# Hoofdstuk 2

## Instructies

Programma voert instructies uit;

Gewone instructies: mov, add, imul, idiv, jmp, cmp

Gepriviligeerde instructies: hlr, in, out

## Processortoestand:

Veiligheid!!!

### User mode:

Standaard modus, heeft toegang tot de gewone instructies.

### Kernel mode:

Heeft toegang tot alle instructies, ook de I/O instructies enkel het besturingsysteem kan in kernelmode gaan.

## Interrupts

Processor voert één programma tegelijk uit, het bestuuringssysteem is een programma.

Besturingsysteem moet in actie komen bij een I/O input of vraag van hogergelegen software

OS is event driven

* Altijd in het geheugen
* Enkel actief wanneer nodig

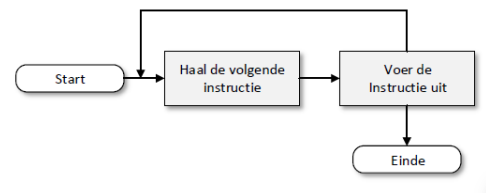
### Interrupt requests:

Stel: netwerkpakket komt binnen

1. Netwerkkaart stuur interrupt request.
2. Processor onderbreekt het actieve programma

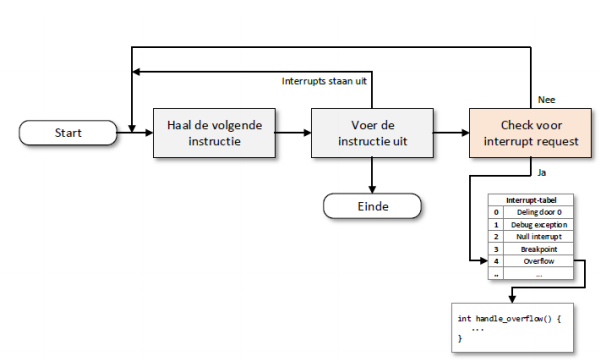
Geen interrupt requests = geen onderbreking van het actieve programma.

### Haal- en uitvoeringscyclus:



1. Haal de volgende instructive
2. Voer de instructive uit
3. Als er nog een instructive is, begin opnieuw
4. Al ser geen meer is einde.

### Met interrupt ondersteuning:



1. Haal de volgende instructive
2. Voer instructive uit
3. Check voor interrupt request
4. Er is geen interrupt: begin terug opnieuw
5. Er is wel een interrupt, kijk welke in de interrupt table.
6. Handel interrupt

Afhandeling van interrupts kan worden uitgeschakeld (tijdens het afhandelen van een interrupt)

Interrupt request = elektrisch signal

Veel mogelijke redenen:

* Delen door nul
* Leesopdracht voltooit
* …

Elke interrupt moet anders afgehandeld worden

### Interrupttabel

Voor elke interrupt een geheugenadres (interrupt vector)

#### Voorbeeld: leesopdracht harde schijf

1. Schijfbesturingseenheid ontvangt een leesopdracht
2. Leesopdracht is voltooid => interrupt request naar interrupt controller.
3. Interrupt controller geeft request door aan processor
4. Interrupt controller sturrt volgnummer van de oorsprong van de interrupt request
5. Processor zoekt adres van de juiste interrupt handler in de interrupttabel
6. Inhoud van de bevelenteller, registers en de vlaggen worden opgeslagen.
7. Adres van interrupt handler wordt in bevelenteller geplaatst en de processor wisselt naar Kerneltoestand
8. Sprong naar handler, en de request wordt afgehandeld
9. Interrupt handler herstelt bevelenteller, registers en processorstatus en de processor wisselt aar gebruikerstoestand.

stap 6 en 7 zijn de context switch toestand wijzigen van de processor

* Toestand van huidige programma opslaan
* Toestand van nieuwe programma laden
* Volgende instructive uitvoeren

## Soorten interrupts

### Externe interrupts (= van buiten de processor)

* Klok inerrupts
* I/O interrupts
* Keyboard interrupts
* Hardwarefouten

Zijn asynchroon, hebben niets te maken met huidige programma

### Interne interrupts (= interrupts van binnen in de processor)

* Uitzonderingen (exceptions)
* Traps (software interrupts)

Zijn synchroon, toepassing kan niet verder tot dat de interrupt afgehandeld is

## Interrupt handler:

Ineterne interrupts:

Processor gaat conditiecode in toestandsbeschrijving na

Programma dat trapt naar BS geeft informatie mee om juiste handler te kiezen

Externe interrupts:

1. Langs de in interruptlijn, afhankelijk van de interruptlijn waar het interrupt signal binnenkomt (niet meer controllers mogelijk dan interruptlijnen)
2. Gemeenschappelijke interruptlijn, Software pollen welke controller een interrupt heft gegeven (traag)
3. Gemeenschappelijke interruptlijn en acknowledge lijn, gaat langs deze lijn na welke controller een interrupt heft gestuurd, Daisy Chaining (snel langs hardware)

### Meerder interrupts tegelijk?

Welke eerst?

Prioriteit, afhankelijk van de prioriteit bepaalt door de interrupt handler.

Asynchronen interrupts tijdens interrupt afhandeling kunnen tijdelijk worden geblokeerd of als ze een hogere prioriteit hebben dan de huidige uitvoeren.

## System calls

Interrupts, Hardware <> besturingssysteem

System calls, Software <> besturingssysteem

* Opstarten of beeindigen van een programma
* Programma laten wachten op een gebeurtenis
* Toekennne en vrijgeven van geheugen
* Bestandsbeheer
* Besturing en beheer randapparaen
* Uitisseln van informatie

### Read-system call in Linux

Read (file\_desc, buffer, nbytes) = C code, wanneer een programma deze functie aanroept zal hetvolgende gebeuren:

1. Oproepend programma zet warden van de drie parameters op de staple
2. Read functie wordt aangeroepen
3. Functie zet system call nummer in het juiste register en voert TRAP functie uit
4. Interrupt service routinge van het OS zoekt juiste routine op voor de system call
5. Processor start systeemroutine
6. Systeemroutine zet resultaat op voorziene plaats, verlaat kernel mode en geeft controle terug aan de read functie
7. Read functie ontvangt gegens
8. Uitvoer oproepend programma gaat verder

# Hoofdstuk 3: boot process

## Opstarten:

1. Voiding inschakelen

* Eigen spanningsniveaus controleren
* Stabiele spanning? (signal naar moederbord)

1. Moederbord

* Continu rest-signaal naar processor
* Power good? Reset signal valt weg

1. Processor start

* Kernel mode
* Adressering in Real mode
* Op adres 0xFFFFFF0

1. Memory-mapped I/O

* Inhoud van de ROM-chip wordt gemapped op de RAM

1. Opstarcode UEFI (Unifed Extensivle Firmware Interface) wordt geladen

* Mini-besturingssysteem dat de hardware initialiseert en het besturingsysteem inlaadt
* Opvolger van BIOS
* Modulair ontwerp
* Boot service: enkel voor BS is opgestart (UEFI interface)
* Runtime service: altijd besikbaar (opvragen datum/tijd)

1. Besturingsysteem inladen (Partition bootloader)

* Besturingsysteem wordt van de harde schijf gehaald (primary partition)

1. Boot loader
2. BOOTMGR

## Real vs. protected mode

### Real mode:

* 20-bit adressen
* In segmentregister

### Protected Mode:

* 32-bit adressen
* Basis in segmentdescriptor

# Besturingssysteem inladen

Waar staat het besturingssysteem, ergens op een secundaire opslag. (volgorde is boot sequence)

## Partities

Harde schijf is verdeeld in partities.

Partitie informatie staat in de GUID

LBA 0 = Protectieve MBR combatibiliteit met oudere systemen, bevat een partitie ter grootte van de schijf

LBA 1 = Primary GPT header, grootte van de schijf, beschikbare sectoren, unieke identifier, pointer naar secondary header

LBA 2 = partition entry, type, unieke identifier, start-en eindsector, attributen, partitienaam

## OS boot loader

Opstarten van het OS op de partitie.

# Hoofdstuk 4

## Dataopslag

Fysische ordening (sectoren).

Logisch ordening (bestanden)

Bestand definitie:

Een verzameling gerelateerde gegevens, die door een gebruiker of een applicatie op een bepaalde manier geïnterpreteerd word

## Kenmerken

* Naam
* Grootte
* Toegangsrechten
* Aangemaakt/aangepast/…

## Lineaire Directory

Alle bestanden in een directory

Eenvoudig maar onhandig

## Hiërarchisch bestandssysteem

Standaard, bastanden in directories en directories in directories

### Absoluut pad:

Vanaf het begin van de schijf tot bestand

### Werkdirectory

Huidige directory van de gebruiker

### Relatief pad:

Ten opzichte van het werkdirectory

## Boomstructuur:

Boomstructuur is de structuur van een filesysteem.

Iedere keer controleren als er een lus kan veroorzaakt worden.

## Aaneengesloten bestanden:

Een bestand staat in sectoren op een harde schijf, dit kan aaneengesloten zijn of gefragment.

Bestanden kunnen heel snel gelezen worden als ze zijn aangesloten.

### First fit:

Gebruikt het eerste beschikbare gat.

### Best fit:

Het kleinste gat waar het bestand in past

### Worst fit

Gebruik het grootste gat op de schijf

## Fragmentatie

Hoe langer de schijf in gebruik is hoe meer gefragmenteerd ze raakt.

Blokken, bestanden opdelen in blokken met vaste grootte

De blokken van een bestand hoeven niet naast elkaar staan.

Blokgrote hangt af van het bestandsysteem.

Groter:

In hoe minder delen een bestand opgesplitst wordt.

Hoe hoger de lees/schrijfsnelheid.

Hoe minder administratie er nodig is om het bestand terug te vinden.

Kleiner:

Hoe minder plaatsverlies aan interne fragmentatie

Vereist evenwicht tussen toegangssnelheid en plaatsverlies, Hangt af van bijv. hoeveel files er zijn, hoe groot ze gemiddeld zijn, hoe erg de interne fragmentatie is.

## Gelinkte lijsten:

De file descriptor bevat het adres van het eerste blok.

Elk blok bevat een verwijzing naar het volgende blok.

Adresinformatie neemt weinig plaats in beslag.

Direct toegang niet mogelijk.

## Indexblokken

Bij openen van een bestand wordt het indexblok ingeladen

Direct toegang is mogelijk

Indexblokken nemen veel schijfruimte in

## Extents

Nog sneller lezen en minder indexblokken.

Extent is een aantal opeenvolgendeblokken

File map

Aparte tabel met alle informatie

* Per blok op de schijf een entry in de tabel
* File descriptor wijst naar beginblok
* File map kan gebruikt worden om andere blokken te vinden.

De file map staat volledig in het werkgeheugen

* Directe toegang mogelijk
* Adresinformatie relatief compact
* Eenvoudig om bestand te verwijder, aan te maken, te verlengen met aaneengesloten blokken

## Directories:

Directory is zelf in feite een bestand dat verwijst naar meerder bestanden

### Eenvoudige lijst

* Zoeken is traag
* Eenvoudig
* Verwijderen is traag
* Toevoegen is snel

### Gesorteerde lijst

* Zoeken is snel
* Bestand verwijderen is traag
* Toevoegen is traag

### Hash tabel

* Vereist relatief veel plaats
* Verwijderen is snel
* Toevoegen is snel

### Gelinkte lijst

* Zoeken is traag
* Verwijderen is snel
* Toevoegen is snel

# NTFS

Maakt gebruik van een Master File Table (MFT)

* Bevat een entry per file
* Standaard 12.5% van de schijf voor MFT

Bevat:

* Filename
* File informatie
* Security descriptor

Een MFT entry is steeds 1KB.

Grote attributen worden non-resident gemaakt, d.w.z. buiten de MFT entry opgeslagen ▪+

Filename, timestamps, security descriptor Wat zijn altijd resident.

Directory-informatie word took in een MFT entry bijgehouden

Grote directories gebruiken B-bomen om gegevens op te slaan

## Ext2

Vier verschillende types blokken

* Bootblock
* Supervlock (bevat globale gegevens en instellingen)
* Inodeblokken (inode is gegevenstructuur die met een bestand geassocieerd wordt bevat metadata)

### Bestandsblokken

Inode heeft 15 pointers naar blokken

* De eerste 12 zijn directe links naar blokken
* De 13e pointer verwijst naar een blok dat directe links bevat
* De 14e pointer is een dubbele indirectie
* De 15e pointer is een driedubbele indirectie

# Hoofdstuk 5:

## Inleiding:

Een programma is uitvoerbare code.

Een proces is een programma in uitvoering.

### Multiprogrammatie:

Processor kan slechts 1 programma tegelijkertijd uitvoeren.

Rekentijd verdelen over processen

Processen kunnen van meerdere gebruikers komen: time sharing.

Scheduling: de BS zorgt voor zo goed mogelijk gebruik van resources

Responstijd: tijd tussen het geven van een opdracht en het moment dat de ontvangst van het antwoord begint

### Toestanden van een proces

Maximum 1 actief proces per processor (core)!

Gereed => actief: Scheduler activeert een proces op de processor

Actief => geblokkeerd: Porces doet bijvoorbeeld een system call, waarvoor het antwoord niet meteen beschikbaar is

Geblokkeerd => gereed: Gebeurtenis waarop het proces wacht vindt plaats

Actief => Gereed: Besturingssysteem onderbreekt actieve proces en zet het terug in wachtrij

Vaak I/O vaak in geblokkeerde toestand.

CPU gebonden processen zijn niet vaak in geblokkeerde toestand (geen I/O)

## Scheduling:

Processor zo efficiënt en goed mogelijk gebruiken

* Rechtvaardigheid, Geen enkel proces mag blijvend achteruit gesteld worden
* Doorvoer maximaliseren: Doorvoer = hoeveelheid “werk” per tijdseenheid
* Omlooptijd minimalisern: omlooptijd = tijd tussen het indienen van een proces en het voltooien ervan
* Responstijd minimaliseren
* Voorspelbaarheid
* Overhead beperken

Preemptief: kan een actief proces terug in de wachtrij worden gezet

## Scheduling-strategieën

### FCFS: first come first served

Vooral gunstig processen met langere CPU bursts (processen met korte bursts hebben vaak slechte relatieve omlooptijd)

Processen die vaak wachten op I/O moeten telkens achteraan aanschuiven.

### SJF: Shortest Job First

Starvation, processen met langere CPU bursts komen nooit aan de beurt.

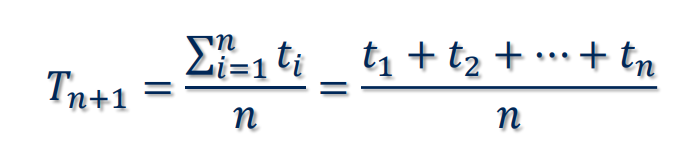
### Priority Scheduling

Eerst uitvoeren met de grootste prioriteit.

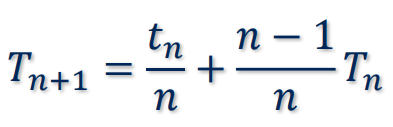
#### Voorspelling

tn = uitvoeringstijd van een vorige CPU burst

Tn = geschatte tijd voor volgende CPU burst



Of simpeler:



Elk proces krijgt een prioriteit P

Kies het proces met de hoogste prioriteit P, P=1/T

Welke prioriteit voor welk proces?

Intern: op basis van gebruik van resources

Extern: Processen van docenten > studenten

## Preemptieve scheduling

### Round Robin (RR):

First in, First Out processen hebben een tijdsquantum (maximale tijd)

Minder lange wachttijden dan FCFS

Geen starvation

Processen die vaak wachten op I/O moeten telkens achteraan aanschuiven

Vaak wisselen tussen processen is trager.

### Shortest Remaining Time (SRT):

SJF zorgt voor een minimale omlooptijd indien alle processen op voorhand klaar staan

SRT kan de processor terug afnemen indien er een kort proces bijkomt

## Hoofdstuk 6:

### Segmentatie

Het werkgeheugen wordt opgedeeld in segmenten.

Segment adres: basis van een segment.

Programma’s gebruiken meerdere segmenten (codesegment, datasegment, stapelsegment, extrasegment, …)

### Segmentdescriptor

Bevat eigenschappen van het segment, 8 bytes groot, wordt opgeslagen in de descriptorentabel.

Elke descriptor heeft een segmentselector, een nummer van (0000)h tot (1FFF)h, wordt bewaard in segmentregisters

### Schema:

1. Base address fetched
2. Offset must be between 0 and limit
3. Add base with offset

### Nadelen

Externe fragmentatie

Alles of niets: Het grootste deel van de programmacode wordt amper gebruikt

## Paginatie (paging)

Een programma wordt opgedeeld in pagina’s.

Het werkgeheugen wordt opgedeeld in frames

Pagina’s kunnen in het werkgeheugen geladen worden in vrije frames

### Page tabel

Page tabel houdt bij welke page in welke pagina zit.

32-bit adressen worden opgesplitst in:

* 20 bits voor het page nummer
* 12 bits voor de offset

Er zijn 220 pagina’s of 1048576 pagina’s

Elke page is 4 bytes dus 4MB

### Two-level page tabel

Naast de page table, ook nog een directory gebreuken (een lijst die verwijst naar page tables)

32-bit adres opsplitsen in:

* Page directory index (10 bits)
* Page number (10 bits)
* Offset (12 bits)

Page table = 210 = 4KB

Page directory 210 = 4KB

Totale overhead: minstens 8KB

* Veel efficienter

## Segmentatie en paging

Vanaf +- 18958 ondersteuning voor paging en segmentatie

…

## Virtueel geheugen

Elk proces denkt dat het volledige geheugenruimte kan gebruiken

Page table zorgt ervoor dat dit correct gebeurt.

Enkel pages die gebruikt worden, worden ingeladen

Er komt een interrupt als deze nog niet in het geheugen.

1. Vraagt de waarde aan de page table
2. Trap interrupt
3. OS vraagt naar de page
4. De page wodt in het geheugen geladen
5. Page tabel wordt gereset
6. De isntructie wordt opnieuw uitgevoerd

Swapping: file naar de schijf gezet en een andere file naar het geheugen

Traag

Thrashing: gebeurt wanneer er niet genoeg geheugen is om alle actieve pagina’s in te laden

Innefficient (te weinig RAM)