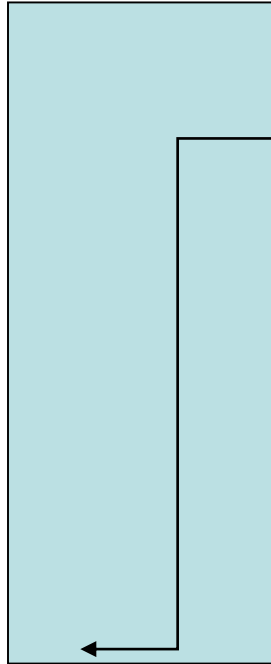

Operativsystem

Processer / trådar
(2.1 / 2.2 i boken)

Processer

- Process - definition
 - Ett program under exekvering
 - Varje process har sin egen "virtuella maskin" (CPU, minne, skivminnen etc.)
- Pseudo-parallelism
 - OS byter mellan processer (alla får sin time-slice)
 - "Multiprogrammering"

Processen ur CPU:s synvinkel



Primärminne

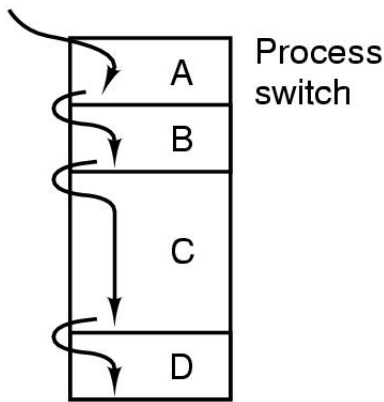
Programräknare
(PC - Program Counter)



"CPU:n är inte medveten om någon processmodell, utan exekverar systematiskt kod. OS måste implementera en processmodell."

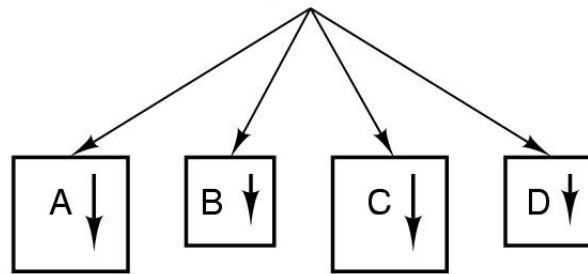
Processmodellen

One program counter

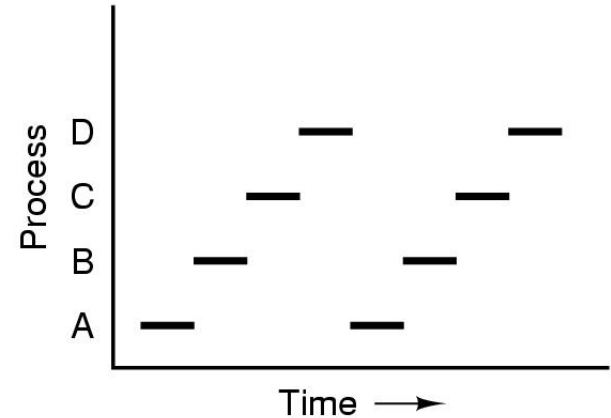


(a)

Four program counters



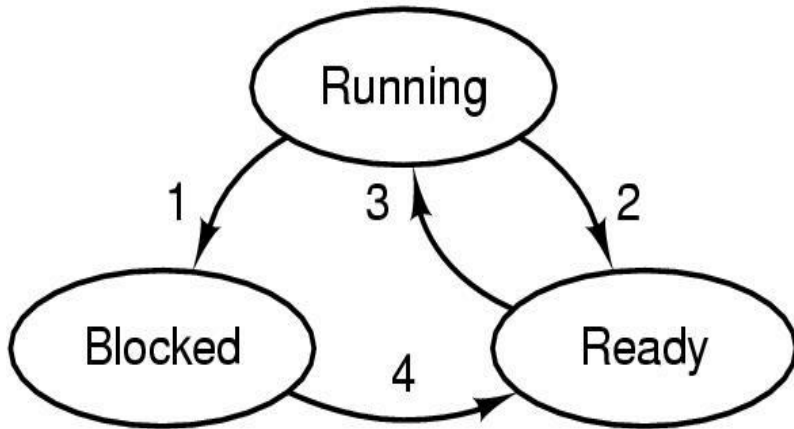
(b)



(c)

- Multiprogrammering med fyra program
- Modell med 4 oberoende, sekventiella program
- Endast ett program aktivt åt gången

Processtillstånd



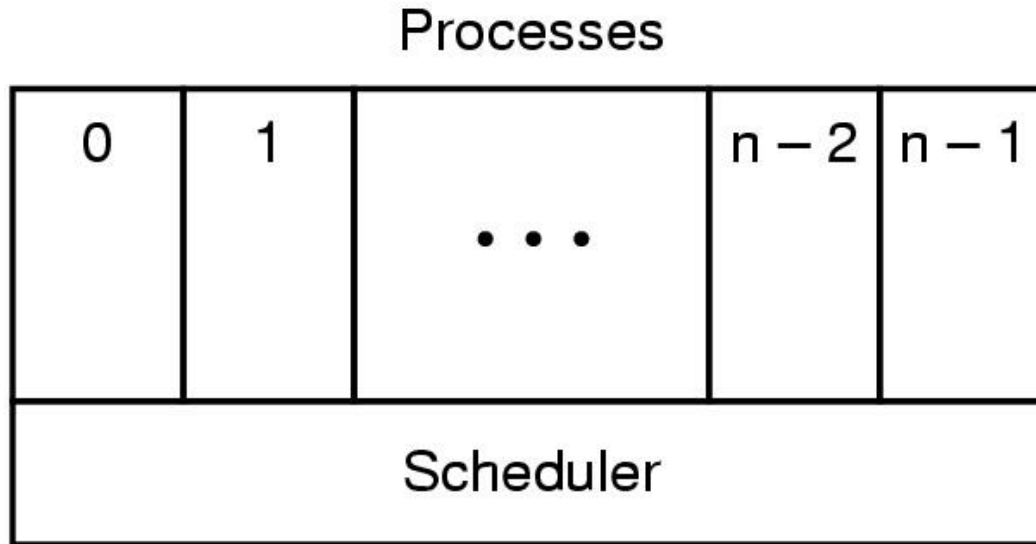
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Möjliga processtillstånd
 - exekverande (running)
 - blockerad (blocked)
 - redo (ready)

Processtillstånd - Linux

```
#define TASK_RUNNING          0
#define TASK_INTERRUPTIBLE    1
#define TASK_UNINTERRUPTIBLE  2
#define TASK_ZOMBIE           4
#define TASK_STOPPED          8
```

Processskedulering

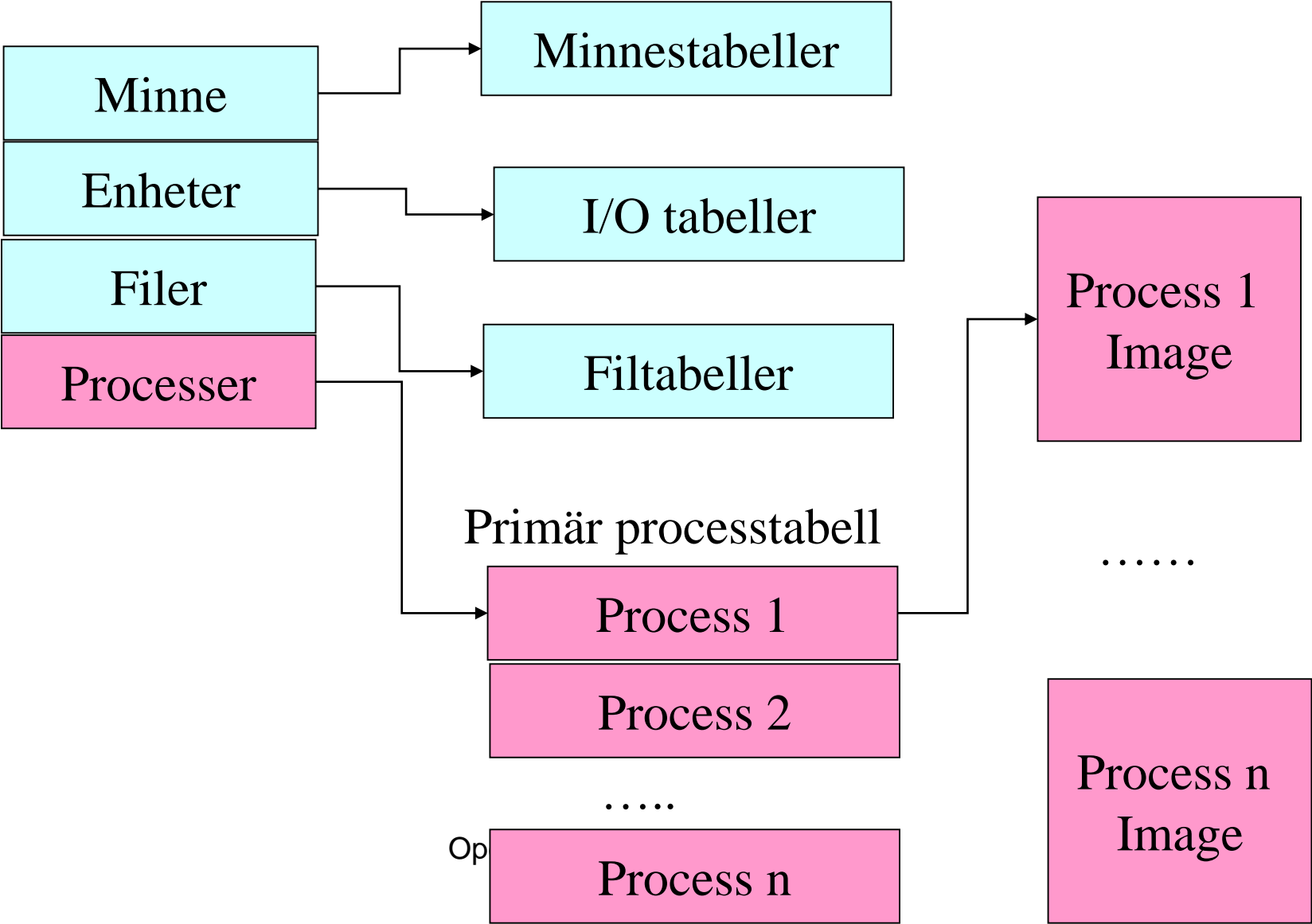


- Skeduleraren (alt. schemaläggaren)
 - Väljer vilken process som till näst får exekvera på processorn

Processköer som OS handhar

- Redokö :
 - Räkka med processer färdiga att exekveras
 - Skeduleraren väljer nästa process som för exekvera (**dispatching**)
- Blockeradkö :
 - Räkka med processer som är blockerade, t.ex. väntar på I/O

Processrelaterade tabeller som OS upprätthåller



tuxedo.abo.fi - default - SSH Secure Shell

File Edit View Window Help

Quick Connect Profiles

```

top - 09:56:46 up 10 days, 16:59, 55 users,  load average: 0.00, 0.00, 0.00
Tasks: 385 total,  2 running, 381 sleeping,  1 stopped,  1 zombie
Cpu(s):  0.1%us,  0.1%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2106480k total, 1977472k used, 129008k free, 105720k buffers
Swap:  4194296k total,   40k used, 4194256k free, 1143428k cached

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7876	jbjorkqv	15	0	21180	1416	888	R	1	0.1	0:00.88	top
1	root	18	0	10304	708	592	S	0	0.0	0:03.13	init
2	root	RT	0	0	0	0	S	0	0.0	0:03.77	migration/0
3	root	34	19	0	0	0	S	0	0.0	0:00.10	ksoftirqd/0
4	root	RT	0	0	0	0	S	0	0.0	0:00.00	watchdog/0
5	root	10	-5	0	0	0	S	0	0.0	0:00.55	events/0
6	root	10	-5	0	0	0	S	0	0.0	0:00.00	khelper
7	root	11	-5	0	0	0	S	0	0.0	0:00.00	kthread
9	root	10	-5	0	0	0	S	0	0.0	0:00.00	xenwatch
10	root	10	-5	0	0	0	S	0	0.0	0:00.00	xenbus
15	root	RT	-5	0	0	0	S	0	0.0	0:02.23	migration/1
16	root	34	19	0	0	0	S	0	0.0	0:00.01	ksoftirqd/1
17	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/1
18	root	10	-5	0	0	0	S	0	0.0	0:00.01	events/1
19	root	RT	-5	0	0	0	S	0	0.0	0:01.84	migration/2
20	root	34	19	0	0	0	S	0	0.0	0:00.03	ksoftirqd/2
21	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/2
22	root	10	-5	0	0	0	S	0	0.0	0:00.01	events/2
23	root	RT	-5	0	0	0	S	0	0.0	0:01.26	migration/3
24	root	34	19	0	0	0	S	0	0.0	0:00.02	ksoftirqd/3
25	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/3
26	root	10	-5	0	0	0	S	0	0.0	0:00.00	events/3
58	root	10	-5	0	0	0	S	0	0.0	0:00.00	kblockd/0
59	root	10	-5	0	0	0	S	0	0.0	0:00.00	kblockd/1
60	root	10	-5	0	0	0	S	0	0.0	0:00.00	kblockd/2
61	root	10	-5	0	0	0	S	0	0.0	0:00.00	kblockd/3
62	root	20	-5	0	0	0	S	0	0.0	0:00.00	cqueue/0

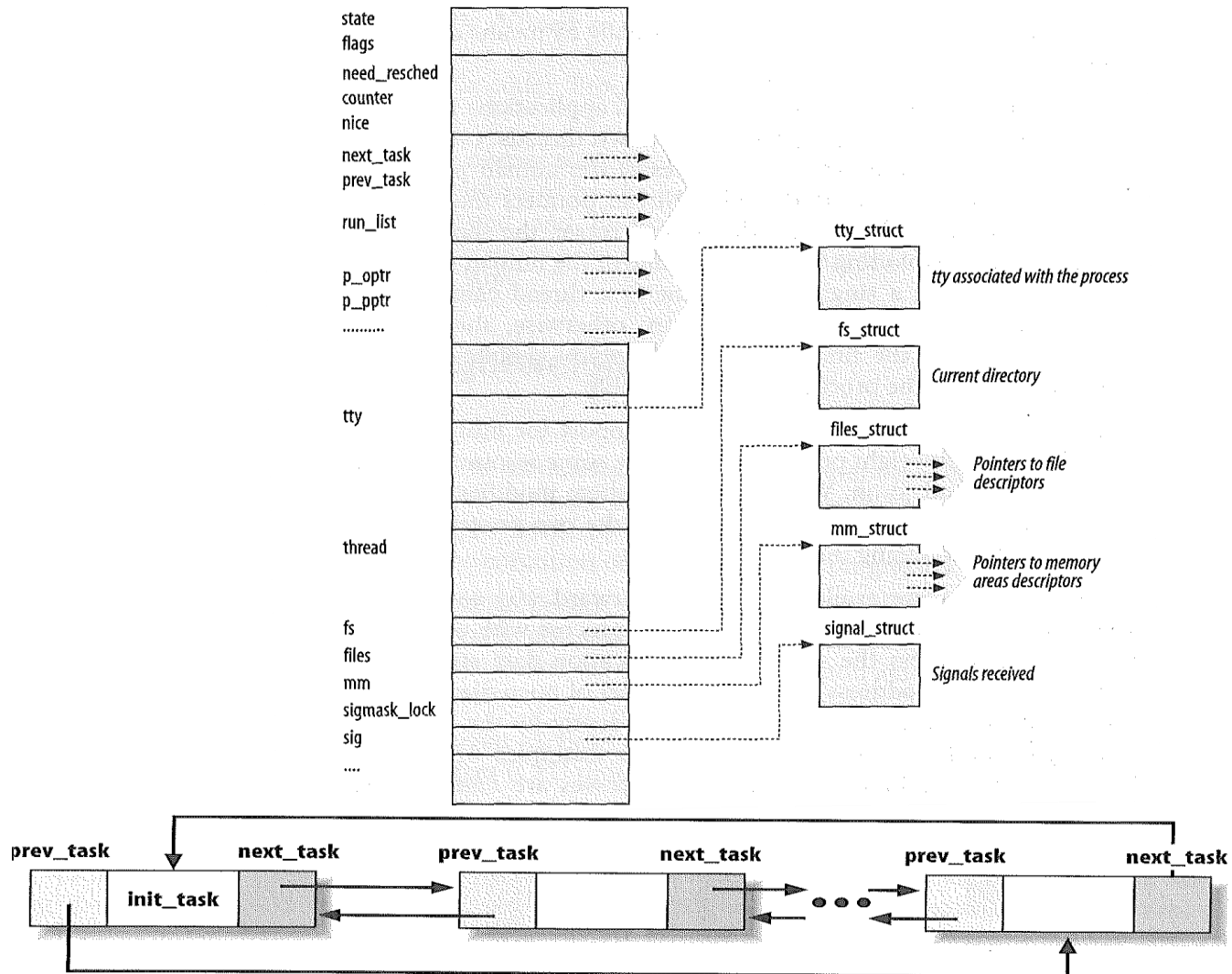
Connected to tuxedo.abo.fi

SSH2 - aes128-cbc - hmac-md5 - none 81x34

Process-deskriptor

- Processidentifizierung
 - Process ID, parent ID, user ID
- Prozessstatus
 - Ready, Running, Blocked
- CPU-status
 - Register, PC, SP
- Filesystem
- Filetabell
- Minnestabell
- Signaler

Datastrukturer i linux



Process-deskriptor - Linux

- definierad i (valda delar nedan)

/usr/src/linux/include/linux/sched.h

```
struct task_struct {
    volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    struct mm_struct *mm; /* memory tables */
    struct task_struct *next_task, *prev_task;
    pid_t pid, pgrp, tty_old_pgrp, session, tgid;
    uid_t uid, euid, suid, fsuid;
    /* CPU-specific state of this task */
    struct thread_struct thread;
    /* filesystem information */
    struct fs_struct *fs;
    /* open file information */
    struct files_struct *files;
}
```

Process-deskriptor – Linux (2)

- `thread_struct`

/usr/src/linux/include/asm-i386/processor.h

```
struct thread_struct {
    unsigned long    esp0;
    unsigned long    eip;
    unsigned long    esp;
    unsigned long    fs;
    unsigned long    gs;
    unsigned long    cr2, trap_no, error_code; /* fault info */
    union i387_union    i387; /* floating point info */
    struct vm86_struct    * vm86_info;
    /* IO permissions */
    int                ioperm;
    unsigned long      io_bitmap[IO_BITMAP_SIZE+1];
};
```

Processer i Linux

- Process PID=0
 - Swapper process
 - Förfader till alla andra processer
 - "Körs" då ingen annan process är i tillståndet TASK_RUNNING
- Process PID=1
 - Skapas av process 0 och kör `init()`-funktionen i kärnan. `init()`-funktionen i sin tur laddar det exekverbara programmet *init*. *Init* startar och övervakar alla andra processer i systemet

Processer - livscykel

- Följande händelser skapar processer
 - Då systemet initialiseras (process 0 / init())
 - Då en annan process skapar en process
- Följand processer avslutar processer
 - Normal exit() / exit() med felsituation
 - Fatalet fel (ofrivillig)
 - Avslutas (kill()) av en annan process

Skapa process i UNIX/Linux

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
main() {
    pid_t pid;
    printf("Nu tänker jag dela mig!!\n");
    pid = fork();
    if (pid)
        printf("Jag är föräldern\n");
    else
        printf("Jag är barnet\n");
}
```

Processer i Windows

- Win 95, 98, ME (MS-DOS-baserade) implementerar EJ processmodellen!!
- Processer skapas i NT, Win2K, XP med:

```
STARTUPINFO si;
PROCESS_INFORMATION pi;
// Start the child process.
if( !CreateProcess( NULL, // No module name (use command line).
    TEXT("MyChildProcess"), // Command line.
    NULL, // Process handle not inheritable.
    NULL, // Thread handle not inheritable.
    FALSE, // Set handle inheritance to FALSE.
    0, // No creation flags.
    NULL, // Use parent's environment block.
    NULL, // Use parent's starting directory.
    &si, // Pointer to STARTUPINFO structure.
    &pi ) // Pointer to PROCESS_INFORMATION structure.
)
```

Context switch

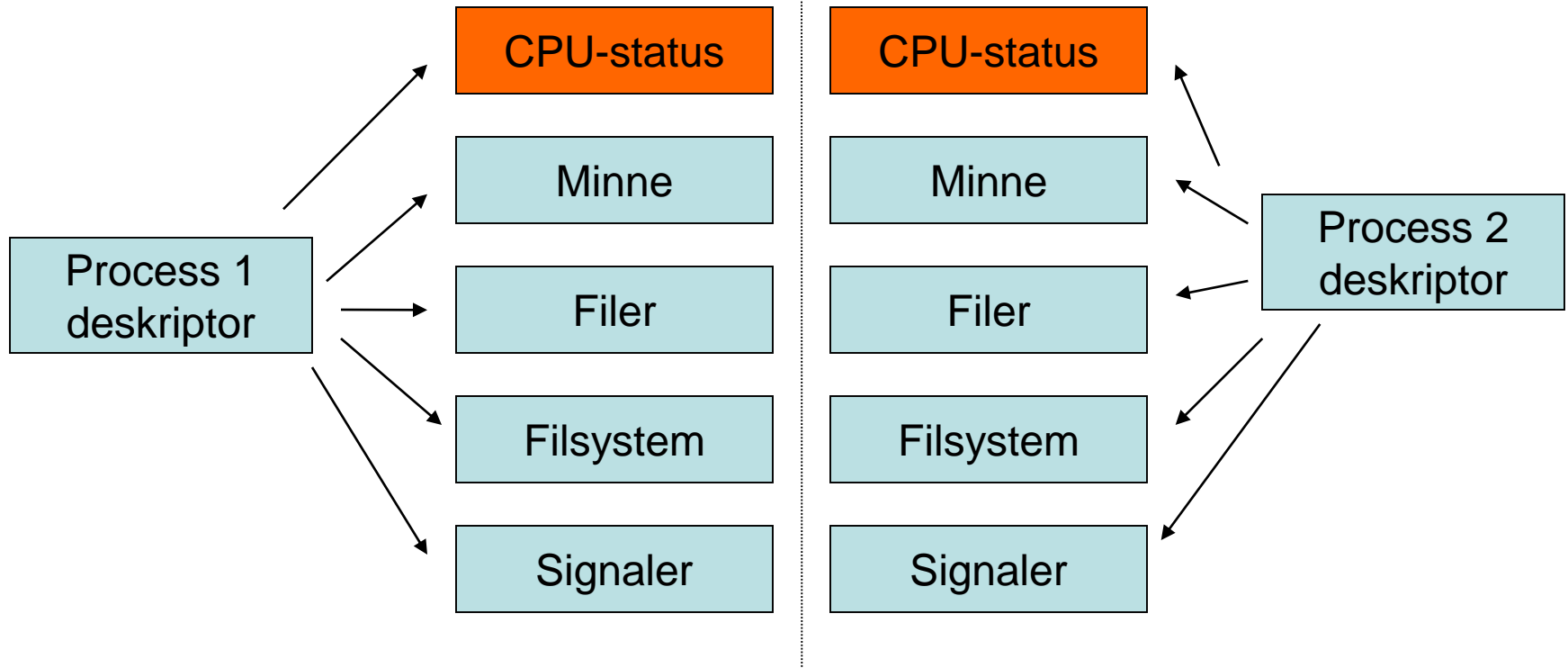
- Kernelen måste kunna avbryta exekveringen av en process och byta till en annan körbar process
- Byte från en process till en annan
 - Spara register, PC, SP
 - Byte av minnesrymd (Virtuell minnesrymd)
 - Ladda register, PC, SP
 - *CurrentProcess = pointer to process table item*
- Kod för kontextswitch till stor del i assembler

Context switch: Linux i386

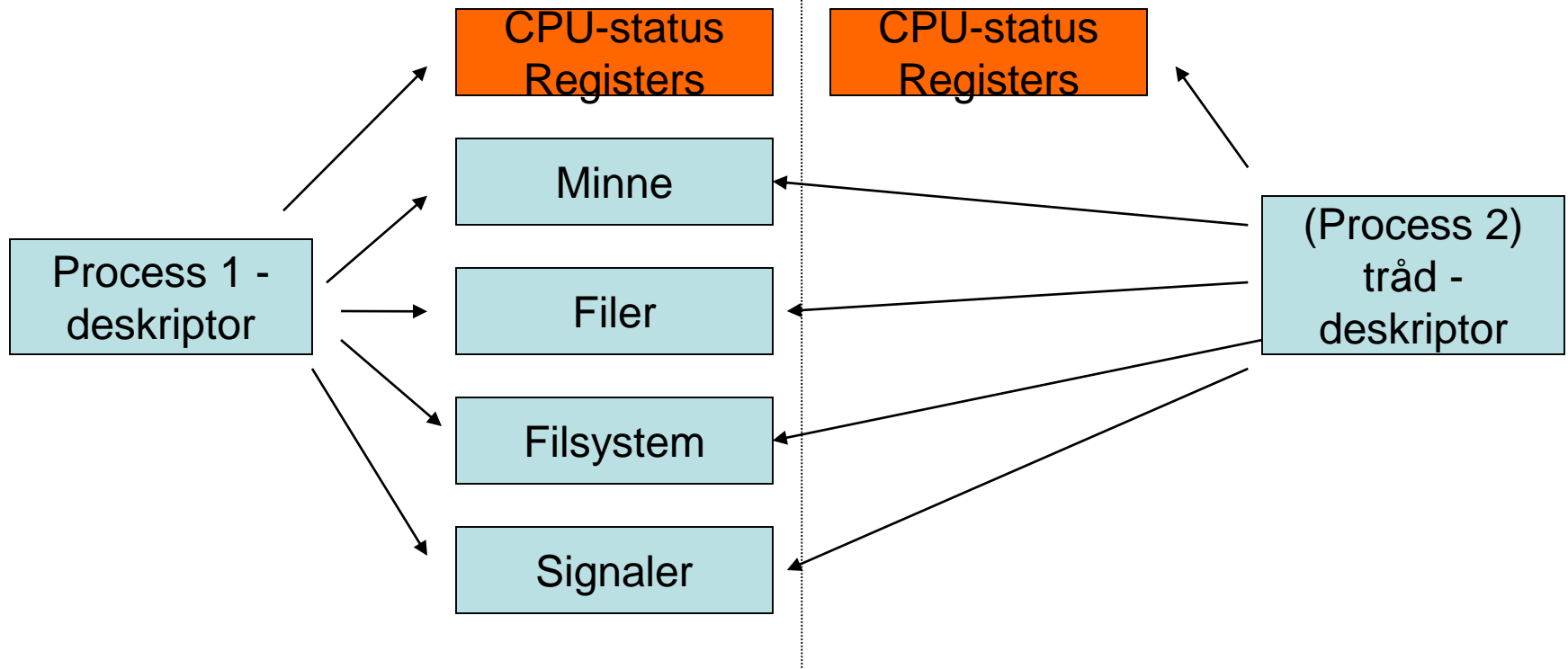
file: include/kernel/asm-i386/system.h

```
#define switch_to(prev,next,last) do {
asm volatile("pushl %%esi\n\t"
             "pushl %%edi\n\t"
             "pushl %%ebp\n\t"
             "movl %%esp,%0\n\t"          /* save ESP */
             "movl %3,%%esp\n\t"        /* restore ESP */
             "movl $1f,%1\n\t"          /* save EIP */
             "pushl %4\n\t"             /* restore EIP */
             "jmp __switch_to\n\t"
             "1:\t"
             "popl %%ebp\n\t"
             "popl %%edi\n\t"
             "popl %%esi\n\t"
             : "=m" (prev->thread.esp), "=m" (prev->thread.eip),
              "=b" (last)
             : "m" (next->thread.esp), "m" (next->thread.eip),
              "a" (prev), "d" (next),
              "b" (prev));
} while (0)
```

Mer om processdeskriptorn



Mer om processdeskriptorn



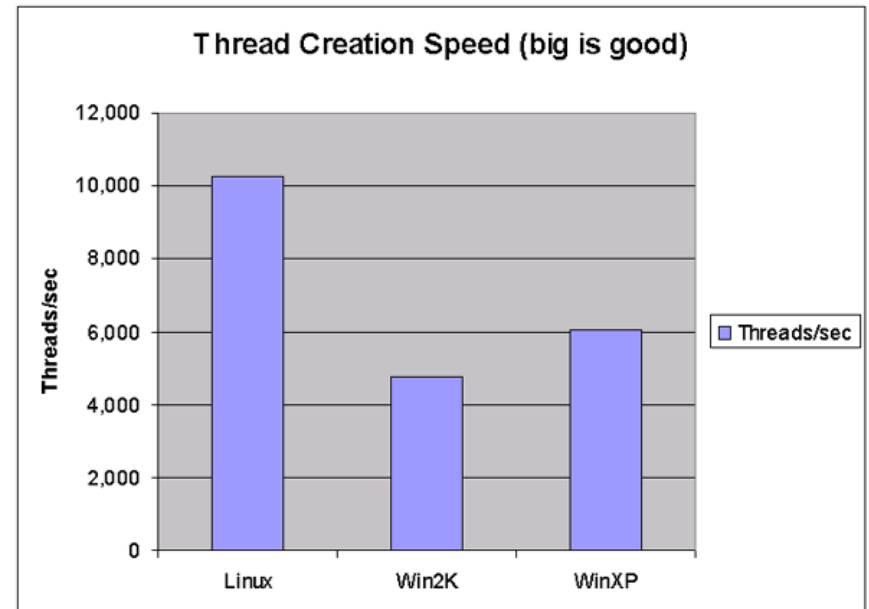
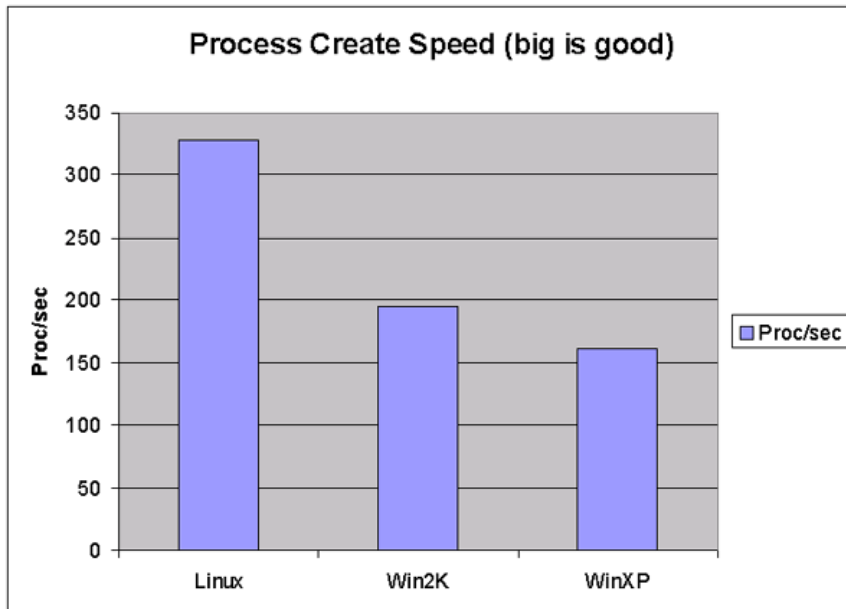
Trådar

- En tråd symboliserar en exekveringsstig i programmet
 - En process kan ha flera trådar
 - Trådarna delar på processens tilldelade resurser
 - minne
 - öppna filer / filsystem
 - signaler
- Kallas också en "lätt" process (lightweight process)

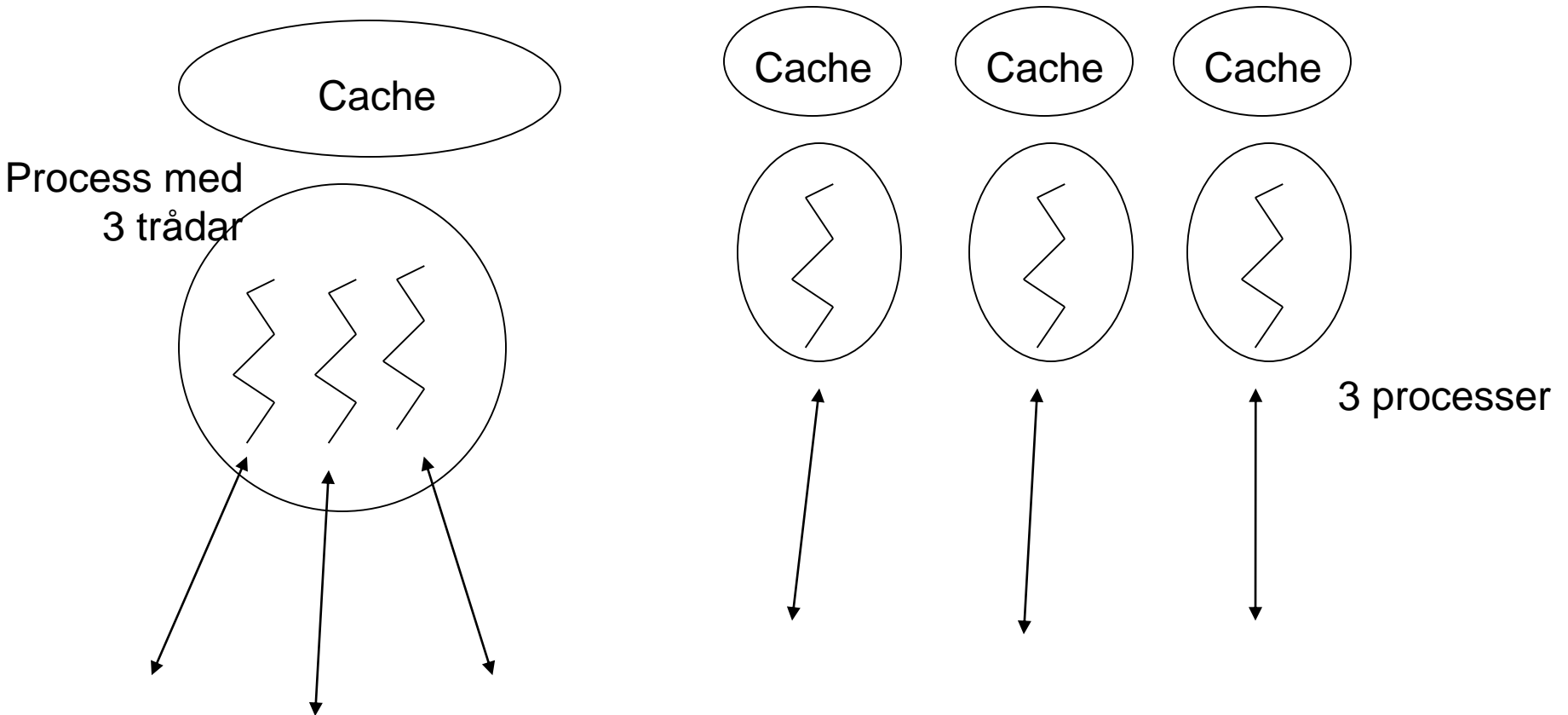
Trådar

- När behövs trådar?
 - Web-server: Flere förfrågningar kan bearbetas samtidigt
 - Databas-server: Samma orsak
 - Textbehandling: Varför vänta medan utskriften formatteras för skrivaren?
- Det går snabbt att skapa trådar (ca 50 gånger snabbare)

Skapa processer / trådar



Exempel: Trådar och web-server



Genom att använda trådar, kan cacheminnet delas !

Nackdelar med trådar

- Delat minne = krav på korrekthet
 - En tråd kan skriva sådan information i det delade minnet så att andra trådar inte kan fortsätta korrekt
- Krav på synkronisering
 - Eftersom minnet är synligt för alla trådar, måste de sinsemellan synkronisera minnesaccess

Exempel på synk.behov

- Tråd 1

```
tmp = count;  
count = tmp+1;
```

- Tråd 2

```
tmp = count;  
count = tmp+1;
```

Resultatet av beräkningen är beroende av exekveringsordningen (skeduleringen)

NOTERA: I kapitlet IPC tas upp synkroniseringsmetoder

Trådar - komplikationer

- Multitråd-program är svårare att skriva
 - funktioner måste normalt tillåta att flere trådar samtidigt är aktiva
 - kritiska sektioner måste skyddas med semaforer / mutex
 - minnesaccesser
 - datastrukturer
 - Trådar måste synkroniseras

Trådar i Linux

- Linux använder “lätta” processer
 - Två lätta processer kan dela vissa resurser
 - Adressrymd (CLONE_VM)
 - Öppna filer (CLONE_FS)
 - Filsystem (CLONE_FILES)
 - Signaler (CLONE_SIGHAND)
 - PID (CLONE_PID)
- System-anrop *clone*
 - `clone(int (*fn)(void *arg), void *child_stack, int flags, void *arg)`
- *clone* är Linux specifik, bör inte användas i portabel kod

Kärn-trådar

- Vissa operationer lönar sig inte att göra enligt den linjära modellen:
 - flusha cache-minnen
 - svappa sidramar
 - handha nätverksuppkopplingar
- Kernel-trådar används för funktionerna ovan
- Kerneltrådar
 - körs i kernel-mode
 - använder endast kernel-minne
 - kan endast skapas av en annan kernel-tråd

Trådbibliotek

- Bibliotek - *pthread*

- *POSIX-standard för trådar (portabel kod)*

- *Skapa tråd*

```
int pthread_create(pthread_t * thread,  
pthread_attr_t * attr, void * (*start_routine)(void  
*), void * arg);
```

- *Avsluta tråd*

```
void pthread_exit(void *retval);
```

- *Vänta på tråd*

```
int pthread_join(pthread_t th, void **thread_return);
```


Trådar i Windows

```
#include <windows.h>
#include <stdio.h>

DWORD WINAPI ThreadFunc( LPVOID lpParam )
{
    printf( "Parameter = %d.", *(DWORD*)lpParam );

    return 0;
}

VOID main( VOID )
{
    DWORD dwThreadId, dwThrdParam = 1;
    HANDLE hThread;

    hThread = CreateThread(
        NULL, // default security attributes
        0, // use default stack size
        ThreadFunc, // thread function
        &dwThrdParam, // argument to thread function
        0, // use default creation flags
        &dwThreadId); // returns the thread identifier

    // Check the return value for success.

    if (hThread == NULL)
    {
        printf( "CreateThread failed (%d)\n", GetLastError() );
    }
    else
    {
        getch();
        CloseHandle( hThread );
    }
}
```

Fiber i Windows

- Windows definierar något de kallar "fiber"
- Fiber definieras som en tråd som programmet själv bör skedulera
- I praktiken motsvarar detta alltså en user mode thread