TDIU11 – Föreläsning 2

Schemaläggning

Filip Strömbäck

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Varför multiprogrammering?

Exempel: Skriv innehållet i en fil på skärmen (`cat`):

Onödigt att CPU (och resten av systemet) måste vänta. Vi nyttjar tiden genom att köra andra trådar!

Processhantering

UNIX (Linux, MacOS)

- `fork()`
- `exec(program, ...)`
- `posix_spawn(program, ...)`
- `waitpid()`

Windows NT

- `CreateProcess(program, ...)`
- `WaitForSingleObject()`

Trådhantering

UNIX (Linux, MacOS)

- `pthread_create()`
- `pthread_join()`
- `pthread_detach()`
- (`clone()`)

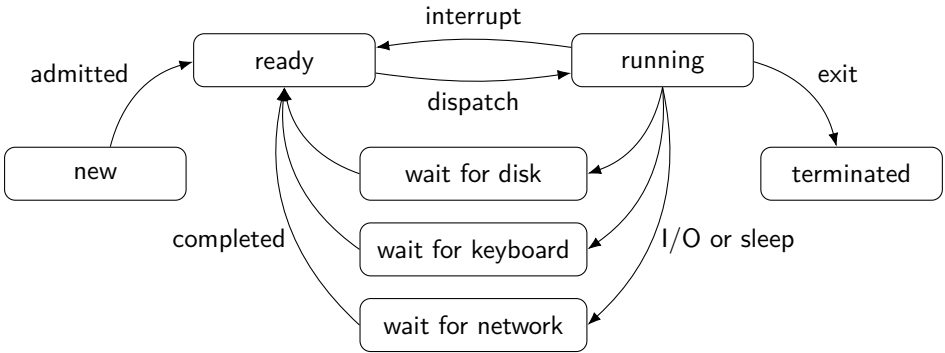
Windows NT

- `CreateThread()`
- `WaitForSingleObject()`

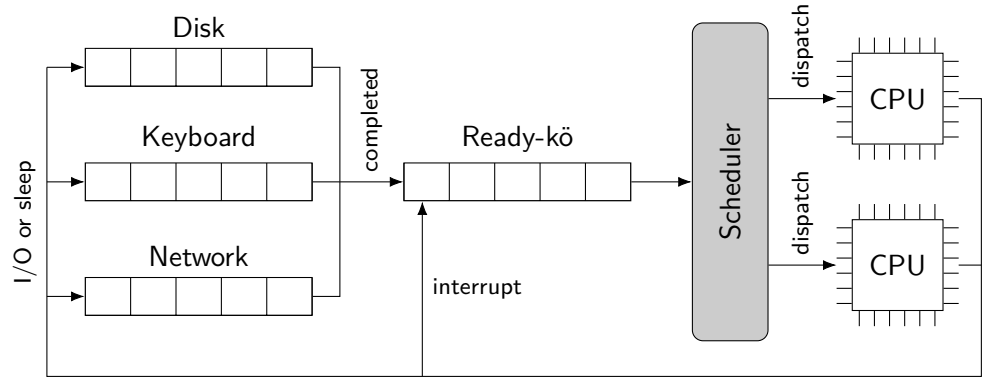
C++

- `std::thread`
- `thread.join()`
- `thread.detach()`

Trådens livstid – Tillståndsdiagram (thread/process-state diagram)



Hantering av trådar i Kernel



Terminologi

Kom ihåg:

Historiskt sett fanns inte trådar. Då fanns bara *processer*. Därför beskrivs många schemaläggare som att de schemalägger *processer* snarare än *trådar*. Idén är densamma. Kursen försöker använda *trådar*.

Undantaget från detta är *job schedulers*. De schemalägger programkörningar, dvs. hela processer.

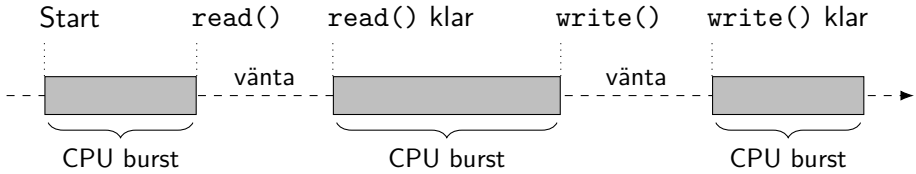
LINKÖPING UNIVERSITY

Typer av schemaläggare

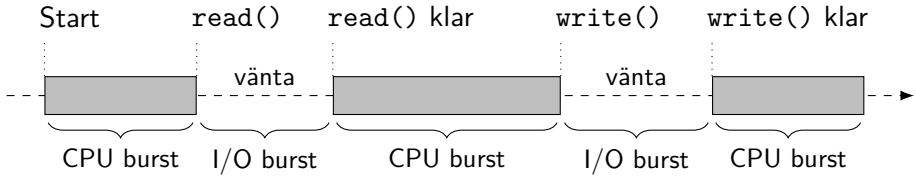
- Korttid: Scheduler (schemaläggare)
  Arbetar *under körning* av processer trådar. Ser till att CPU har något att göra så ofta som möjligt. Måste därför ta beslut snabbt.
- Långtid: Job scheduler
  Körs ibland, och beslutar vilka job (=processer) som ska startas. Ser till att korttids-scheduler har "lagom" mycket att hantera, och att systemet inte får slut på RAM.
  (Detta är vad som används ex.vis på superdatorer tillsammans med en "vanlig" schemaläggare)

Vad behöver schemaläggaren veta?

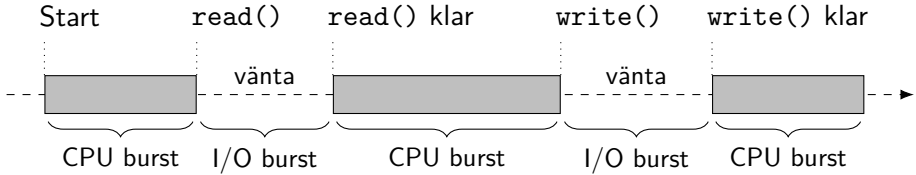
Vad behöver schemaläggaren veta?



Vad behöver schemaläggaren veta?

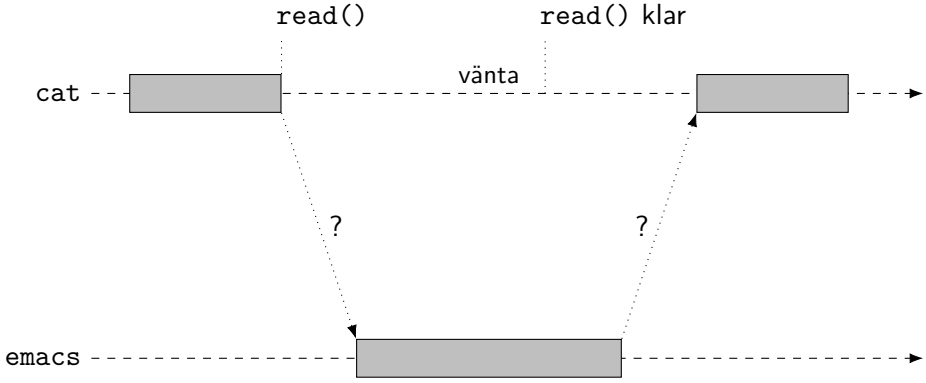


Vad behöver schemaläggaren veta?

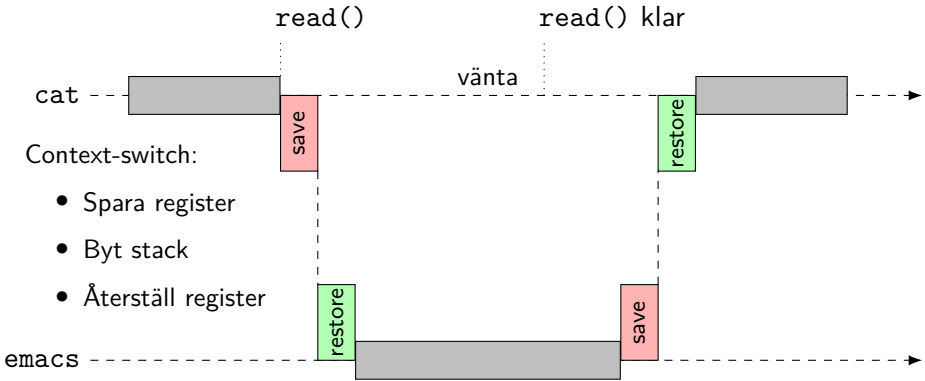


- CPU burst följt av I/O burst
- Vi schemalägger CPU $\Rightarrow$ en CPU burst kan betraktas i isolation
- Tänk: bryr oss bara om *ready-kö*

Hur byter vi mellan olika trådar/processer?



Hur byter vi mellan olika trådar/processer?



Hur jämför vi schemaläggare?

CPU utilization:	Hur stor andel av CPU-tid används (i %)?
Throughput:	Hur många trådar blir klara per tidsenhet?
Turnaround time:	Hur lång tid tar det för en tråd att bli klar?
Waiting time:	Hur lång tid har en tråd fått vänta **ofrivilligt**?
Response time:	Hur lång tid tar det för en tråd att reagera på indata?

Även *average* turnaround time och *average* waiting time.

LINKÖPING UNIVERSITY

FIFO (First In First Out)/FCFS (First Come First Served)

- Låt processer köra i ordningen de lades in i *ready*-kö

Exempel:

Tråd	Start	Burst
A	0	10
B	0	5
C	2	1

SJF (Shortest Job First)

- Kör den process som är kortast först!

Exempel:

Tråd	Start	Burst
A	0	10
B	0	5
C	2	1

Prioritetsbaserad schemaläggning

- Ge varje process en prioritet!
- Problem: risk för *starvation*, kan lösas med *ageing*

Exempel:

Tråd	Start	Burst	Prio
A	0	10	2
B	0	5	3
C	2	1	1

När körs schemaläggaren?

Schemaläggaren är en bit kod som måste köras på CPU.

Vad händer om tråden/processen aldrig terminerar eller väntar?

LINKÖPING UNIVERSITY

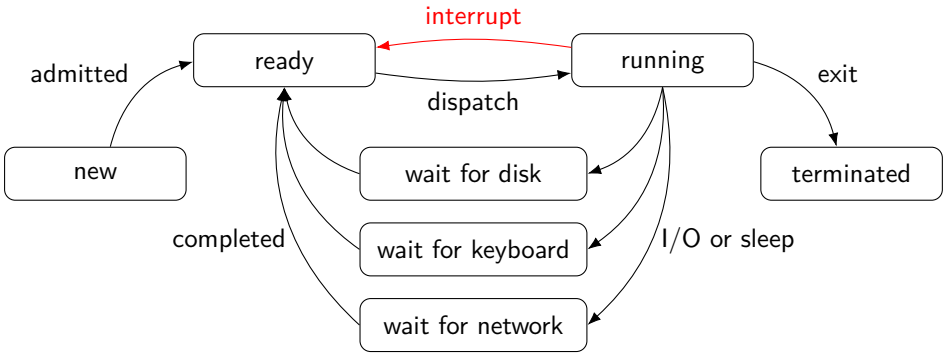
Preemption

Vi tillåter OS att avbryta körande tråd/process

- När *time-quantum* är slut, med hjälp av timeravbrott
- När andra trådar/processer blir redo (nya processer, eller om de väntat klart)

Vi får då *preemptive schedulers*

Tillståndsdiagram



Round-robin = FIFO/FCFS med preemption

- Som FIFO, men om en process kör längre än time-quantum, byt till nästa
- Time quantum måste vara "lagom"
  - Litet $\Rightarrow$ mycket overhead
  - Stort $\Rightarrow$ ingen preemption

Exempel:

Tråd	Start	Burst
A	0	10
B	0	5
C	2	1

Time quantum = 2

Shortest remaining time first = SJF med preemption

- Som SJF, men nya trådar/processer kan avbryta körande process.
- Optimal med avseende på *average waiting time* (vi vet dock sällan på förhand exakt hur lång en *burst* är — vi måste estimera den)

Exempel:

Tråd	Start	Burst
A	0	10
B	0	5
C	2	1

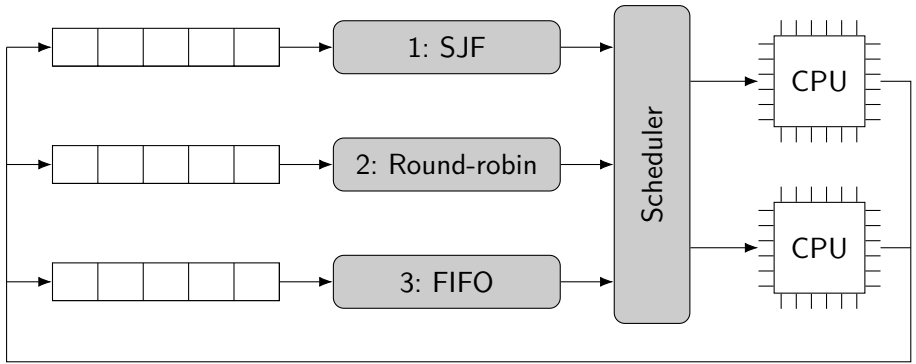
Prioritetsbaserad schemaläggning

- Som i ej-preemptive variant, men nya trådar/processer kan avbryta körande process.
- Om flera med samma prioritet, round-robin mellan dessa.

Exempel:

Tråd	Start	Burst	Prio
A	0	10	2
B	0	5	3
C	2	1	1

Multilevel scheduling



Hur gör ”riktiga” operativsystem?

- Hybridapproach, ofta prioritetsbaserad i grunden
- Preemptive med tidskvantum i storleksordning 10 ms
- Prioritet justeras dynamiskt (ex. Windows ger högre prioritet till aktiv process)
- Se artikeln ”The Linux Scheduler: A Decade of Wasted Cores” länkad från ”Litteratur” på kurshemsidan.

LINKÖPING UNIVERSITY

Exempel på redovisning, 1

Exempelproblem baserat på gårdagens och dagens föreläsning:

Processer som körs i *user-mode* får inte kommunicera direkt med exempelvis hårddisk i systemet. Hur gör OS för att låta processer exempelvis läsa innehållet i en fil på ett säkert sätt? Redogör översiktligt för de steg som krävs för att läsa data från en fil.

Exempel på redovisning, 2

Exempelproblem baserat på schemaläggning:

Schemalägg följande trådar enligt *preemptive SJF*:

	Arrival	**Burst**
T_1	0	4
T_2	2	3
T_3	7	3
T_4	8	2

Time quantum $= 2$

Beräkna också *average waiting time*

Filip Strömbäck

www.liu.se

LINKÖPING UNIVERSITY