



NTNU
Norwegian University of
Science and Technology

Linux prosesser

Donn Morrison

Department of Computer Science

With slides contributed by Geir Maribu

Outline

Review

Linux prosesser

- Prosess-tabellen

- Prosessens tilstander

- Nye prosesser

- Eksempel C kode

Om Linux

- Først utviklet av hvem? Nå?
- Lisens GPL? Hva er det? Hvorfor er det viktig?
- Kjernen til Linux er av hvilken type?
- Hva gjøre `ls -l /etc/*`? Hva med `rm -rf /*tc`?

Outline

Review

Linux prosesser

- Prosess-tabellen

- Prosessens tilstander

- Nye prosesser

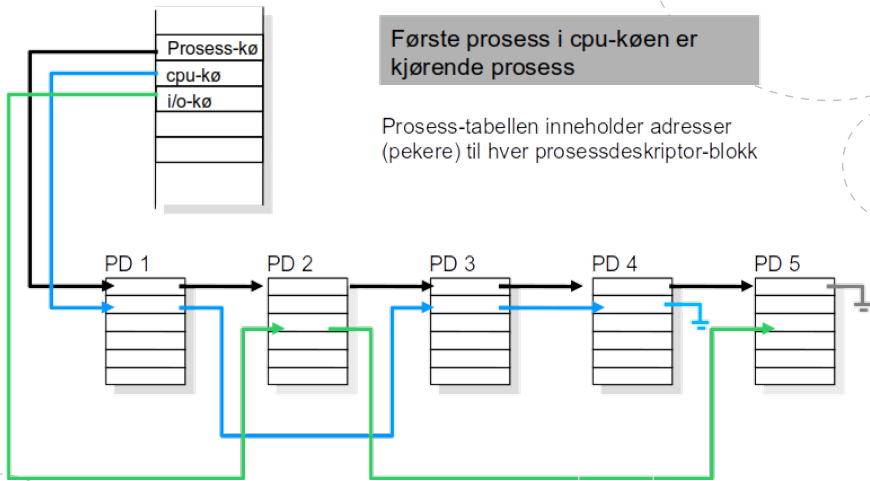
- Eksempel C kode

Prosess-tabellen

- Hver prosess har en PID
- På Linux er PIDer lagret i en prosess-tabell
- Prosess-tabell er felles for hele Linux-systemet
- Bestemt antall plasser
 - `cat /proc/sys/kernel/pid_max`
 - 32768 (32-bit)
 - 4194303 (64-bit)

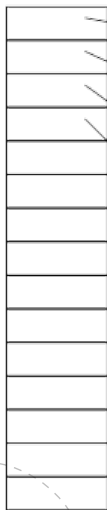
Prosess-tabellen

Prosess-tabell



Prosessskøen

prosess-tabell



PD 1

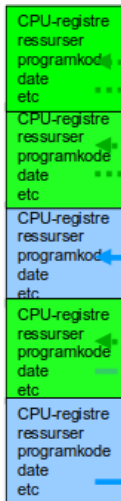
PD 2

PD 3

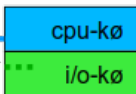
PD 4

PD 5

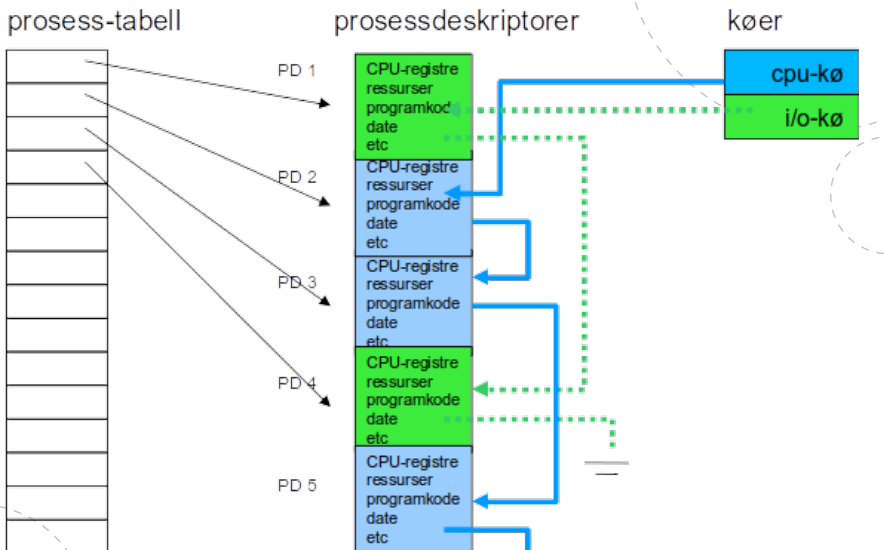
prosessdeskriptorer



køer



Prosessskøen



Prosesskøen

— Først i CPU-kø betyr?

Prosesskøen

- Først i CPU-kø betyr?
 - Kjørende prosess
- Prosess i I/U kø betyr?

Prosesskøen

- Først i CPU-kø betyr?
 - Kjørende prosess
- Prosess i I/U kø betyr?
 - Blokkert prosess, venter på I/U

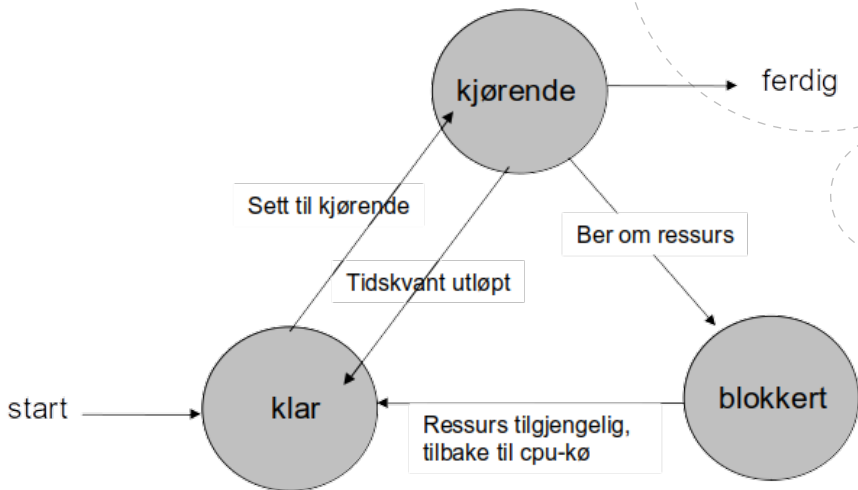
Prosess-tabellen

— Når endrer vi prosess-tabellen?

Prosess-tabellen

- Når endrer vi prosess-tabellen?
 - Når en prosess starter
 - Når en prosess slutter
 - Når tilstanden til en prosess endres

Prosessens tilstander



Prosessens tilstander

— Hvordan ser vi tilstander til prosesser?

Prosessens tilstander

- Hvordan ser vi tilstander til prosesser?
 - ps-kommandoen (ps = process status)
 - Viser informasjon fra prosess-tabellen og prosess deskriptorene

ps-kommandoen

```

x  _  □  donn@donn-ThinkPad-X260: ~

File Edit View Search Terminal Help
donn@donn-ThinkPad-X260:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 205724  5548 ?        Ss   okt.31   0:11 /sbin/init s
root         2  0.0  0.0      0      0 ?        S    okt.31   0:00 [kthreadd]
root         4  0.0  0.0      0      0 ?        S<   okt.31   0:00 [kworker/0:0
root         6  0.0  0.0      0      0 ?        S    okt.31   0:03 [ksoftirqd/0
root         7  0.0  0.0      0      0 ?        S    okt.31   1:57 [rcu_sched]
root         8  0.0  0.0      0      0 ?        S    okt.31   0:00 [rcu_bh]
root         9  0.0  0.0      0      0 ?        S    okt.31   0:00 [migration/0
root        10  0.0  0.0      0      0 ?        S<   okt.31   0:00 [lru-add-dra
root        11  0.0  0.0      0      0 ?        S    okt.31   0:00 [watchdog/0]
root        12  0.0  0.0      0      0 ?        S    okt.31   0:00 [cpuhp/0]
root        13  0.0  0.0      0      0 ?        S    okt.31   0:00 [cpuhp/1]
root        14  0.0  0.0      0      0 ?        S    okt.31   0:00 [watchdog/1]
root        15  0.0  0.0      0      0 ?        S    okt.31   0:00 [migration/1
root        16  0.0  0.0      0      0 ?        S    okt.31   0:02 [ksoftirqd/1
root        18  0.0  0.0      0      0 ?        S<   okt.31   0:00 [kworker/1:0
root        19  0.0  0.0      0      0 ?        S    okt.31   0:00 [cpuhp/2]
root        20  0.0  0.0      0      0 ?        S    okt.31   0:00 [watchdog/2]
root        21  0.0  0.0      0      0 ?        S    okt.31   0:00 [migration/2
root        22  0.0  0.0      0      0 ?        S    okt.31   0:05 [ksoftirqd/2
root        24  0.0  0.0      0      0 ?        S<   okt.31   0:00 [kworker/2:0
root        25  0.0  0.0      0      0 ?        S    okt.31   0:00 [cpuhp/3]
root        26  0.0  0.0      0      0 ?        S    okt.31   0:00 [watchdog/3]

```

ps-kommandoen

```

x  _  □  donn@donn-ThinkPad-X260: ~

File Edit View Search Terminal Help

PROCESS STATE CODES
Here are the different values that the s, stat and state output
specifiers (header "STAT" or "S") will display to describe the state of
a process:

      D    uninterruptible sleep (usually IO)
      R    running or runnable (on run queue)
      S    interruptible sleep (waiting for an event to complete)
      T    stopped by job control signal
      t    stopped by debugger during the tracing
      W    paging (not valid since the 2.6.xx kernel)
      X    dead (should never be seen)
      Z    defunct ("zombie") process, terminated but not reaped by
           its parent

For BSD formats and when the stat keyword is used, additional
characters may be displayed:

      <    high-priority (not nice to other users)
      N    low-priority (nice to other users)
      L    has pages locked into memory (for real-time and custom IO)
      s    is a session leader
      l    is multi-threaded (using CLONE_THREAD, like NPTL pthreads
           do)
      +    is in the foreground process group

Manual page ps(1) line 509/1167 45% (press h for help or q to quit)

```

Opprettelse av nye prosesser

- Når vi kjører ps-kommandoen, starter vi en ny prosess
- `/bin/ps`
- Hvordan?

Opprettelse av nye prosesser

- Når vi kjører ps-kommandoen, starter vi en ny prosess
- /bin/ps
- Hvordan?
 - Skall-programmet bruker systemkaller
 - Disse er **fork** og **exec**
 - Skall-programmet blir foreldre-prosess, og ps-kommandoen blir barne-prosess

Fork/exec

1. Lages det en ny prosessdeskriptor til den nye barneprosessen
2. Kopieres innholdet fra foreldreprosessen sin PD til barneprosessen sin PD
3. Barneprosessen får kopi av programkode, data og stakk til sine egne områder i minnet
 - Barneprosessen får adgang til alle filene som foreldre har adgang til

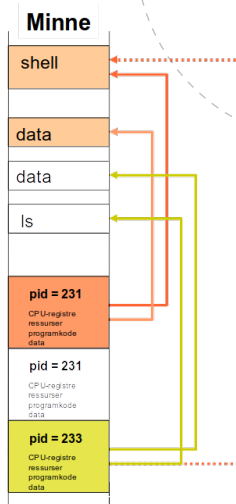
Fork

Rett etter fork-systemkallet har begge prosessene, både foreldre og barn, akkurat samme programkode (shell-koden). Etter at barneprosessen har utført exec-kallet skifter barneprosessen over til å bruke ls sin programkode.

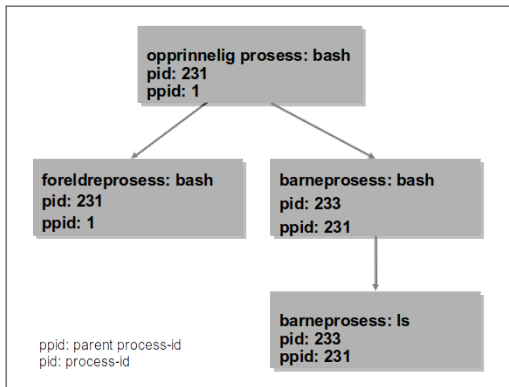
Prosesstabel
med PD'er til
prosessene

foreldre

barn



Fork/exec (ls)



før fork og exec

fork

etter fork

exec

etter exec

Fork/exec

- I Linux lages en prosess med systemkallet `fork()`.
 - Et systemkall er et funksjonskall til en tjeneste i kjernen av operativsystemet
- Hver prosess har en foreldre
- Det er foreldre-prosessen som utfører `fork`-kallet
- Barne-prosessen arver alt fra foreldre-prosessen
- Barne-prosessen er helt identisk med foreldre-prosessen bortsett fra returverdien fra `fork()`
 - Foreldren får returverdi lik barnets PID
 - Barnet får 0 i returverdi
- Prosesser har altså foreldre og barn
 - Vi kan snakke om hele prosesstrær
 - `ps axf -f`
 - `ps tree`

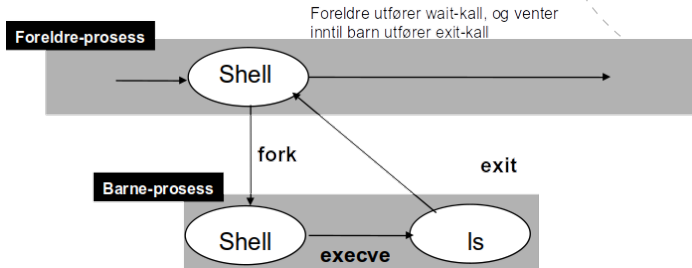
Fork/exec (ls)

- `fork()` - oppretter en ny prosess
- `execve()` - Kallende prosess bytter ut sin egen programkode med en annen
- `wait()`, `waitpid()` - foreldreprosess får status om sitt eget barn etter at det har avsluttet, og rydder opp i prosesstabellen etter barnet (hindrer dermed at det dannes en zombie)
- `exit()` - prosess som utfører `exit()` avslutter

Fork i C

```
// forktest.c
// Compile: gcc forktest.c -o forktest
int main()
{
    int i=0,n;
    n=fork();
    if (n == 0)
        i=i+1;
    else
        i=i-1;
    printf("pid %d has %d\n",n,i);
}
```

Fork/exec (ls)



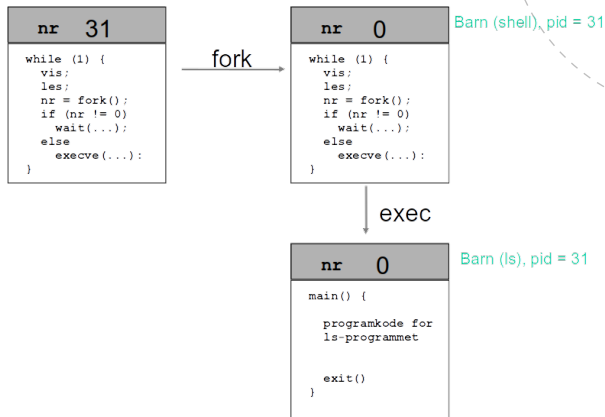
— Zombie prosesser?

http://yarchive.net/comp/zombie_process.html

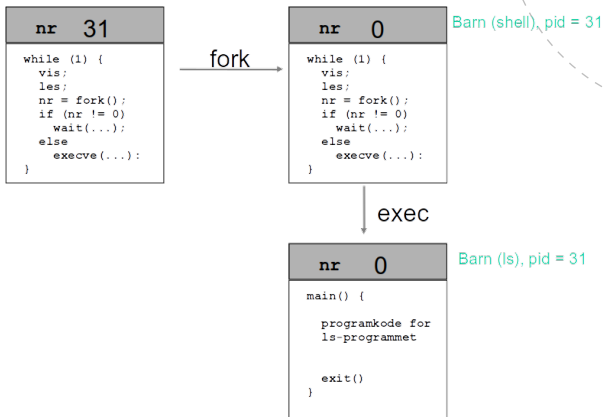
Enkelt skall-programm

```
int main()
{
    int nr;
    while (TRUE)
    {
        vis_prompt();
        les_kommando(kmd, opsjoner);
        sjekk_kommando(kmd, opsjoner);
        if (kmd == OK)
        {
            nr = fork();
            if (nr != 0)
                wait(&status);           /* foreldre */
            else
                execve(kmd, opsjoner, 0); /* barn */
        }
    }
}
```

Fork/exec (ls)



Fork/exec (ls)



Ex 1

Eksempel 1:

```
printf("hei\n");  
fork();  
printf("paa deg\n");
```

Utskrift fra programmet?

- a) hei
- b) hei
paa deg
- c) hei
paa deg
paa deg
- d) hei
paa deg
hei
paa deg

Ex 2

Eksempel 2:

```
printf("hei\n");  
fork();  
printf("paa deg\n");  
fork();  
printf("gamle venn\n");
```

Hva blir utskrift fra programmet:

- A) hei
- B) hei
paa deg
gamle venn
- C) hei
paa deg
gamle venn
gamle venn

- D) hei
paa deg
paa deg
gamle venn
gamle venn
gamle venn
gamle venn

Ex 2

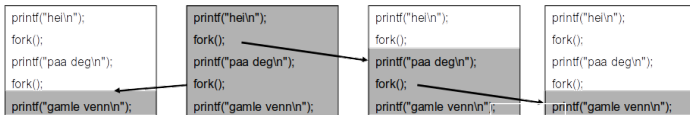
Eksempel 2:

```
printf("hei\n");
fork();
printf("paa deg\n");
fork();
printf("gamle venn\n");
```

Hva blir utskrift fra programmet:

- A) hei
- B) hei
paa deg
gamle venn
- C) hei
paa deg
gamle venn
gamle venn

D) hei
paa deg
paa deg
gamle venn
gamle venn
gamle venn
gamle venn



Ex 3

Eksempel 3:

```
printf("hei\n");  
execl("/bin/ls", "ls", "-al", (char *)0 );  
printf("studenter\n");
```

Hva blir utskrift fra programmet:

- A) hei
- B) hei studenter
- C) hei studenter + utskrift fra ls -al
- D) hei + utskrift fra ls -al

Ex 4

Gitt følgende program som er lagret på filen **a.out** i stående mappe:

```
#include "stdio.h"
#include "unistd.h"

main()
{
    printf("hei på deg ");
    sleep(2);
    execl("./a.out", "a.out", (char *) 0);
    printf("gamle ørn ");
    exit(0);
}
```

Hva skrives ut?

- A) Ingenting
- B) hei på deg
- C) hei på deg hei på deg hei på deg Etc
- D) hei på deg gamle ørn