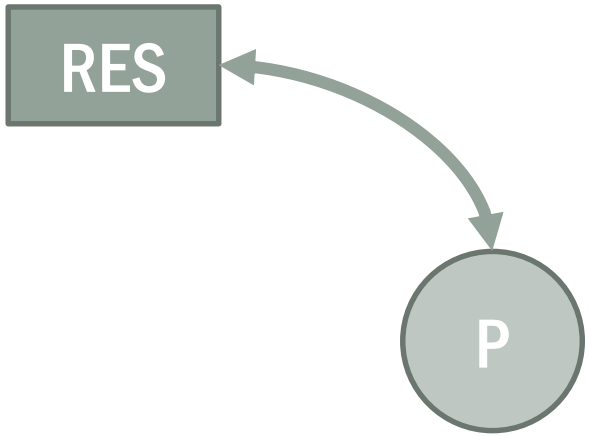
Prof. Ass. Dr. Isak Shabani

Pershkrimi deadlock-ut

- Proceset jane programe ne ekzekutim.
- Per ti perfunduar punet e tyre proceset perdorin resurset, keto resurse mund te jene te cfaredoshme, si falljat, disk drive, printeret etj.
- Ne rastet e multiprogramimit, kur procesori mund te ekzekutoj shume procese ne te njëjtën kohe, eksiton mundësia që shume procese te konkurrojnë për përdorimin e resurseve që janë duke u përdorur nga një proces tjetër.
- Ne këtë rast, këto procese që dëshirojnë te përdorin këto resurse vendosen ne një gjendje pritje, dhe presin deri sa këto resurse te lirohen.
- Nganjëherë lirimi i këtyre resurseve varet nga një proces tjetër i cili është duke pritur që te lirohen disa resurse te tjera nga disa procese te tjera.
- Ne këtë rast krijohet një cikël i proceseve që janë duke pritur për resurset.
- Si pasoj proceset vendosen ne një gjendje konfliktuoze.
- Ne boten e sistemeve operative kjo gjendje njihet si **deadlock**.
- Kur te ndodh deadlock, prceset nuk mbarojne se perfunduar punen, resurset jane te zena per procese te tjera dhe parandalon procese te tjera per startim.
- Jane disa metoda qe te trajtohet deadlock, metodat per: Detektimi, Shmangia, Parandalimi dhe Rikthimi

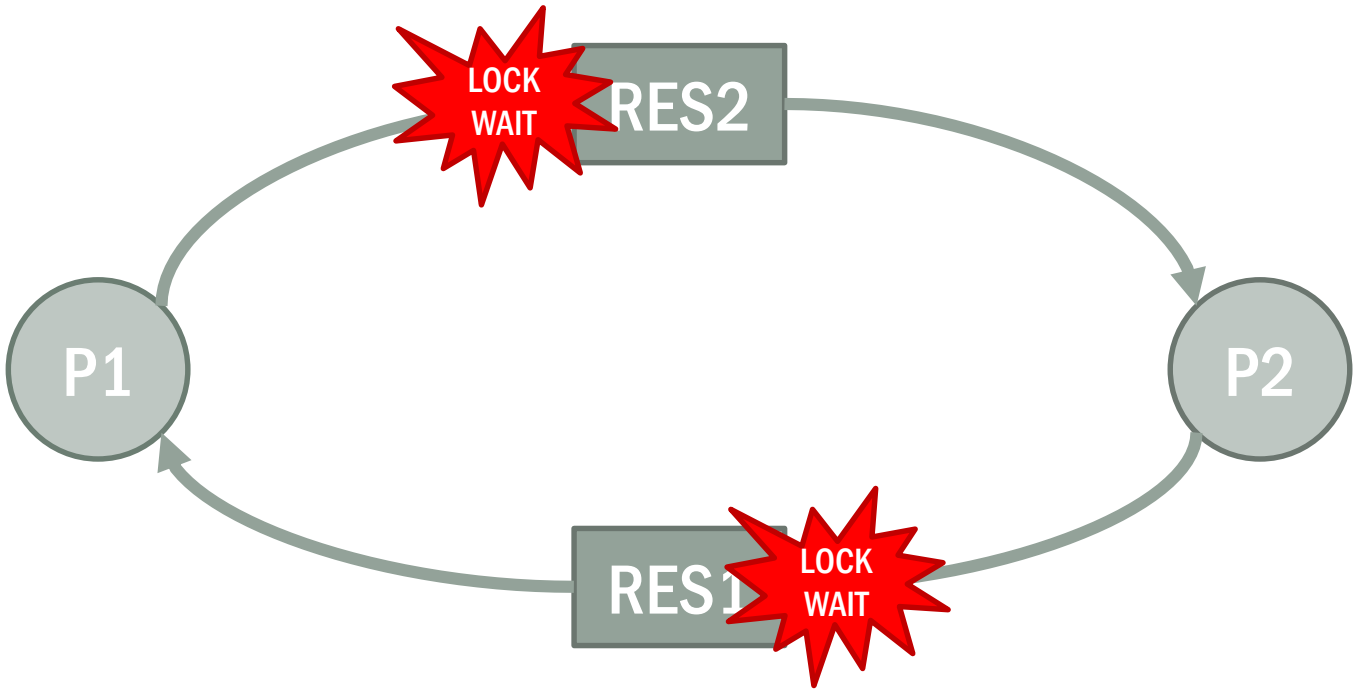
Modeli Sistemit

- Kërkesa e Resurseve
- Përdorimi
- Lëshimi i Resurseve



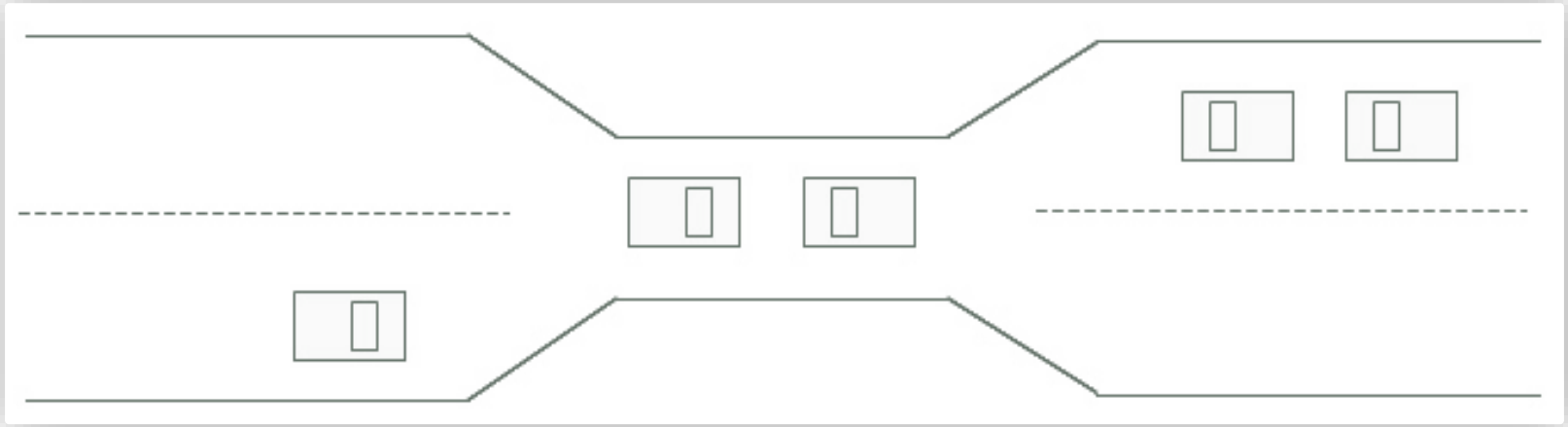

```
request(), release(), open(),
close(), allocate(), free()
```

Si ndodh Deadlock-u në SO



Ndodhi Deadlock-u (Bllokimi)

Shembull ne Jeten Reale



Kalimi i urës me një drejtim

Metodat per trajtimin e deadlockut

Egzistojne tri mënyra :

- Përdorimi i një protokolli për të parandaluar apo shmangur deadlokun, duke siguruar që sistemi kurrë nuk do të futet në gjendje të bllokuar(deadlok).
- Lejimi që sistemi të hyjë në një gjendje të bllokuar, dediktimi dhe shërimi.
- Injoroimi i problemit dhe pretendimi se deadlocku kurrë nuk do ndodhe ne sistem.

Parandalimi i Deadlock-ut

- Kushti i perjashtimeve reciproke

Secili resurse është i caktuar apo ka në dispozicion vetëm një proces gjatë një kohe

- Kushti i mbajes dhe pritjes

Një proces duhet të jetë duke mbajtur ne shfrytezim se paku nje resurs dhe në pritje për së paku një resursi tjeter

- Kushti i jo-parazbrazjes

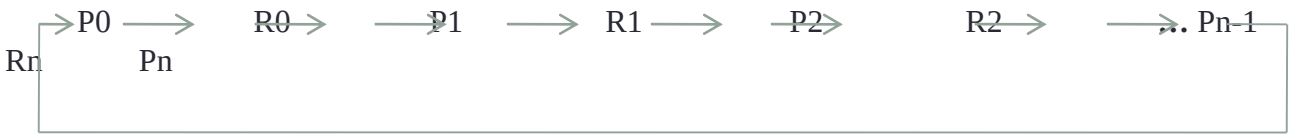
Resurset e dhëna më pare nuk mund të largohen me forcë nga një proces.Ato duhet të lirohen nga vete procesi qe i mbanë.

Vazhdim…

- Kushti i pritjes ne cikel

Cikel i dy apo më shumë proceseve,ku secili është duke pritur për te marrë resurse që mbahen nga procesi tjeter.

R= {R0,R1,R2,..Rn} , P={P0,P1,P2,…Pn}



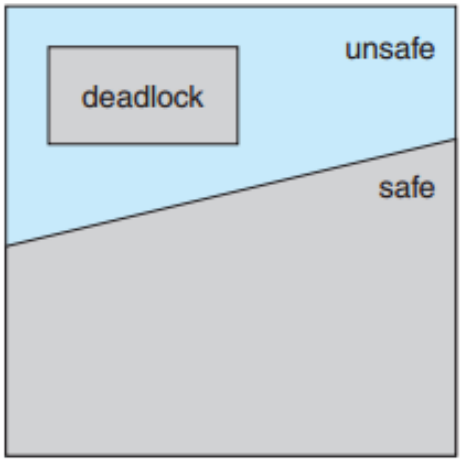
Shmangia e Deadlock-ut

- Algoritmet e parandalimit te deadlockut parandalojne deadlockun duke e kufizuar parashtrimin e kerkesave.

- *Modeli me I thjeshte dhe I perkohshem:*
  kerkon qe cdo process te deklaroje numrim maksimal te resurseve te cdo tipi te cilat mund te nevojiten

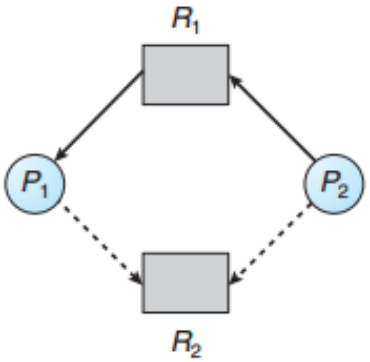
Vazhdim…

- **Gjendja e sigurte** - nese sistemi mund te alokoje resurse ne cdo proces (deri ne maksiumin) dhe ende te shmange deadlockun.
- **Gjendja e pasigurte -** mund te dergoje tek deadlock, mirepo jot e gjitha gjendjet e pasigurta jane gjendje te deadlockut.



Vazhdim…

Grafi I alokimit te resurseve



Algoritmi i Bankerit

- Grafi i alokimit te resurseve nuk aplikohet ne sistemin e alokimit te resurseve me instance te shumefishta te cdo tipi te resursit.

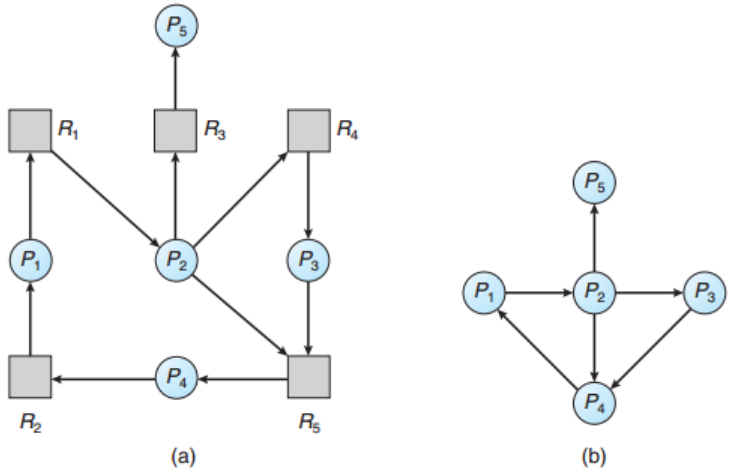
Vazhdim…

- **Available-** Nese *Available[j]* eshte e barabarte me *k,* atehere jane te disponueshme *k* instance te tipit te resursit Rj
- **Max. -** Nese *Max[i][j]* eshte e barabarte me *k,* atehere procesi Pi mund te kerkoje me se shumti *k* instance te resursit te tipit Rj
- **Allocation-** Nese *Alokimi[i][j]* eshte I barabarte me *k,* atehere procesi Pi momentalisht alokon *k* instance te tipit te resursit Rj.
- **Need-** Nese *Need[i][j]* eshte e barabarte me *k,* atehere procesit Pi mund t'I nevojiten *k* instance me shume te tipit te resursit Rj per te kompletuar detyren. Te verejme se *Need[i][j]* eshte e barabarte me *Max[i][j]- Allocation[i][j]*

Detektimi i deadlock-ut

- Lejimi qe sistemi te hyje ne nje gjendje te deadlockut
- Perdorimi i algoritmeve te detektimit
- Perdorimi i algoritmeve per rimekembje te sistemit pas deadlokut

Grafi i alokimit te resurseve



(a) Resource-allocation graph. (b) Corresponding wait-for graph.

- Nje resurs per cdo tip
- P_i -> P_j (P_i pret qe P_j te liroje nje resurs qe P_i i nevoitet)
- P_i -> P_j nese kemi dy kende P_i -> R_q dhe R_q -> P_j ateher pritja behet per resursin e R_q

Algoritmi i detektimit

Le te jene *Work* dhe *Finish* vektoret *m* dhe *n* :

(a) *Work* = *Available*

(b) Per $i = 1,2, \ldots, n$, nqs $Allocation_i \neq 0$, then
Finish[i] = false; perndryshe *Finish*[i] = *true*.

1. Gjeje nje index *i* ashtu qe te dyja te vlejne:

   $Finish[i] == false$ dhe $Request_i \leq Work$

   Nese ekziston nje *i* e tille shko te hapi 2.

Work = *Work* + *Allocation*

Finish[*i*] = *true ;* Shko tek hapi 1

2. Nese *Finish*[*i*] == false, per ndonje *i*, $1 \leq i \leq n$, sistemi eshte ne gjendje deadlocku

Shembull i detektimit te Deadlock-ut

- Pese procese P_0 deri tek P_4; Tri lloje te resurseve: A (7 instanca), *B* (2 instanca), and *C* (6 instanca).
- Ne nje kohe T_0:

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

- Sekuenca<P_0, P_2, P_3, P_1, P_4> do rezultoje *Finish*[*i*] = true per cdo *i*

Vazhdim…

- P_2 kerkon nje instance shtese te tipit 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

- Gjendjet e sistemit?
  - Mund te kerkojme burimet e mbajtura nga procesi P0 por qe jane te pamjaftushme per permbushur kerkesat e proceseve te tjera.
  - Deadlocku ekziston dhe eshte i perber nga P1, P2, P3, dhe P4

Rikthimi nga Deadlock-u

- Process Termination
  - Ndërprerja e të gjitha Deadlock procesëve.
  - Ndërprerja e procesëve një nga një deri sa te eleminohet cikli i Deadlock-ut
- Resource Preemption
  - Selecting victim
  - Rollback
  - Starvation

Vazhdim

- Egzistojne 2 opcione per ta thyer nje Deadlock :
  -Duke nderprere nje apo me shume procese per ta thyer pritjen qarkore(Process Termination)
- -Duke parandaluar disa burime nga nje ose me shume procese te Deadlock-ut.(Resource Preemption)
- **PROCESS TERMINATION** (Procesi I Perfundimit)
- Procesi I perfundimit eshte nje metode per tu rikthyer nga deadlock-u. Per procesin e perfundimit perdorim 2 metoda , te cilat jane :
- **1) <u>Nderprerja e te gjitha Deadlocked proceseve :</u>**
- -Nenkupton lirimin e d te gjitha proceseve ne gjendjen e deadlock-ut, dhe fillimin e shperndarjes nga pika e fillimit. Eshte nje metode e mire dhe e shtrenjte.
- **2) <u>Nderprerja e proceseve nje nga nje deri sa te eleminohet cikli I deadlock-ut :</u>**
- -Ne kete metode se pari nderprehet nje prej proceseve ne gjendjen e deadlock-ut, dhe shperndan burimet e disa prej proceseve ne gjendjen e deadlock-ut dhe pastaj kontrollon se a eshte thyer deadlock-u apo jo . Nese jo , nderprehet procesi tjeter nga gjendja e deadlock-ut. Ky process vazhdon deri sa te rikthehemi nga deadlock-u. Edhe kjo metode eshte po ashtu e shtrenjte , por ne krahasim me te paren eshte me e mire.

Vazhdim..

- **RESOURCE PREEMPTION** (Parandalimi I burimeve)
- Per te eleminuar deadlock-at duke perdorur resource preemption (parandalimin e burimeve), duhet t'I parandalojme disa burime te proceseve, dhe keto burime tua japim disa proceseve tjera derisa cikli I deadlock-ut te jete thyer.
- Egzistojne 3 metoda per ti eleminuar deadlock-at duker perdorur "Resource Preemption".
- Keto jane :
  1) Selecting a victim (Selectimi I nje viktime) :
- -Selekton burimin e nje viktime nga gjendja e deadlock-ut, dhe e parandalon ate.
- **2)Rollback**
- -Nese burimi nga nje process eshte parandaluar , c'ka duhet te behet me ate process. Procesi duhet te kthehet mbrapa tek nje gjendje e sigurte, dhe te restartohet nga ajo gjendje .
- **3) Starvation**
- -Duhet te garantohet qe burimi nuk do te parandalohet gjithmone nga I njejti process per ti shmangur problemet e "Starvation".
-