Nödvändiga villkor för baklås

1. **Mutex**: bara en process i taget kan använda minst en resurs.
2. **No preemption:** bara processen själv kan släppa en resurs.
3. **Circular wait:** några processer väntar cirkulärt på varandra.
4. **Hold and wait:** minst en process håller minst en resurs och väntar på fler resurser (som hålls av andra processer).

Hur hantera baklås?

1. Se till att baklås inte uppstår.
2. Upptäck baklås och återhämta systemet.
3. Ignorera baklås.

Hur hantera baklås?

1. Se till att baklås inte uppstår.
   a) Deadlock prevention: se till att ett av villkoren aldrig håller, **eller**
   b) Deadlock avoidance: mha mer information om processerna, allokera resurser pss att deadlock inte uppkommer.
2. Upptäck baklås och återhämta systemet.
   1. Deadlock detection (upptäckt) **och**
   2. Deadlock recovery (återhämtning)
3. Ignorera baklås.

1a. Deadlock prevention

Undvik att uppfylla (minst) ett av villkoren:

- Mutex: svårt med vissa resurser som inte kan delas (skrivare, CPU, ...), men funkar bra med t.ex. read−only−filer.
- No preemption: två varianter:
  - Om P skulle ha behövt vänta, avallokera alla dess resurser och vänta på alla samtidigt.
  - Om allokerad till ngn som väntar, stjäl resursen! Annars, vänta (och bli kanske själv bestulen).

  OK för CPU, minne etc, men inte för skrivare o dyl.

1a. Deadlock prevention

(undvik att uppfylla ett av villkoren)

- Circular wait:
  - a) inför en total ordning mellan resurserna (som <)
  - b) allokera alltid resurser i stigande ordning; flera instanser av samma typ allokeras <u>samtidigt</u>
- Hold–and–wait: två varianter
  - tillåt bara resursallokering när man inte har några allokerade
  - allokera <u>alla</u> resurser före start

  (resursslöseri, risk för svält)

1b. Deadlock avoidance

- Om OS vet i förväg vilka resurser och i vilken ordning processerna vill allokera kan tilldelningarna ordnas så att deadlock inte uppstår.
- Det räcker att veta maximala antalet av varje resurstyp varje process kan vilja ha.

Kräver mer tillstånd i varje process (PCB).

1b. Deadlock avoidance

Idé: bevara ett *säkert tillstånd* hos systemet; undvik circular wait.

Säkert tillstånd: om man kan ge varje process dess maximala antal resurser (i någon ordning) och fortfarande bevara säkert tillstånd.

En sådan ordning kallas *säker sekvens*. Ex $<P_1,P_2,P_3>$: P_i kan få max res. från tillgängliga plus från P_j där $j<i$.

Om ingen säker sekvens finns: *osäkert tillstånd*, <u>kan</u> leda till baklås.

2.1. Deadlock detection

- En instans av varje resurs: skapa wait–for–graf, leta efter cykel – då finns baklås!
  - Wait–for–grafen måste hållas uppdaterad
  - att leta cykler i en graf tar tid proportionell mot n^2, där n är antalet noder i grafen (antalet processer)
- Flera instanser: använd variant av Banker's algoritm.

Resurskrävande!

2.2. Deadlock recovery

- Be Operatören fixa själv...
- Återhämta automatiskt: två sätt:
  - Döda processer:
    - alla baklåsta: enkelt, men dyrt (bortkastat arbete)
    - en efter en tills inget baklås finns kvar: dyrt, måste kolla ofta; svårt att bestämma ordningen
  - Preempta resurser: avallokera resurser och ge till andra tills baklåset löst
    - svårt att välja offer
    - vad gör med offret? Döda? Backa? Mycket info behövs
    - svält ska undvikas – inte samma offer varje gång

Deadlock–hantering

Att kombinera metoder är naturligtvis bäst! Klassindela resurserna, ordna klasserna m. total ordning, använd bästa metoden för varje klass.

Exempel:

- OS–interna resurser: prevention av cirkulär väntan mha total ordning
- Minne: prevention genom preemption – använd virtuellt minne och swapping
- Externa devices, filer: avoidance mha extra information

3. Enklaste metoden

Ignorera baklås!

- Motiv: de uppstår sällan och kostar mer än det smakar att förhindra, undvika, eller upptäcka.
- Vad händer?
  - systemet går trögare och trögare
  - liten grupp av baklåsta processer kan växa tills systemet är oanvändbart och måste startas om
- Kan inte användas t.ex. i säkerhetskritiska system
  - som banker, processövervakning etc