# Inleiding

## Waarom gebruiken we computers?

Toepassingssoftware schrijven en gebruiken.

Maar een klein deel van het hele computersysteem.

Software onderverdeeld in toepassingen en systeemsoftware

Systeemsoftware:

* Besturingssysteem
* Hulpprogramma’s

Hardware:

* Zie computersystemen

Besturingssysteem= een programma dat werkt als “tussenpersoon” tussen een gebruiker van een computersysteem en de computerhardware

Voornaamste doelen:

* Programma’s uitvoeren en beheren
* Gebruiksvriendelijkheid bevorderen
* Hardware efficient gebruiken

## Evolutie:

### ENIAC: 1950 – 1975.

Maakt gebruik van ponskaarten.

Heeft planningsproblemen want een menselijke operator moet fysiek de ponskaarten activeren => traag.

### Krachtigere computers

Gevolg: Fabrikanten leveren uitgebreide ingebouwde bibliotheken mee met het computersysteem die het computersgebruik beheren.

Bibliotheken voor:

* Opstarten van programmas
* Bijhouden van Facturatiegegevens
* Vaakgebruikte routines
* Beheer en bescherming tegen misbruik => eerste OS.

Gebruik van residente monitors:

* Resident: steeds aanwezig in het geheugen
* Monitor: staat in voor controle van het systeem

Programma uitvoeren:

* Spring naar eerste lijn
* Na afloop, spring naar residente monitor.

Eerste eenvoudige batch-besturingssystemen

* Programmas als globaal pakket aan computersysteem doorgeven
* Besturingssysteem voert programmas een na een uit zonder tussenkomst van de gebruiker
* Heeft efficientieproblemen: processor wordt niet gebruikt tijdens afhandelen van I/O

Oplossing? Multiprogrammatie:

* Meerdere programmas aanwezig in het geheugen
* Eerste programma moet wachtem? Start het tweede.
* Processor is veel meer gebruikt maar zit vast bij een oneindige lus.

Mulitprogrammatie-systemen zijn niet interactief.

Meerde gebruikers aan een computer.

* Processor is om de beurt bezig met een gebruiker
* Elke gebruiker heeft illusie het volledige systeem te bedienen
* Time sharing
* Wat als een programma nooit I/O doet?

Niet enkel wisselen van programma bij wachten, maar ook na bepaald verloop van tijd

* Interactief werken wordt mogelijk

Ook nog batch-processing op de achtergrond

Wanneer processorbelasting laag is.

### Soorten computers

Na een tijd werd er onderscheid gemaakt tussen soorten computers:

* Minicomputer 60s
* Mainframe grote centrale computers

Elk nieuw model had een nieuw besturingssysteem

* Zware inspanning
* Alle software moet opnieuw worden geschreven

IBM lanceert familie van computersystemen (System/360) zelfde besturingssysteem voor elk nieuw model.

### UNIX:

* Niet ontwikkeld door computerfabrikant
* Klein besturingssysteem
* Op elke hardware bruikbaar (mits kleine aanpassingen)
* iet geschreven in assembleertaal

### Personal Computer

Eind jaren 70 kwam de IBM microcomputer

Nieuw besturingssysteem: MS-DOS

* groot succes!
* Andere contructeurs nemen dit over
* Enorme markt met uniforme computers
* Toepassingssoftware voor MS-DOS werkt op bijna alle computers

Grote verspreiding en wordt zeer populair

1984: Macintosh SE eerste GUI

1985: Eerste versie van Windows (GUI voor MS-DOS)

Ontstaan van multitasking.

Eerste netwerken: LAN, WAN, Internet.

## Voornaamste Functies:

Gebruiksgemak

* Toegang tot apparaten eenvoudiger maken
* Informatie bewaren en overzichtelijk weergeven
* Operaties en gegevens beveiligen

Efficient gebruik van hardware

* Onderlinge communicatie tussen onderdelen organiseren
* Goede werking van onderdelen bewaken
* Hulpmiddelen rechtvaardig verdelen

Uitvoeren van besturingssysteem zelf = verloren tijd.

Hardware onafhankelijkheid: device drivers, uniforme interface aanbieden aan toepassingssoftware en gebruiker.

## Taken:

Toegang tot randapparaten regelen

* Conflicten vermijden
* Toegang geven aan bepaalde software

Beheer van werkgeheugen

* Programmas krijgen geen otegang tot elkaars geheugensegmenten
* Efficient gebruik

Procesbeheer

* Verdelen van processortijd
* Multiprogrammatie
* Multitasking

Communicatie

* Tussen processen
* Tussen computers via netwerken

Gebruikersinterface

* Gebruiksvriendlijkheid
* Commandline
* Tekstgebaseerd
* Grafisch

Veiligheid

* Onderdelen beschermen tegen onrechtmatig gebruik
* toegangscontrole

# Basismechanismen van een besturingssysteem:

## Kernel en gebruikerstoestand:

Programma over instructies uit en neemt ook in/out put van de hardware

De processor heeft twee toestanden:

### Gebruikerstoestand:

enkel gewone “veilige” instructies zoals voor Word/Excel/Steam/Chrome

### Kerneltoestand:

Alle instructies zijn beschikbaar, ook de niet veilige instructies.

Doel? Veiligheid

Dlicate operaties enkel door besturingssysteem, rest moet door de OS

OS kan tussen kernel mode naar user mode gaan.

Hardware kan van user mode naar kernel mode gaan.

## Interrupts:

Processor voert een programma tegelijk uit

* besturingssysteem is zelf een programma

Wanneer moet het besturingssysteem in actie komen?

* Signaal vanuit hardwre
* Vraag van hogergelegen software

OS is event driven, zit altijd in geheugen enkel actief wanneer nodig.

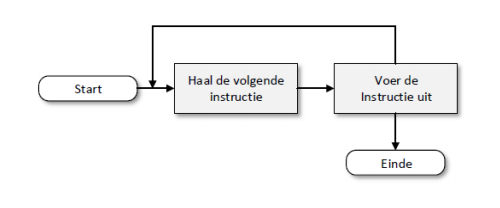
Stel een netwerkpakket komt binnen op de netwerkkaart

* Netwerkkaart stuurt een interrupt request naar de processor
* Processor onderbreekt het actieve programma

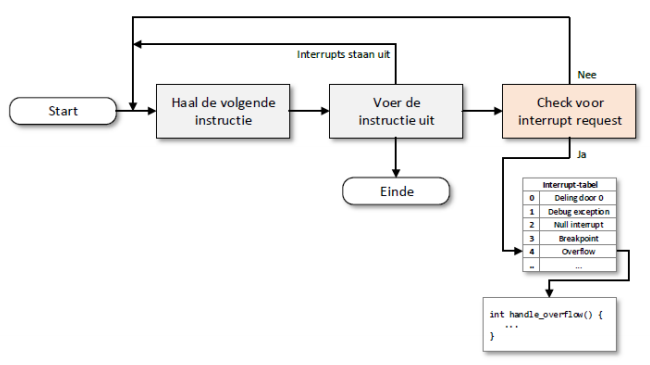
Geen interrupt request = geen onderbreking vqn het actieve programma

Wanneer checken of er een interrupt request is? Hardware ondersteuning!

Haal en uitvoeringscyclus zonder interrupt



Haal en uitvoeringscyclus met intterupt



Afhandeling van interrupt requests tijdelijk worden uitgeschakeld

* Niet controleren of er een interrupt request plaatsvindt
* Bijvoorbeeldtijdens het afhandelen van een interrupt

Interrupt request = elektrisch signaal

Veel mogelijk redenen:

* Deling door nul
* Leesopdracht is voltooid
* Gebruik van geprivilegieerde instructie in gebruikerstoestand

Elke interrupt moet anders afgehandeld worden

Interrupttabel

* Voor elk type interrupt een geheugenadres
* Interrupt vector

Wat als een leesopdracht voor de harde schijf klaar is?

1. Schrijf ontvangt een leesopdracht
2. Leesopdracht is voltooid => interrupt request naar interrupt controller
3. Interrupt controller geeft request door aan processor
4. Interrupt controller stuurt 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 ende processor wisselt naar gebruikerstoestand

Context switch = toestandswijziging van de processor

* Toestand van huidige programma opslaan
* Toestand van nieuwe programma inladen
* Volgende instructie uitvoeren

## Soorten interrupts

### Externe interrupts:

Externe interrupts zijn asynchroon

* Klok interrupts
* I/O interrupts
* Keyboard interrupts
* Hardware fouten

### Interne interrupts:

Interne = Synchrone

* Uitzondereingen (exceptions)
* Traps (software interrupts)

Interrupt treedt op => welke handler?

Eenvoudig bij exceptions en traps:

* Processor gaat conditiecode in toestandsbeschrijving na
* Programma dat trapt naar BS geeft informatie mee om juiste handler te kiezen

Wat bij hardware interrupts?

Bij hardware interrupts moet de processor eerst achterhalen waar de interrupt vandaan komt?

### Eerste manier

* Elke controller heeft eigen interruptlijn
* Welke hadler? Afhankkelijk van de interruptlijn waar het intrruptsignaal op binnenkomt
* Nadelen? Maximum aantal controllers

### Tweede manier

* Gemeenschappelijke interruptlijn voor alle controllers
* Controller geeft met 1 bit aan of deze een interrupt heft gestuurd of niet
* Welke handler? Elke controller afgaan oftewel software polling
* Traag

## Derde manier

* Gemeenschappelijke interruptlijn
* Gemeenschappelijke acknowledge-lijn
* Welke handler?
* CPU stuurt signaal naar eerste controller via ack lijn
* Controller stuurt identificatiecode als hij de interrupt veroorzaakt
* Of stuurt het signaal door naar de volgende controller
* Daisy Chaining
* Zeer snel

Kunnen meerdere interrupts tegelijk plaatsvinden? Ja

Eerst? Prioriteit op bassis van de interrupt latency

Asynchrone interrupt requests tijdens interrupt afhandeling

* Blokkeren, vlag instellen om asynchrone interrupts tijdelijk te blokkeren
* Prioriteren enkel interrupts accepteren met hogere prioriteit dan de huidige

## System Calls

Interrupts: Hardware <> Besturingssysteem

System calls: Software <> Besturingssysteem, Geimplementeerd als interrupts

* Opstarten en beeindigen van programmas
* Programma laten wachten op een gebeurtenis
* Toekennen en vrijgeven van geheugen
* Bestandbeheer: creeren en vernietigen van bestanden, lezen en sluiten, openen en schrijven, bepalen van toegang.
* Besturing en beheer van randapparaten
* Verwerven en afstaan, positioneren van een randapparaat, lezen van en schrijven naar een randapparaat
* Opnemen in of verwijderen uit de configuratie
* Uitwisselen van informatie
* Opvragen of wijzigen van informatie over systeemconfiguratie, uitgevoerde activiteiten, processen

## Read-system call in Linux

Read = C code

Wanneer een programma deze functie aanroept zal het volgende gebeuren

1. Oproepend prograamma zet waarden van de drie parameters op de stapel
2. Read functie wordt aangeroepen (sprong anar de code van read-functie)
3. Functie wet system call nummer in het juiste register en voer TRAP-instructie uit:
4. Interrupt service routine van het OS zoekt juiste routine op voor de system
5. Processor start systeemroutine
6. Systeem zet resultaat op voorziene plaats
7. Read functie ontvangt de gegevens, zet ze op hun plaats
8. Uitvoer oproepend programma gaat verder

# Bootprocess:

## Opstarten

### Voeding inschakelen

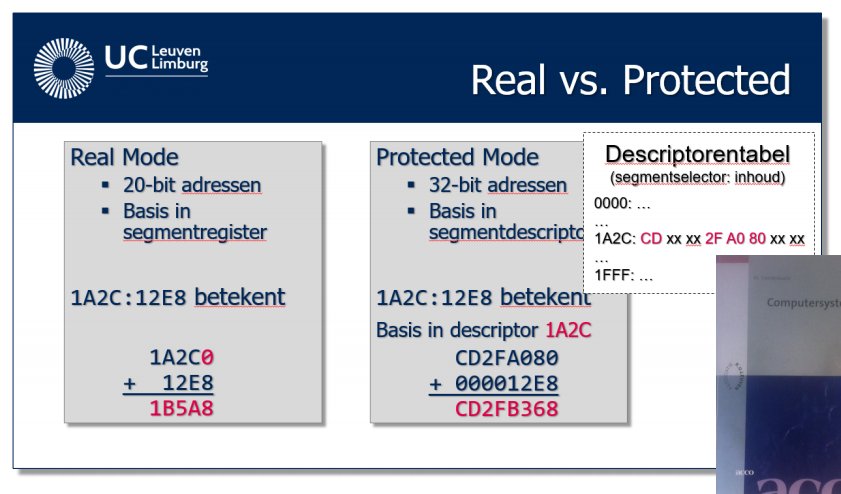
* Eigen spanningsniveaus controleren
* Stabiele spanning

### Moederbord

* Continu reset-signaal naar processor
* Power good?

### Processor

* Start in kernel mode
* Addressing in real mode.



* Op adres 0xFFFFFFF0

Werkgeheugen is leeg. We hebben een ROM chip op het moederbord met opstartcode die bij boot naar de memory wordt gekopieerd, Memory-mapped I/O!

### Opstartcode:

* Unified Extensible Firmware Interface
* Vervanging van BIOS
* Soort van mini-besturingssysteem met basisfunctionaliteiten (initialisatie van de hardware, inladen van besturingssysteem)

#### Voordelen van UEFI

* Ondersteuning voor grote schijven (tot 8 ZB)
* CPU-onafhankelijke architectuur
* CPU-onafhankelijke stuurprogramma’s
* Modulair ontwerp

#### Eigenschappen van UEFI

* OS geschreven voor BIOS werkt ook met UEFI

#### UEFI Services

* Boot services
* Runtime services

### Besturingssysteem inladen

* Waar staat het besturingssysteem?
* Vaak staat het besturingssystem op een harde schijf

## Schijven en Partities

### Partities

Een harde schijf kan worden opgedeeld in partities

Waarom?

* Snelste sporen van de harde schijf (de buitenste)
* Verschillende besturingssystemen met elk hun eigen partitie
* Scheiden van systeem en data bestanden

### GUID Partition Table

De partition table staat op de schijf zelf.

### Protetive MBR

* Compatibiliteit met oudere systemen
* Bevat een partitie ter grootte van de schijf

### Primary GPT Header

* Grootte van de schijf
* Beschikbare sectoren
* Unieke identifier
* Pointer naar secondery header

### Partition entrys

* Partitietype
* Unieke identifier
* Start-en eindsector
* Attributen (alleen-lezen, verborgen)
* Partitienaam

### Backups

## UEFI Boot Process

1. Architecture specific routines
2. Security phase
3. Pre-EFI initialization phase
4. Driver execution envirement
5. EFI runtime UP
6. Boot device selection
7. UEFI Bootloader
8. OS Bootloader

### Boot menu

Zoeken van de opstartpartitie

Bewoon vragen aan gebuiker(meerdere bootable devices)

Is een UEFI-applicatie

### UEFI Bootloader

Zoeken van de opstartcode op een bepaalde partitie

### Afhankelijk van partitietype OS boot loader

Opstarten van het IS op de partitie

Afhankelijk van besturingssysteem

BOOTMGR start dan windows verder op

# Bestandssystemen

## Bestanden

Fysiche ordening

Logische ordening

Bestand definitie: “Een verzameling gerelateerde gegevens, die door een gebruiker of een applicatie op een bepaalde manier geïnterpreteerd worden.”

# Procesbeheer

Een programma is een uitvoerbare code

Een proces is een programma in uitvoering.

Meerder instanties van hetzelfde programma en meerdere processen, meerdere procesbeschrijvingen.

Processor kan slechts 1 programma tegelijkertijd uitvoeren

Meer programma’s = meer processoren.

## Multiprogrammatie

Processen kunnen van meerdere gebruikers komen (Time sharing).

OS zorgt voor zo goed mogelijk gebruik van resources.

Processorgebruik maximaliseren.

## Toestanden van een proces

Maximum 1 actief proces per processor (core)

Process kan onderbroken worden om een ander proces te starten.

Een niet geactiveerd process kan niet altijd geactiveerd worden, als het nog geblokeert is gaat diet niet.

Het kan gereed worden gemaakt door dat een bepaalde gebeurtenis gebeurt.

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

Actief => geblokkeerd: Proces 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

### Procestypes

Vaak I/O vaak in geblokkeerde toestand

Zelden in geblokkeerde toestand = CPU-gebonden (video compression)

### Scheduling

Processor zo efficient en goed mogelijk gebruiken

Wat is efficient?

* Rechtvaardigheid: geen enkel proces mag blijvend achteruit gesteld worden
* Doorvoer maximaliseren: hoeveelheid werk maximaliseren
* Omlooptijd minimaliseren: tijd tussen het indienen van een proces en het voltooien ervan
* Responstijd minimaliseren

Kiezen opbasis van systeem noden

## Scheduling-strategieen

Enkel de processor zelf kan de processor vrij geven.

* First Come First Served (FCFS)
* Shortest job First (SJF)
* Priority Scheduling

### FCFS

Vooral gunstig processen met langere CPU bursts

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

FCFS is zeer geschikt voor systemen met voornamelijk lange, CPU-gebonden processen

Starvation, het kan dal lange processen “verhongeren” als ersteeds processen met korte CPU bursts bijkomen.

#### Voorspelling

ti= uitvoerinstijd van een vorige CPU burst

Ti = geschatte tijd voor volgende CPU burst

Wordt schatting voor gebruikt.

De tijd van meer recente uitvoeringsperioden moet een hogere invloed hebben op de schatting.

### Priority scheduling

Elk proces krijgt een prioriteit P

Kies het proces met de hoogste prioriteit P

Welke prioriteit voor welk proces?

* Intern: op basis van gebruik van resources
* Extern: processen van docenten > processen van studenten

## Starvation

Priority scheduling kan snel leiden tot starvation

Aging: prioriteit van een proces in de wachtrij laten stijgen, elk proces komt uiteindelijk aan bod.

## Preëmptieve scheduling

Round Robin (RR)

Shortest Remaining Time (SRT)

Priority Scheduling

Minder lange wachttijden dan FCFS

Geen starvation

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

Vaak wisselen tussen processen is trager.

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

Preemptieve priority scheduling is heel gelijkaardig aan niet preemptieve priority schedluing.

# Geheughenbeheer

## Segmenten

Het werkgeheugen wordt opgedeeld in segmenten, opeenvolgende bytes die bij elkaar horen.

Segment-adres, basis van een segment (adres van de eerste byte)

Programma’s gebruiken meerdere segmenten:

Codesegment (bevelen), datasegment (data), stapelsegment, extrasegment.

### Segmentdescriptor

Elk segment heeft een descriptor.

Eigenschappen, grootte, staat, basisadres

8 bytes groot

Wordt opgeslagen in de desriptorentabel

### Segmentselector

Elke descriptor heeft een uniek nummer van (0000)h tot (1FFF)h, de segmentselector.

Wordt bewaard in segmentregisters in de CPU

Insegmentregister staat een segmentselector