

---

# Operativsystem

Skedulering och deadlocks  
(2.5 , 3 i boken)

# Skedulering

---

- Väljer bland de exekverbara processerna (status redo) och allokerar CPU till denna process
- Skedulering sker
  1. När en process skapas (skall föräldern eller avkomlingen köras först)
  2. När en process avslutas (en ny process måste nu väljas)
  3. När en process blockeras (I/O, semafor, annan orsak)
  4. Vid ett avbrott (sannolikt blir någon process redo att köras. Notera klock-avbrottet!)

# När behövs skedulering?

---

- Persondatorer
  - Kontorsanvändning - Skeduleringen spelar inte någon större roll, en användare märker knappast i vilken ordning processer blir utförda
  - Multimedia – dålig skedulering leder till ”tröghet” i systemet
- Server
  - Bra skedulering kan vara mycket relevant
- Realtidssystem
  - Skedulering kan vara det som avgör om systemet överhuvudtaget är användbart

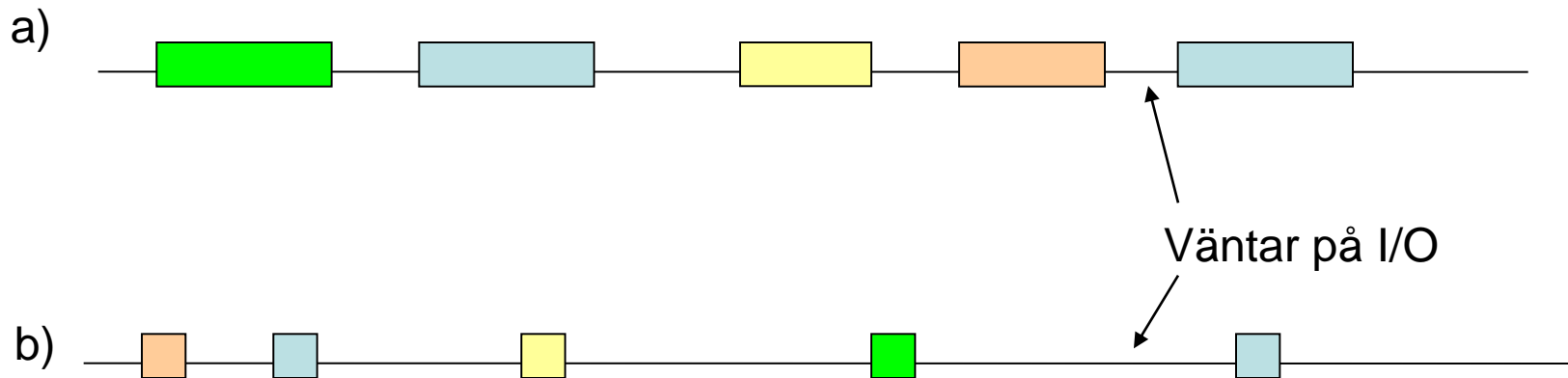
# Preemptive vs. nonpreemptive

---

- Non-preemptive skedulering (eller 'co-operative')
  - En process exekveras tills
    - Den blockeras, t.ex. väntar på I/O
    - Den frivilligt överlåter CPU:n
    - Klock-avbrottet förorskar EJ omskedulering
  - Exempel: MS-DOS-baserade (Win3.11, Win95/98/Millennium; fast där ej processer i egentlig mening)
- Preemptive skedulering
  - En process kan endast exekvera en fixerad maxtid åt gången
    - Om processen fortfarande exekveras vid slutet av sin tidskvanta, väljer skeduleraren en annan process
    - “Normala” OS: WinNT baserade / Linux

# Karakteristika för process

---



a: Beräkningsintensiv

b: I/O-intensiv

# Val av tidskvantum

---



$$\text{Nyttjandegrad} = 1 - (\text{tid för kontextbyte} / \text{tidskvantum})$$

# Skeduleringskriterier

---

- Utnyttjandegrad för CPU – CPU:n skall hållas i arbete
- Genomströming – antal processer som blir färdiga per tidsenhet
- Väntetid – den tid en process blir väntande i status redo
- Responstid – tid mellan det att en ”förfrågan” görs tills det att det första ”svaret” skapas
- Deadlines – då något senast måste utföras

# Skeduleringskriterier

---

- Generellt
  - Rättvisa, policy, balans
- Batch-system
  - Genomströmning, turnaround, CPU utnyttjandegrad
- Interaktiva system
  - Responstid, proportionalitet
- Realtidssystem
  - Deadlines, förutsägbarhet



# Skeduleringsalgoritmer

---

- Batch

- “Först till kvarnen får mala” (First come first served)
- Kortaste jobbet först

$$\min \sum_j t_j^e$$

Dvs minimerar summan av tider då jobben är färdiga  $t_j^e$

- Kortast återstående till näst
  - Men: Hur veta vilket som är det kortaste återstående

# Skeduleringsalgoritmer (2)

---

- Interaktiva
  - Round-robin – alla processer exekveras i tur och ordning
  - Priority – processer med högre prioritet kör först
  - Multiple queues – låter beräkningsintensiva processer efterhand få längre kvanta (men mer sällan)
  - Shortest process next – likt i batchskedulering
  - Guaranteed scheduling – varje användare garanteras en andel av CPU-tiden
  - Lottery scheduling – nästa process väljs randomiserat

# Skeduleringsalgoritmer (3)

---

- Realtid

- Skedulerbarhet

$N$  händelser, händelse  $i$  förekommer med perioden  $P_i$  och behöver  $C_i$  tidsenheter. Systemet är skedulerbart om

$$\sum_{i=1}^N \frac{C_i}{P_i} \leq 1$$

- Hard real time – deadlines måste till varje pris hållas (t.ex. krockkudde i bil)
    - Soft real time – det är tolererbart att nu och då missa en deadline (t.ex. en mjukvaruvideospelare)

# Exempel: Skedulering i Linux version 2.4

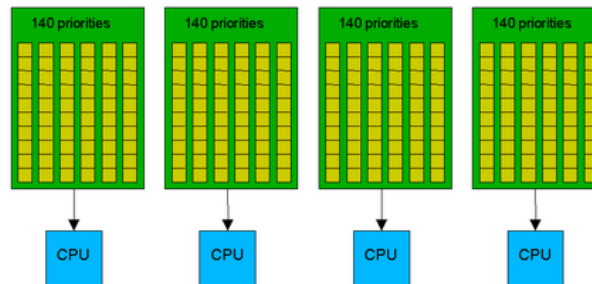
---

- CPU-tiden indelas i epoker
- Varje process har ett tidskvantum som för varje process beräknas i början av epoken
  - Om en process använder sitt tidskvantum, kan denna process inte längre skeduleras i denna epok
- En epok avslutas då alla processer i tillståndet *redo* har använt sitt kvantum
  - Nästa tidskvantum för varje process omräknas nu enligt
$$\text{quantum} = (\text{quantum} \gg 1) + (20 - \text{nice}) / 4 + 1:$$
$$\text{nice} = -20 \text{ (högsta)} \quad - \quad +19 \text{ (lägsta)}$$
  - Om en process har konsumerat sitt tidskvantum, ges processen ett nytt kvantum som motsvarar dess *nice* värde
- OBS: En tråd är en lätt process, dvs. skeduleras som en process

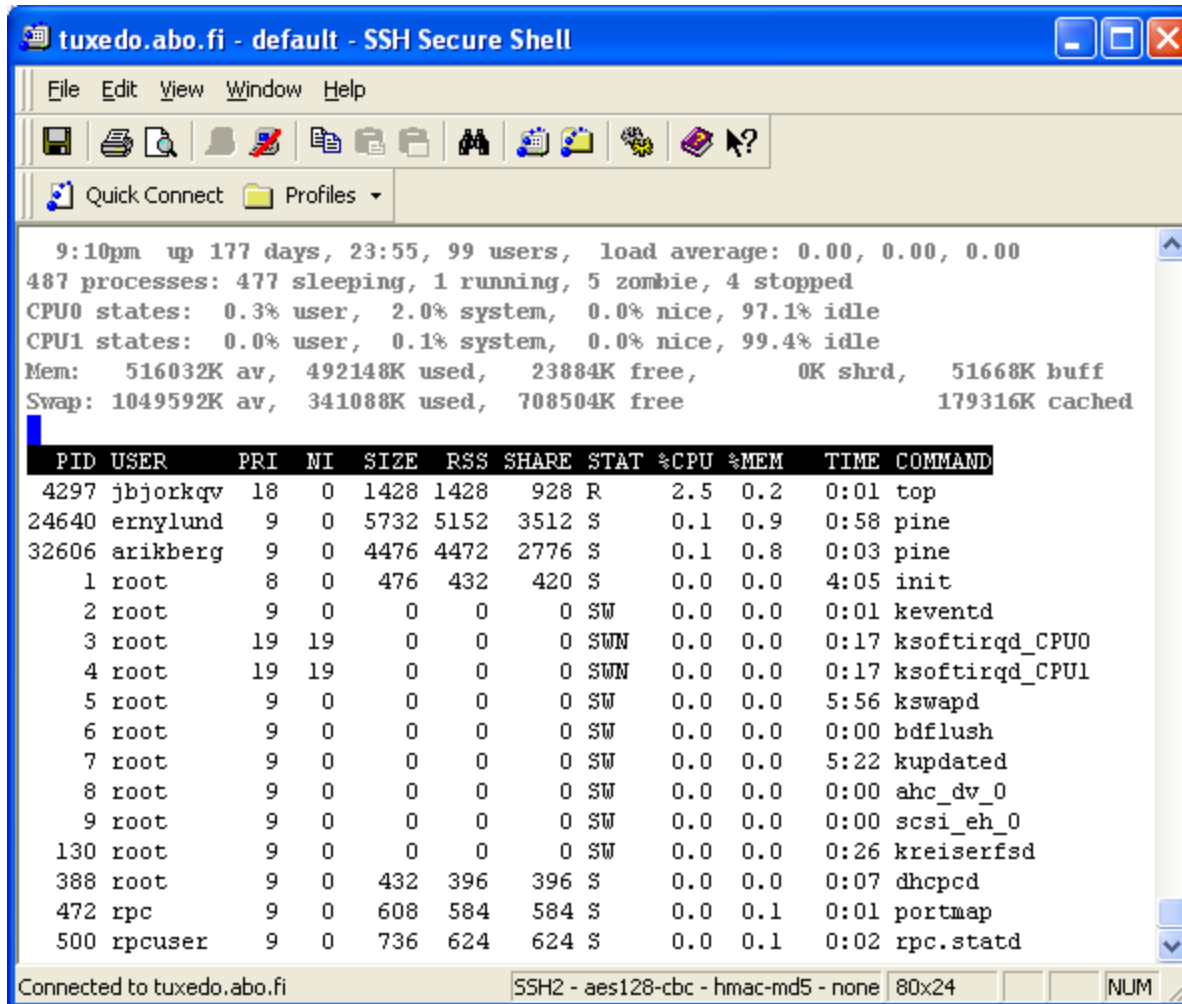
# Skedulering i Linux (2)

---

- I Linux 2.6 (våren 2004) infördes ny skedulering:
  - Målsättning: en  $O(1)$  skedulerare
    - Implementerad via prioritetssköer
- Bättre responstider (bättre byggda möjligheter att avbryta pågående exekveringsstigar)
- Realtidsegenskaperna väsentligt förbättrade



# Processlista - Linux



The screenshot shows a terminal window titled "tuxedo.abo.fi - default - SSH Secure Shell". The terminal output displays system statistics and a process list. The statistics include system uptime, user count, load averages, process counts, CPU usage for two CPUs, and memory/swap usage. The process list is a table with columns for PID, USER, PRI, NI, SIZE, RSS, SHARE, STAT, %CPU, %MEM, TIME, and COMMAND.

```
9:10pm up 177 days, 23:55, 99 users, load average: 0.00, 0.00, 0.00
487 processes: 477 sleeping, 1 running, 5 zombie, 4 stopped
CPU0 states: 0.3% user, 2.0% system, 0.0% nice, 97.1% idle
CPU1 states: 0.0% user, 0.1% system, 0.0% nice, 99.4% idle
Mem: 516032K av, 492148K used, 23884K free, 0K shrd, 51668K buff
Swap: 1049592K av, 341088K used, 708504K free 179316K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
4297	jbjorkqv	18	0	1428	1428	928	R	2.5	0.2	0:01	top
24640	ernylund	9	0	5732	5152	3512	S	0.1	0.9	0:58	pine
32606	arikberg	9	0	4476	4472	2776	S	0.1	0.8	0:03	pine
1	root	8	0	476	432	420	S	0.0	0.0	4:05	init
2	root	9	0	0	0	0	SW	0.0	0.0	0:01	keventd
3	root	19	19	0	0	0	SWN	0.0	0.0	0:17	ksoftirqd_CPU0
4	root	19	19	0	0	0	SWN	0.0	0.0	0:17	ksoftirqd_CPU1
5	root	9	0	0	0	0	SW	0.0	0.0	5:56	kswapd
6	root	9	0	0	0	0	SW	0.0	0.0	0:00	bdflood
7	root	9	0	0	0	0	SW	0.0	0.0	5:22	kupdated
8	root	9	0	0	0	0	SW	0.0	0.0	0:00	ahc_dv_0
9	root	9	0	0	0	0	SW	0.0	0.0	0:00	scsi_eh_0
130	root	9	0	0	0	0	SW	0.0	0.0	0:26	kreiserfsd
388	root	9	0	432	396	396	S	0.0	0.0	0:07	dhcpcd
472	rpc	9	0	608	584	584	S	0.0	0.1	0:01	portmap
500	rpcuser	9	0	736	624	624	S	0.0	0.1	0:02	rpc.statd

Connected to tuxedo.abo.fi SSH2 - aes128-cbc - hmac-md5 - none 80x24 NUM

# Skedulering WinNT

---

- Skedulering sker alltid per tråd-basis
- Varje tråd associeras med en grundprioritet i intervallet 1-31, där 1-15 (dynamisk prioritet) är för vanliga trådar
- Varje process tilldelas en nuvarande prioritet, som kan vara högre än grundprioritet
  - höjs t.ex. efter en I/O operation
  - då en tråd har väntat på ett synkroniseringobject
- Trådarna placeras enligt nuvarande prioritet i prioritetsköer: trådar med högsta prioritet får exekvera

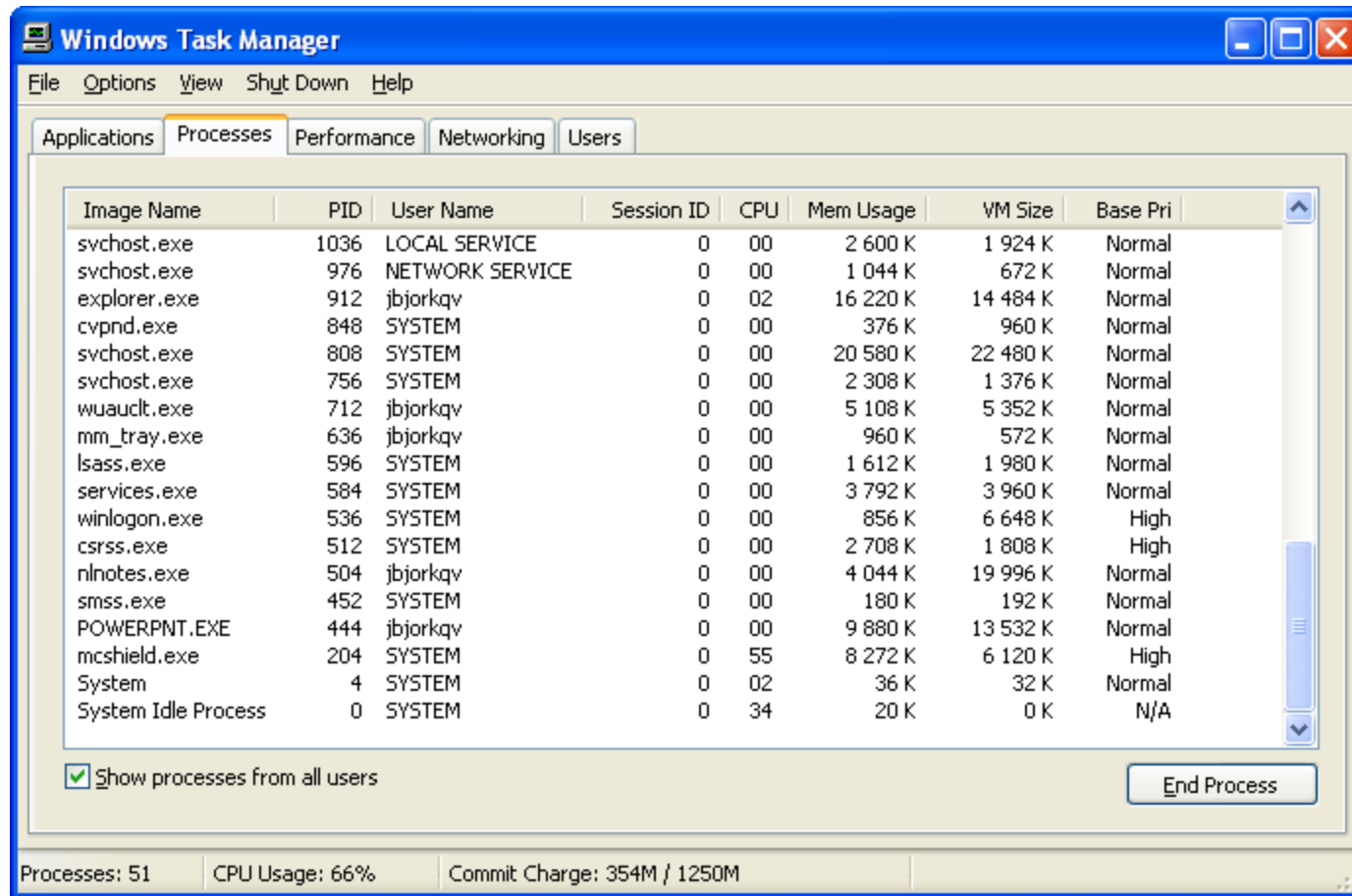
# Prioriteter i Win2000

---

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1



# Processlista - WinXP



Windows Task Manager

File Options View Shut Down Help

Applications Processes Performance Networking Users

Image Name	PID	User Name	Session ID	CPU	Mem Usage	VM Size	Base Pri
svchost.exe	1036	LOCAL SERVICE	0	00	2 600 K	1 924 K	Normal
svchost.exe	976	NETWORK SERVICE	0	00	1 044 K	672 K	Normal
explorer.exe	912	jbjorkqv	0	02	16 220 K	14 484 K	Normal
cvpnd.exe	848	SYSTEM	0	00	376 K	960 K	Normal
svchost.exe	808	SYSTEM	0	00	20 580 K	22 480 K	Normal
svchost.exe	756	SYSTEM	0	00	2 308 K	1 376 K	Normal
wuauclt.exe	712	jbjorkqv	0	00	5 108 K	5 352 K	Normal
mm_tray.exe	636	jbjorkqv	0	00	960 K	572 K	Normal
lsass.exe	596	SYSTEM	0	00	1 612 K	1 980 K	Normal
services.exe	584	SYSTEM	0	00	3 792 K	3 960 K	Normal
winlogon.exe	536	SYSTEM	0	00	856 K	6 648 K	High
csrss.exe	512	SYSTEM	0	00	2 708 K	1 808 K	High
nlnotes.exe	504	jbjorkqv	0	00	4 044 K	19 996 K	Normal
smss.exe	452	SYSTEM	0	00	180 K	192 K	Normal
POWERPNT.EXE	444	jbjorkqv	0	00	9 880 K	13 532 K	Normal
mcshield.exe	204	SYSTEM	0	55	8 272 K	6 120 K	High
System	4	SYSTEM	0	02	36 K	32 K	Normal
System Idle Process	0	SYSTEM	0	34	20 K	0 K	N/A

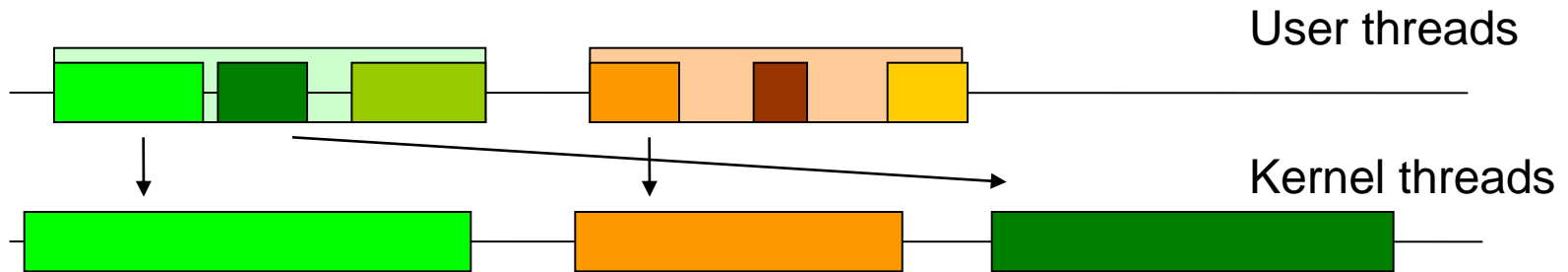
Show processes from all users

End Process

Processes: 51 CPU Usage: 66% Commit Charge: 354M / 1250M

# Skedulering av trådar

---



- User threads
  - Trådar skeduleras inom själva processen
- Kernel threads
  - Tråden ses som det skedulerbara objektet

# Låsning

---

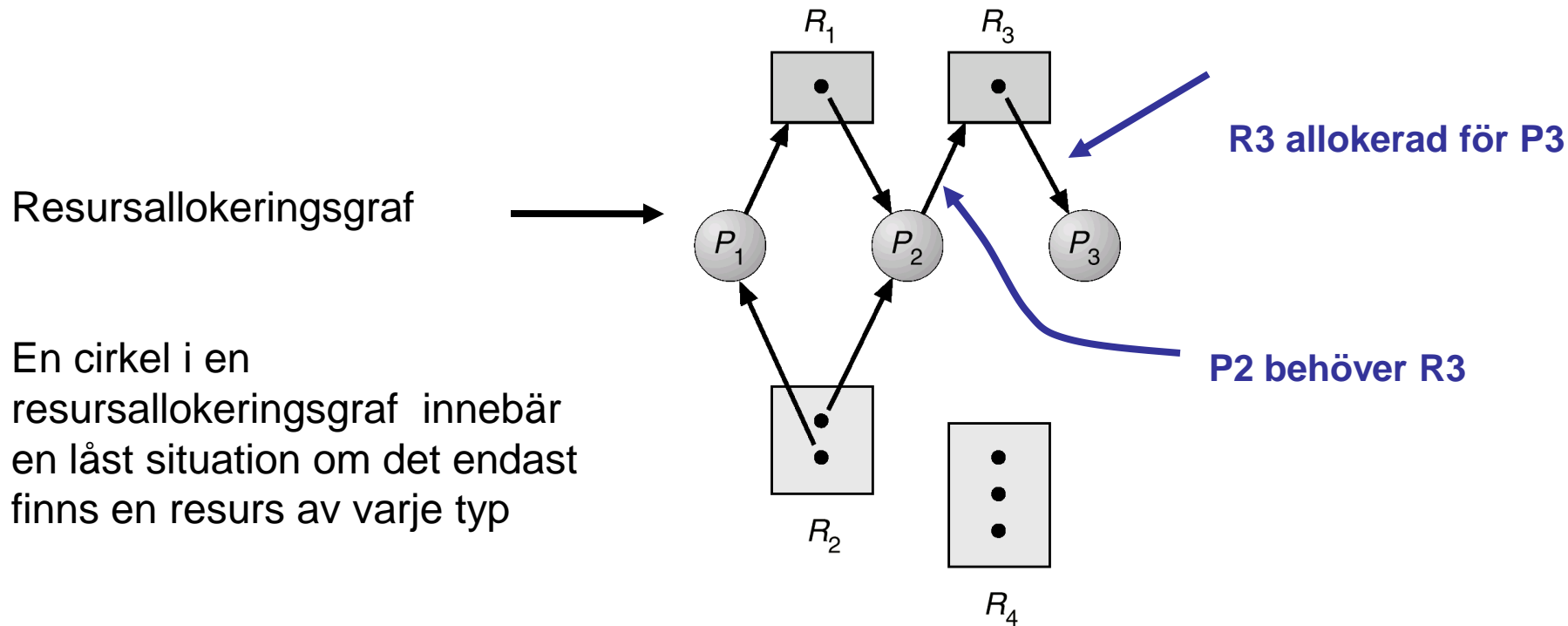
- **Definition:** *En mängd processer är låsta om varje process i mängden väntar på en händelse som endast en annan process i mängden kan generera*
- **Ex. gatukorsning med bilar från fyra håll:**  
samtliga bilar väntar på att bilen från höger skall köra först
  - typiska ”skolexempel”: Två processer: bägge behöver både printer och CD-rom samtidigt. Vid ett tillfälle har den ena processen allokerat printern, den andra CD-rommen. Bägge väntar på den resurs den andra processen har allokerat.

# Villkor för låsning

---

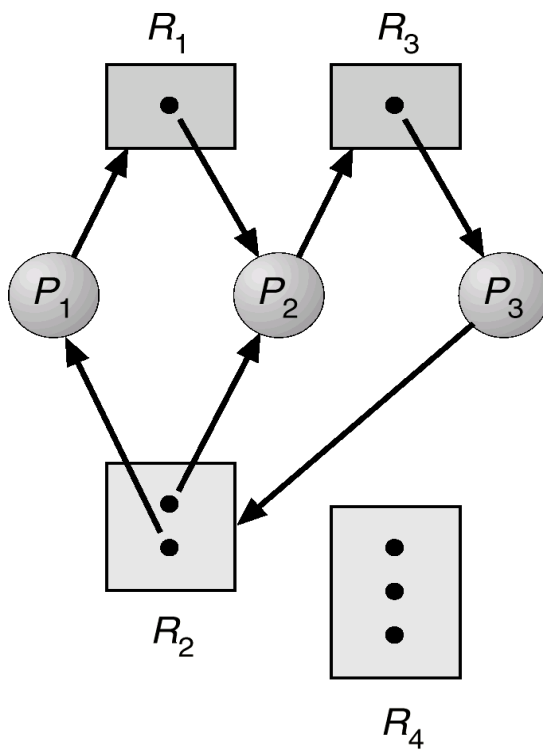
1. Ömsesidig uteslutning (Mutal exclusion )
  - Endast en process åt gången kan accessera en resurs
2. Hold and wait
  - En process behåller resurser allokerade medan den väntar på andra resurser
3. Ingen preemption (No preemption)
  - Resurser kan endast frigöras frivilligt av den process om allokerat dem
4. Cirkulärt väntande (Circular wait condition)
  - Dessutom måste ett speciellt tillstånd vara gällande: Det måste finnas ett cirkulärt väntande på resurser

# Identifiering

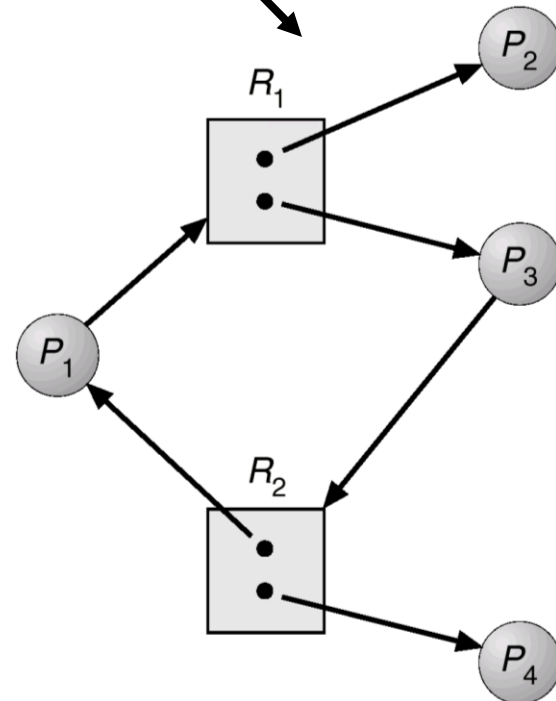


# Identifisering (2)

Resursallkoeringsgraf  
med låsning



Resursallokeringsgraf,  
men ingen låsning



# Låsningar – hur bearbeta?

---

- Strunta i problemet – kanske aldrig blir relevant
- identifiering och reparation – identifiera situationen och åtgärda
- dynamiskt undvikande genom noggran resursallokering
- förhindra, genom att se till att de fyra villkoren inte uppfylls

# Struts”algoritmen”

---

- Stick huvet i sanden och glöm problemet.
- De flesta OS tillämpar åtminstone ställvis - det finns delsystem som potentiellt kan låsa sig, men man antar att sannolikheten är mycket liten



# Reparation

---

- Preemption
  - Ta resurs från process och ge till annan behövande progress
- Rollback
  - Gör det möjligt att gå tillbaka till kontrollpunkt tidigare i exekveringsstigen
- Avsluta process
  - Avsluta den process som håller en resurs

# Förhindrande av låsning

---

- Se till att någon av de fyra villkoren tidigare INTE uppfylls
- Mutual exclusion
  - Man kan använda sig a köer istället (t.ex printerköer, disk-access-köer etc.)
- Hold and wait
  - T.ex. processer måste begära sina resurser från början, om ej lyckas, frigörs alla
- No preemption
  - Vanligen inte en så fungerande strategi
- Circular wait
  - Vanligen genom att resurser alltid måste begäras i en given ordningsföljd

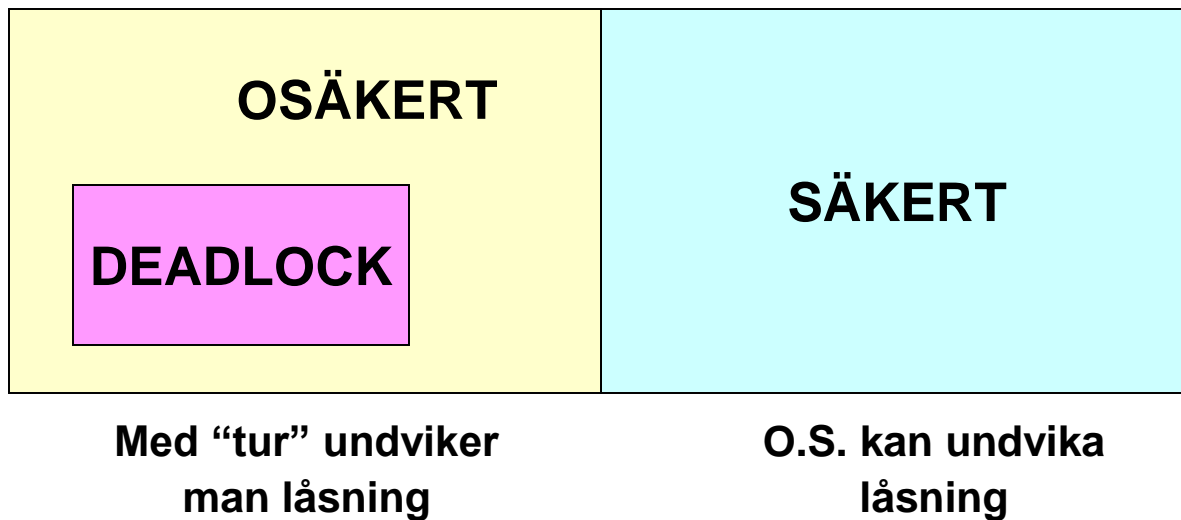
# Undvikande av låsning

---

- Säkra och osäkra tillstånd
  - Säkert tillstånd: Det finns minst ett sätt att fortsätta så att låsning ej uppstår
  - Osäkert tillstånd: Det finns inget säkert sätt att undvika låsning
  - Låsning: Det går inte att gå vidare

# Undvikande av låsning

---



# Undvikande av låsning

---

Varje process har ett maximalt resursbehov (specificerat i början). Vi kan nu avgöra om tillståndet är säkert eller osäkert

## Exempel:

Det finns totalt 12 resurser. För tillfället är allokeringar enligt följande

Process	Max behov	Allokerade	Behov
P0	10	5	5
P1	4	2	2
P2	9	2	7

I detta exempel <p1, p0, p2> är en fungerande sekvens

Vad händer om p2 allokerar ännu en resurs?

# Hur låsning undviks i Linux

---

- Använder sig av ett mycket begränsat antal synkroniseringsobjekt (semaforer)
- Semaforer måste alltid begäras i en given ordning
  - semaforer begärs i stigande ordningsföljd