

TDIU16

Föreläsning 1

Filip Strömbäck, Klas Arvidsson

- 1 Repetition: Vad gör ett operativsystem?
- 2 Pintos
- 3 Trådning i Pintos
- 4 Program i user-mode
- 5 Systemanrop

Varför finns operativsystem?

- Effektivitet
 - Kör fler processer samtidigt
 - Utnyttja resurser mer effektivt
- Säkerhet
 - Olika processer ska kunna samsas
 - Fel i en process ska inte påverka andra
 - (Isolera skadlig kod)
- Portabilitet
 - Köra samma program på olika hårdvara
- ...

Hur gör OS för att åstadkomma målen?

- Effektivitet
 - När en process väntar vill vi ha något annat att göra
 - Kör fler trådar/processer samtidigt
- Säkerhet
 - Processer ska inte komma åt varandras minne
 - OS kontrollerar åtkomst till hårdvara via systemanrop
- Portabilitet
 - OS tillåter inte direkt åtkomst till hårdvara, måste använda systemanrop
- ...

- 1 Repetition: Vad gör ett operativsystem?
- 2 **Pintos**
- 3 Trådning i Pintos
- 4 Program i user-mode
- 5 Systemanrop

Pintos: Översikt

- Pintos är ett litet operativsystem för undervisning
- ca 15 000 rader kod
- Innehåller:
 - Drivrutiner för hårdvara
 - Enkel schemaläggare (round-robin)
 - Enkelt filsystem (contiguous allocation)
 - Paging och minnesskyd för processer i user-mode
- I kursen kör vi Pintos i QEMU

Pintos: Struktur

Kärnan är strukturerad som följer:

- `userprog/` Kod för att hantera processer i user-mode
- `threads/` Implementation av trådning, synkronisering och schemaläggning
- `filesystem/` Implementation av filsystemet
- `devices/` Kod för att prata med hårdvara
- `lib/` Biblioteksfunktioner, både för kernel och user-mode

Finns också program att köra i user-mode:

- `examples/` Diverse exempel.
- `tests/` Automatiska tester.

Pintos: Kompilera och kör

- I TDIU16 vill du cd:a till userprog när du kompilarar och kör
- Kompilera: `make` (alt. med `-j8` eller bara `-j`)
- Kompilera examples: `make -C ../examples`
- Kör: `pintos <till QEMU> -- <till kernel>`
Exempel: `pintos -v -- -q`
- Tips: `make ... && pintos ...` för att inte glömma att kompilera.

- 1 Repetition: Vad gör ett operativsystem?
- 2 Pintos
- 3 **Trådning i Pintos**
- 4 Program i user-mode
- 5 Systemanrop

Trådning i Pintos

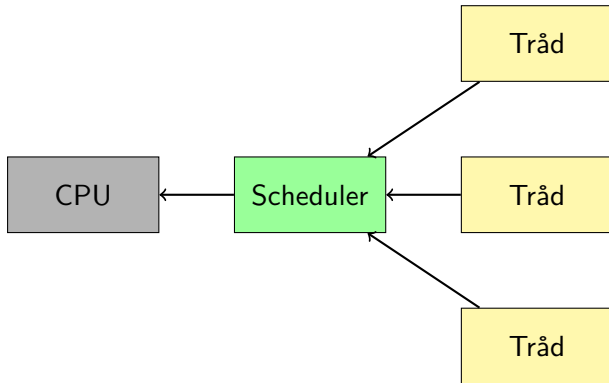
Mål:

- Kunna göra flera saker "samtidigt"
- Om någon behöver vänta kan vi göra något annat

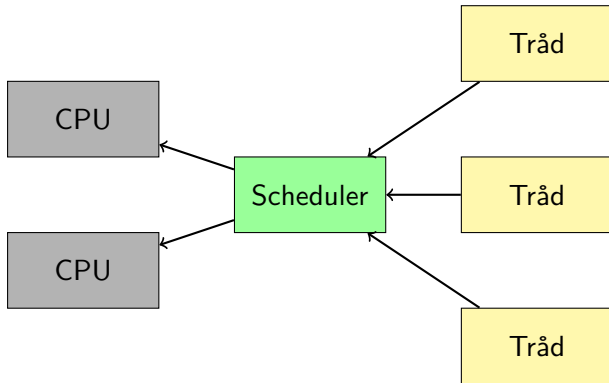
Lösning:

- Simulera fler "CPU:er" än vi har
- Varje sådan "CPU" kallar vi *tråd*
- Varje tråd kör sin del av programmet sekventiellt

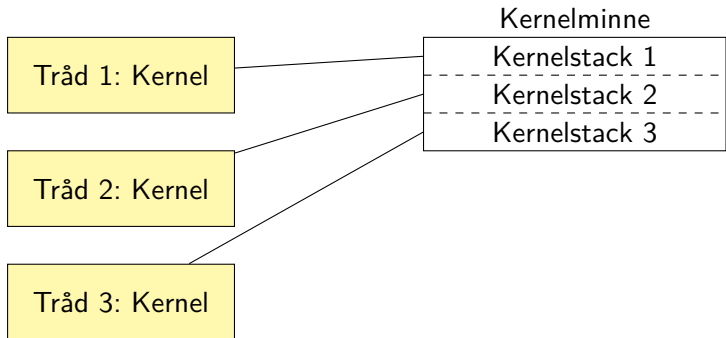
Trådning



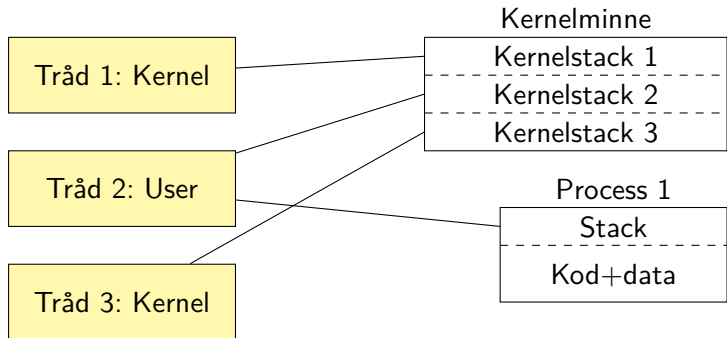
Trådning



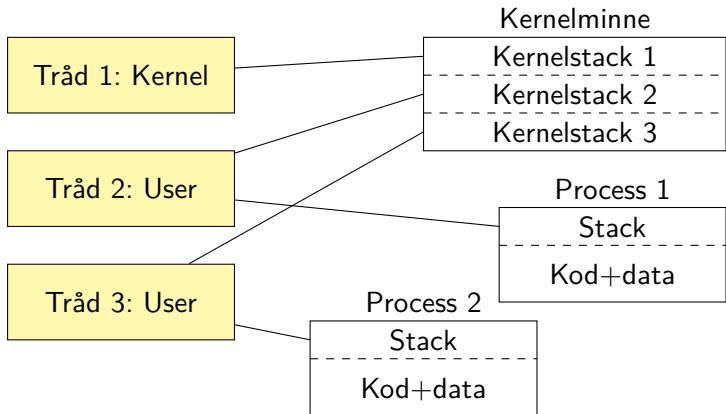
Typer av trådar



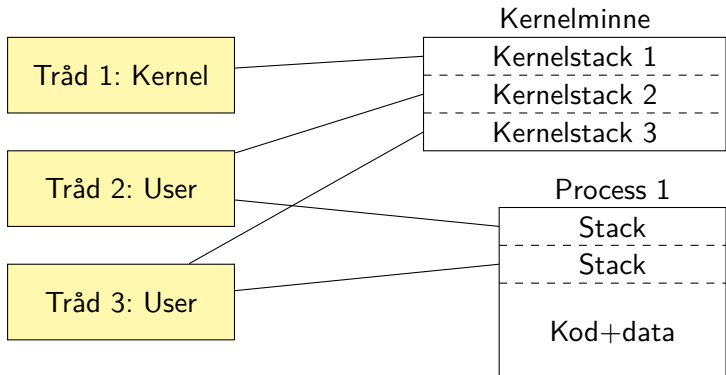
Typer av trådar



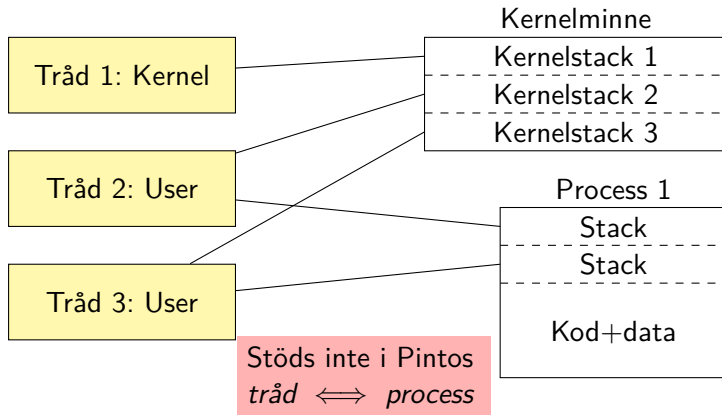
Typer av trådar



Typer av trådar



Typer av trådar



- 1 Repetition: Vad gör ett operativsystem?
- 2 Pintos
- 3 Trådning i Pintos
- 4 Program i user-mode
- 5 Systemanrop

Hur kör vi program i Pintos?

- Kompilera programmet (make i examples-mappen)
- Kopiera programmet till Pintos filsystem:
`pintos -p <fil> -a <namn i Pintos> -- ...`
- Be Pintos att köra programmet:
`pintos ... -- run <namn>`

Hur kör vi program i Pintos?

- Kompilera programmet (make i examples-mappen)
- Kopiera programmet till Pintos filsystem:
`pintos -p <fil> -a <namn i Pintos> -- ...`
- Be Pintos att köra programmet:
`pintos ... -- run <namn>`
- (Hur kan man felsöka?)

Dual-mode på x86

x86 implementerar
dual-mode med *ringar*:

- 2. (System Management)
- 1. (Hypervisor)
- 0. Kernel mode
 - 1. (används ej)
 - 2. (används ej)
 - 3. User mode

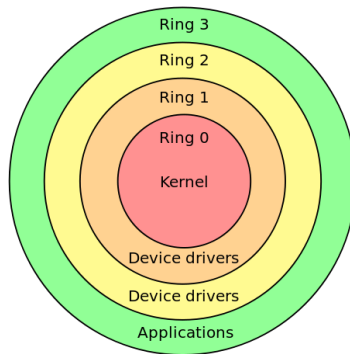


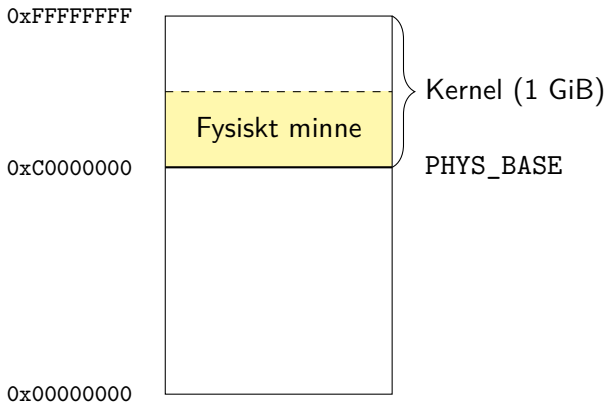
Bild: Wikipedia: "Protection ring"

Virtuellt minne

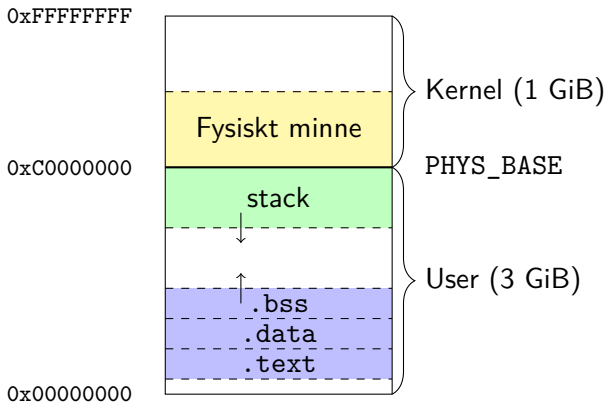
Varje process har sitt eget minne

- x86 har 2-nivåer av page-tables
- 4 KiB pages (12 bitar)
- `src/threads/pte.h`
- `src/threads/vaddr.h`
- `src/userprog/pagedir.{h,c}`

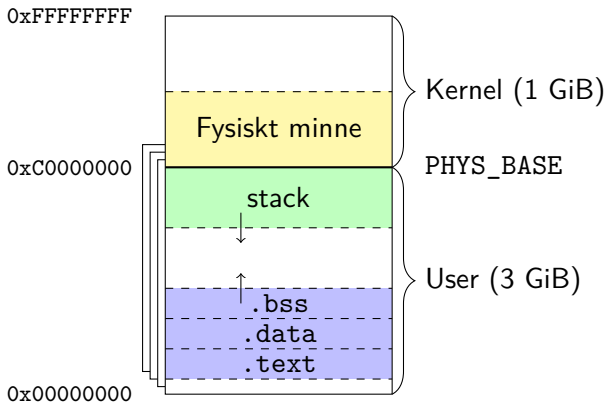
Virtuellt minne – layout i Pintos



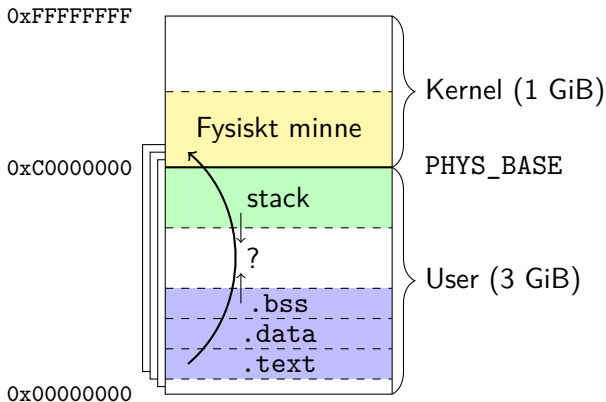
Virtuellt minne – layout i Pintos



Virtuellt minne – layout i Pintos



Virtuellt minne – layout i Pintos



- 1 Repetition: Vad gör ett operativsystem?
- 2 Pintos
- 3 Trådning i Pintos
- 4 Program i user-mode
- 5 Systemanrop

Varför systemanrop?

- Ett program i usermode (ring 3) är helt isolerat $\Rightarrow$ måste kunna kommunicera med omvärlden.
- OS kan hantera komplicerade resurser i större detalj än hårdvaran kan.
- Program behöver inte bry sig om olika typer av hårdvara. OS sköter detaljerna!

Mekanismer

På x86 finns:

- **Mjukvaruinterrupt** (int 0x30, int 0x80 på Linux)
- `sysenter / sysexit`
- `syscall / sysret`

Repetition: Funktionsanrop i C

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
void main() {  
    sum(1, 2);  
}
```

```
1  main:  
2      ;; ...  
3      pushl $2  
4      pushl $1  
5      call sum  
6      addl $8, %esp  
7      ;; ...
```

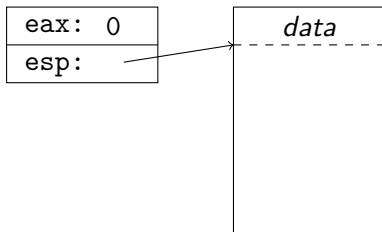
Repetition: Funktionsanrop i C

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
void main() {  
    int x = sum(1, 2);  
}
```

```
1  sum:  
2      movl 4(%esp), %eax  
3      addl 8(%esp), %eax  
4      ret  
5  main:  
6      ;; ...  
7      pushl $2  
8      pushl $1  
9      call sum  
10     addl $8, %esp  
11     movl %eax, "x"  
12     ;; ...
```

Exempel: Funktionsanrop i C



sum:

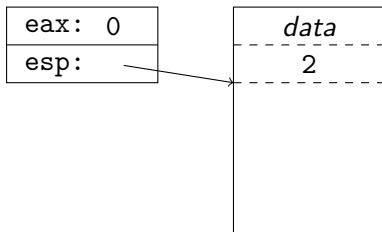
```
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
```

main:

```
;; ...
```

```
⇒ pushl $2
   pushl $1
   call sum
   addl $8, %esp
   movl %eax, "x"
```


Exempel: Funktionsanrop i C



sum:

```
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
```

main:

```
;; ...
```

```
pushl $2
```

⇒

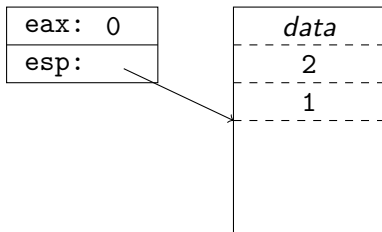
```
pushl $1
```

```
call sum
```

```
addl $8, %esp
```

```
movl %eax, "x"
```

Exempel: Funktionsanrop i C



sum:

```
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
```

main:

```
;; ...
pushl $2
pushl $1
```

⇒

```
    call sum
    addl $8, %esp
    movl %eax, "x"
```

Exempel: Funktionsanrop i C

eax: 0
esp:

<i>data</i>
2
1
<i>ret</i>

sum:
⇒ movl 4(%esp), %eax
addl 8(%esp), %eax
ret
main:
;; ...
pushl \$2
pushl \$1
call sum
addl \$8, %esp
movl %eax, "x"

Exempel: Funktionsanrop i C

eax: 1
esp:

<i>data</i>
2
1
<i>ret</i>

⇒

```
sum:
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
main:
    ;; ...
    pushl $2
    pushl $1
    call sum
    addl $8, %esp
    movl %eax, "x"
```

Exempel: Funktionsanrop i C

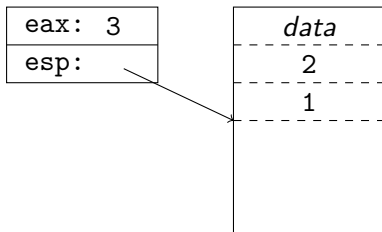
eax: 3
esp:

<i>data</i>
2
1
<i>ret</i>

⇒

```
sum:
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
main:
    ;; ...
    pushl $2
    pushl $1
    call sum
    addl $8, %esp
    movl %eax, "x"
```

Exempel: Funktionsanrop i C



sum:

```
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
```

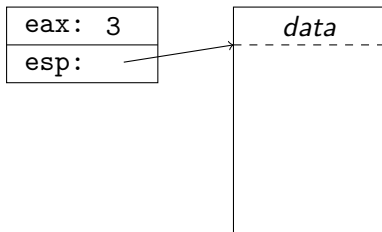
main:

```
;; ...
pushl $2
pushl $1
call sum
```

⇒

```
    addl $8, %esp
    movl %eax, "x"
```

Exempel: Funktionsanrop i C



`sum:`

```
    movl 4(%esp), %eax
    addl 8(%esp), %eax
    ret
```

`main:`

```
    ;; ...
```

```
    pushl $2
```

```
    pushl $1
```

```
    call sum
```

```
    addl $8, %esp
```

```
⇒    movl %eax, "x"
```

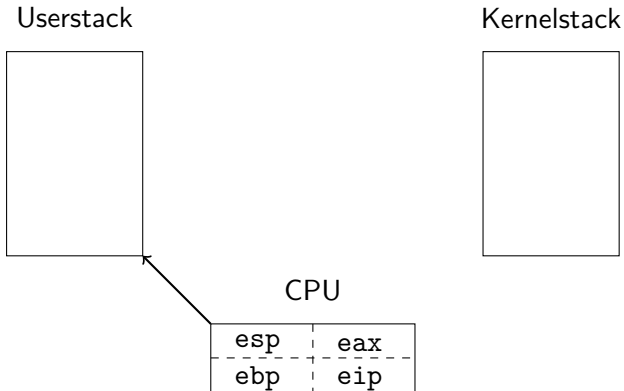
Systemanrop

Idé: `call` $\implies$ `int $0x30`

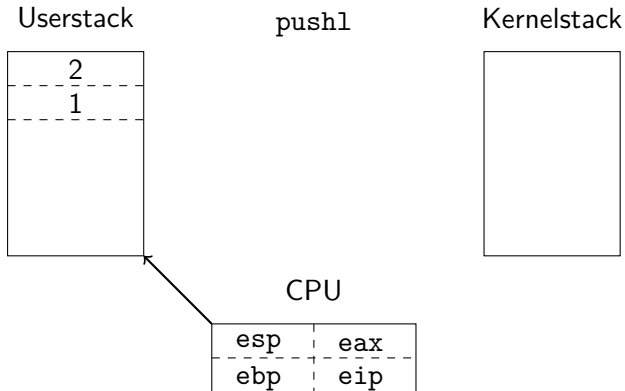
```
1    pushl $2
2    pushl $1
3    call sum
4    addl $8, %esp
```

```
1    pushl $2
2    pushl $1
3    int $0x30
4    addl $8, %esp
```

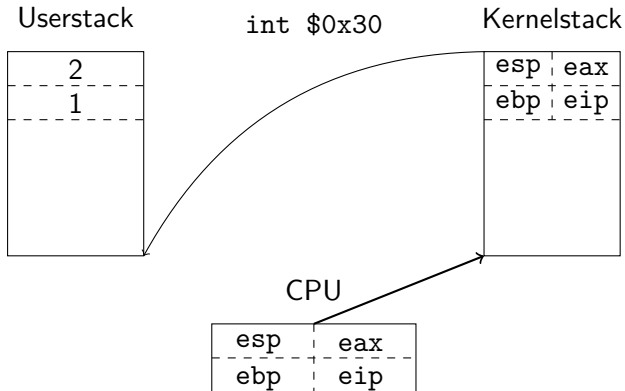

Till kernelmode



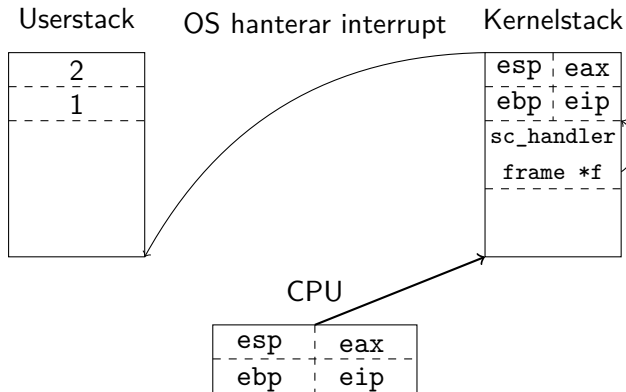
Till kernelmode



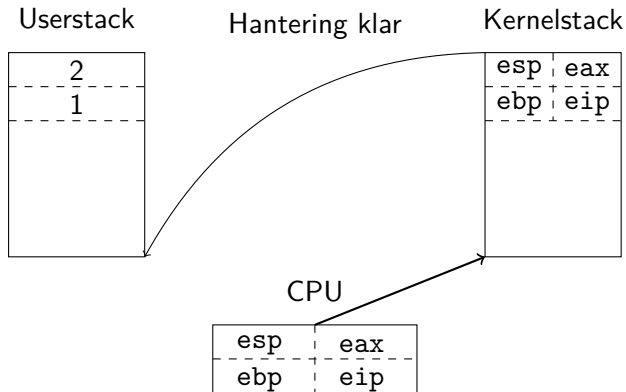
Till kernelmode



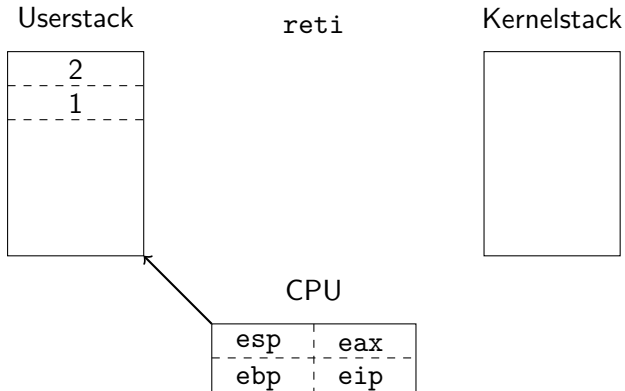
Till kernelmode



Till kernelmode



Till kernelmode



Vilket systemanrop?

Hur vet OS vilket systemanrop som ska köras?

```
1  pushl $2
2  pushl $1
3  pushl SYSCALL_NR
4  int $0x30
5  addl $8, %esp
```

src/lib/user/syscall.c

Till deadline 1

- (Installera Pintos)
- (C-intro)
- Systemanrop:
 - `exit`, `halt`
 - `read`, `write`
 - `open`, `close`
 - `seek`, `tell`, `filesize`

Vad är en fildeskriptor?

User: `int fd = open("namn");`
 `write(fd, "hej", 3);`
 `close(fd);`

Vad är en fildeskriptor?

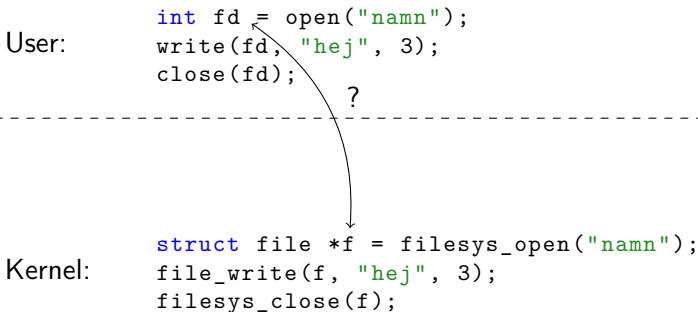
User:

```
int fd = open("namn");  
write(fd, "hej", 3);  
close(fd);
```

Kernel:

```
struct file *f = filesys_open("namn");  
file_write(f, "hej", 3);  
filesys_close(f);
```

Vad är en fildeskriptor?



Vad är en fildeskriptor?

User:

```
int fd = open("namn");  
write(fd, "hej", 3);  
close(fd);
```

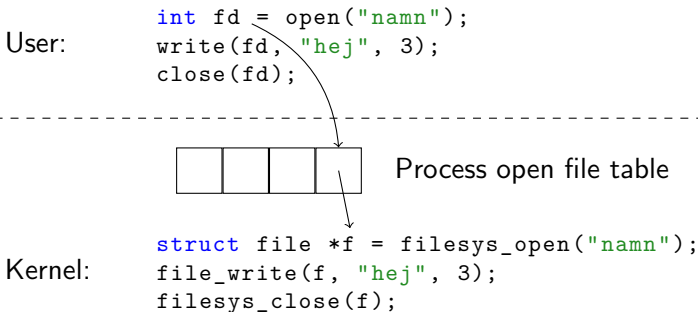


Process open file table

Kernel:

```
struct file *f = filesys_open("namn");  
file_write(f, "hej", 3);  
filesys_close(f);
```

Vad är en fildeskriptor?



Filip Strömbäck, Klas Arvidsson

www.liu.se