Septiminations of the protection of the protecti		- Starten/Beenden von Programmen
Language of the Service Servic		- Kommunikation von I/O geräten
Programme of theme on protein entire former on command without and the command of		
Parabet Parabet Parabet	Aufgaben eines Betriebssystems	- Userprozesse dürfen nicht mehr direkt auf Hardwaresysteme zugreifen. Nur noch über Syscalls erlaubt
Person P		
De Co Co Accessable Proposation of Processame on Survivo de Composation de Control Processame on Pro		
Visite to Richards Richards - Note 1 decree (Altermone) - Process of the Character (Altermone) - Process of the Character (Altermone) - Richards Recommendation - Richa	Syscalls	
- Vergrant Manors of Manimensory in control of the Control of Processor of Processo		Virtuellen Resourcen:
Indication management also CSS - Place (Fuedding according) Dishydrother for Processor particles of the Processor particles of		- Virtuell Memory (Mainmemory)
- Frincipals da ex Visional Control Co	Resourcenmanagment des OS	- Files (Harddrive storage)
November 1 - Fire Process bear makine Treated bealting in Fire Process bear makine Treated bealting in Fire Process bear makine Treated bealting in Fire Process beat all parts of the Process and the Process of the Proces		- Ermöglicht das Vortäuschen von Gleichzeitigkeit durch intelligentes Scheduling der verschiedenen Prozesse
En Circument Agrament Processor Control Pr	Prozesse	
Rabby De Primers State De Primers service D		Es Existieren folgenede Prozesszustände:
Booked - Dee Presents working and falsacy (Bill Pedigos (CV) or Commerce Management or Service Servi		- Ready -> Der Prozess läuft gerade nicht, könnte aber laufen
Regió Supportion of Der Process home infection before the close passer in this missurementy, sordern in Falucione		- Blocked -> Der Prozess wartete auf etwas (BSP: Belegtes I/O)
*** Assumptions on Whatehor Speciations and processors or whatehor Speciations and Posterior of the Abstraction of a Chapter of the Abstraction of Speciations and Statistics of Speciations of Speciatio	Prozesszustände	- Ready Suspendet -> Der Prozess könnte laufen befindet sich aber gerade nicht im Mainmemory, sondern im Harddrive
- Mesagement on of European September 1999 in evem System an algebrates - Absorber de physikation (Congestion de la Standerdes - Introduction of European September 1999 in evem System an algebrate (European September 1999) - Introduction of European September 1999 in evem System an algebrate (European September 1999) - Introduction of European September 1999 in evem September 1999 in the even simple challenge 29 filters - Introduction of European September 1999 in the even simple challenge 29 filters - Introduction of European September 1999 in the even simple challenge 29 filters 29		- Management vom Virtuellen Speicherraum pro Prozess
Interrupt Inte		- Managemnt von der Langzeitspeicherung in einem System aufgebaut als Dateien
Part	Files	
Ris g o Hostonic Name Justice (1997) For Immer vertices (1997) For Imm	Ports	- Ermöglicht es viele verschiedene Netzwerkverbindungen über eine einzige Leitung zu führen
Riji 5-15 minor weekige Zupffliscords OR Kermal und molgischerste lede set Sol in Roy B. Userprocesse in Ring 3 Procession Riging Bestimmen Michisen, die nicht in Ring J desidert der Historia in Kern Systems Min. maus er Contest Switches Morren Zustechen Processen und Ringan weekenin De Kork Kermal De Mondingsfach Kerman in Riging die desidert der Systems De Mondingsfach Kerman in Riging die desidert der Systems De Mondingsfach Kerman in Riging die desidert der Systems De Mondingsfach Kerman in Riging die desidert der Norden Ausstalieren Riging die Aus		
Sections National Nat		
De CS Kernal Ubersammt de Bastinutationen des Systems Min mass er Cointrol Monta between (Carbon Processes and Ringes werdnach) De Montajorien Kernal enthalt view for Aufgaber in Ring (p. de auch augnatique twention homiton. De Montajorien Kernal enthalt view for Aufgaber in Ring (p. de auch augnatique twention homiton. De Montajorien Kernal enthalt view for Aufgaber in Ring (p. de auch augnatique twention homiton. De Montajorien Kernal enthalt view for Aufgaber in Ring (p. de auch augnatique twention in Ring (p. de auch augnatique twention in Augnation and Leve Ringe auch Bernal enthalt view des excluter Besiehendschaften in Ring (p. de auch augnatique twention in Ring (p. de auch augnatique) and the De Montajorien in de Minder in de Ring (p. de auch augnatique) and de auch des Rings (p. de auch augnatique) and de auch de auc		OS Kernal und möglicherweise teile des OS in Ring 0. Userprozesse in Ring 3
Min muse of Cortext Switcher Switchers (Switchers Switchers Pricesses und Ringer verdented) Der Monoglische Kernal erhalt vision of nutber sing military state of until state of an explangation regin of, all each application in ring of the sing of the state of the	Protective Rings	
Der Kermal sebels at zere deuturch göder und messer, ober er entotig voll winger Kommunikation und vei weniger Content Stricthes als der Micro Kermal Schedundenberchers der de Marcharian besser, die er weniger Prücestudt im Filip of bit schedung versichen der Schedung in der Marcharian besser, ober er weniger Prücestudt im Filip of bit schedung versichen die Userprocesse in durch er Bit per schedung in der Kerpsten versichen der Schedung Prüceste Ring of Prüveligen bestehen, doch berüftigt der Inreh Kommunikation. Bit pr Kermal Artharig in der Erlegsten werd erst an der Kernal geschört und versich er der aus an das Filesystem versichen muss in der Schedung versichen muss in der Kerpsten versichen muss in der Schedung versichen siehen siehen muss in der Schedung versichen siehen muss in der Schedung versichen siehen	OS Kernal	Min. muss er Context Switches können (zwischen Prozessen und Ringen wechseln)
De Micro Kernal emitha' nur die abeablen Basteutierkontalisten en Rog Dur lageri jediche welten Erich auf berinden (1998) Abode Vierriche auf der Jame in den Schreibert, die weringer Florese Rog Of Prindigen bestezen, doch benörig werder Kommunikanie (2004) Abode Swirlich - Politich aus der jediche der Schreiber verbeilen wird od in der Kennta geschicht und von dort aus an das Filewysten, welches sich im Ring 3 belindet (2 Context Swirches) - Politichen des generation (2004) - Politichen des generation des Processes zuweisen - Hole Responsiveness - Politichen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie Stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie Stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie Stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie Stelen sich in wer ((1904) publichen Processes zuweisen - Hole Responsiveness - Leie Stelen sich in wer ((1904) publichen Weisen zuweisen - Hole Responsiveness - Hole Responsiveness - Processes Scheduling - Processes Scheduling - Processes Scheduling - Hole Leie Responsiveness - Scheduling erit des Stelen sich in wer (1904) - Hole Leie Responsiveness - Schedulin erit des Stelen sich in wer (1904) - Hole Leie Responsiveness - Sch	Marcal Richards Marcal	Der Kernal selbst ist zwar dardurch größer und massiver, doch er benötigt viel weniger Kommunikation und viel weniger Context Switches als der Micro Kernal.
Root Komal Sapt Kemal Antings and sas Filesystem with certa and ear Kernal geachloid uith von drift aus an das Filesystem, weiches sich im Ring 3 befindet (2 Context Switches)	Monolytnische Kernal	Der Micro Kernal enthält nur die absoluten Basisfunktionalitäten im Ring 0 und lagert jedliche weitere Funktion als Userprozess in äußere Ringe aus.
Power in use nicht umbedringt gesichert werden muss	Micro Kernal	
Ander Switch - Specialtem dies gesamten PSW - Steptide bildereigen Prozess kompten - Dez CO Net gitt price Prozess kompten - Dez CO Net gitt price Prozess kompten - Dez CO Net gitt price Prozess sempten - Dez CO Net gitt price Prozessor medicinent prozess gespeichert, wenn er nicht gerade fauft - SSP PSW und CPU status - Vertiere informationen des Prozessor vertieren - Prozessor am efficienteisten auslasten - Hoher Errozogitut - Prozessor am efficienteisten auslasten - Stenduling Errozogitut - Stenduling Errozogitut - Prozessor am efficienteisten auslasten - Stenduling Errozogitut - Prozessor auslasten - Prozessor auslasten - Stenduling Errozogitut - Prozessor auslasten - Prozessor		
Process Switch Slope of the Distribution process somplet Date of Selective (PCB) Date of Selective (picked Processes lone alignmen PCB an in PCB weeden Informationen zum Processe gespenichert, wenn er nicht gerade läuft Date of Selective (PCB) Zule Processes Controll Block (PCB) Zule Processes Controll Block (PCB) Zule Processes Controll Block (PCB) Zule Processes Controll Block (PCB) Zule Processes Controll Block (PCB) Zule Processes Controll Block (PCB) Zule Processes Switch Leben Sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Zele stehen sich im weg (Bsp.	Mode Switch	-> verbesserte Performance, da weniger gesichert werden muss
in PCB werden Informationen zum Prozess gespuchert, wenn er nicht gerade lauft - 85P-PSW und CPU status - 150P-PSW und CPU s	Process Switch	- Stoppt den bisherigen Prozess komplett
Zelei: Prozessoren möglichst efficien Prozesse zuweisen Hoher Throughput	Process Controll Black (PCB)	- Im PCB werden Informationen zum Prozess gespeichert, wenn er nicht gerade läuft - BSP: PSW und CPU status
- Hoher Throughput - Prozesco an effizientesten auslasten - Hote Responsiveness Ziele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Prozescustante können eingelatil werden in jarachdem wie achtell sie wieder benötigl/angefragt werden können: - Soher Tem (Bobedt, Ready, Munera) gat für Forzesse die demänchst benötigt werden - Scheduler entscheidet, welche Prozess stufen soll - User Orientet (Responsiveness) - System Overtiert (Effictivenesse) - System Overtiert (Effictivenesse) - System Overtiert (Effictivenesse) - System Overtiert (Effictivenesse) - Os entscheid, welcher Prozesse stufen soll - User Orientet (Responsiveness) - Os entscheid, welcher Prozesse stufen soll - User Orientet (Responsiveness) - Os entscheid, welcher Prozesse stufen soll - User Orientet (Responsiveness) - Os entscheid, welcher Prozesse stufen soll - Long Term (New und Exist) Prozesse die fisch sind oder fertig sind - meist User oder OS welcher Prozesse starten - Prozesse Scheduling - Preemtives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu - OS kann taulerde Prozesse starten in Time-Prozesse starten - Os entscheid, welcher Prozesse starten - Prozesse nicht von der CPU bis er fertig ist - Preemtives Scheduling - Noch Preemtive Scheduling Verfahren neinen Prozesse sincht von der CPU bis er fertig ist - Bis er sich selbst blockeit - Sie ann passieren, dass ein höherstufiger Prozess auf einen metigreren gestuffen Prozesse warten muss. Daher kann es passieren, dass ein Prozess, der zwischen diesen beließer Prozesse in der Prointiat fleis for dem höherstufiger Prozesses - Siehe duling FCFS - Einst Comer First Served Nicht Preemtive - Kein Vertrugem möglich - Scheduling SPN - Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert - Highest Response Ration Next - Kein Vertrugem möglich - Scheduling SRT - Der Prozess mit der pringsten todo Zeit wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cyces_walted)/cycles_todo - Vorteel. lange wartende Prozesse werden auch	Troccoc Control Block (FGB)	Ziele:
Ziele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Ziele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness) Prozesszutände können eingeteitt werden in jenachdem wie schnell sie wieder benötigt/angefragt werden können: - Short Term (Blocked, Ready, Running) gut für Prozesse die demaichst benötigt werden - Scheduler enscheidet, werden rescheidet, werden soll - User Orientiert (Responsiveness) - Shorteut ernscheidet, welcher Prozess ausgelagert werden soll - User Orientiert (Responsiveness) - Shorteut enscheidet, welcher Prozesse die demaichst benötigt werden - OS entscheidet, welcher Prozesse ausgelagert werden soll - Ung Term (Blocked Suspendet und Ready Sissendet (wobel das letztere blod ist)) gut für Prozesse die länger nicht mehr laufen müssen/können - OS entscheidet, welcher Prozesse ausgelagert werden soll - Ung Term (Revu mit Zusit) send oder fertig ist - Preemtives Scheduling rehreses dei frisch and oder fertig ist - Preemtives Scheduling verfahren nehmen einen Prozess nicht von der CPU bis er fertig ist - Preemtives Scheduling verfahren nehmen einen Prozess nicht von der CPU bis er fertig ist - Sis er sich selbst blockeit - Sis er sich selbst bl		- Hoher Throughput
Prozesszustánde können eingetelit werden in jenachdem wie schnell sie wieder benötigtvangefragt werden können: - Short Term (Blöcked, Ready, Running) gut für Prozesse die demarkst benötigt werden - Short Term (Blöcked, Ready, Running) gut für Prozesse die demarkst benötigt werden - Short Term (Blöcked Sprosinerens) - System Orientiert (Effektivestes nutzen der CPU) - Medium Term (Blöcked Sprosinerens) - System Orientiert (Effektivestes nutzen der CPU) - Medium Term (Blöcked Sprosinerens) - System Orientiert (Effektivestes nutzen der CPU) - Medium Term (Blöcked Sprosinerens) - System Orientiert (Effektivestes nutzen der CPU) - Medium Term (Blöcked Sprosinerens) - System Orientiert (Effektivestes nutzen der CPU) - Medium Term (Blöcked Sprosinerens) - De system Orientiert (Effektivestes nutzen der CPU) - Medium Term (Note of Sprosinerens) - Prozess Scheduling - Prezentives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu - OS kan nutzen der Dezesse abezteren (Bear) schleben) - Kein Prozess kann die CPU Monopolisieren - Nicht Preemitve Scheduling Verfahren nehmen einen Prozesse nicht von der CPU bis er fertig ist - Sie er sich selbst blockiert Es kann passieren, dass ein höberstufiger Prozess auf einen niedrigeren gestuften Prozess warfen muss. Daher kann es passieren, dass ein Prozess, der zwischen diesen beiden Prozesse in der Priorität legt vor dem höherstufiger elledigt wird. Das nennt man Priority Inversion - Losung Der niedrigere Prozesse sied tille Priorität des höhereingestuften Prozesses - Friet Come First Served Nicht Preemitve - Kann Verhungern möglich - Scheduling SPN - Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert - Viriette i Lange warfande Prozesse verden auch Priorisiert. Der Response ratio ergit sich aus (cycles_nodo+cycles_walted)/cycles_todo - Vorteit. Lange warfande Prozesse verden auch Priorisiert. Der Prozess kommt dran - Scheduling SRT - Der Prozess mit der geringsten todo Zeit wird priorisiert. - Der Prozess		
Short Term (Blocked, Ready, Running) gut für Prozesse die demnächst benötigt werden Scheduler entscheidert, welcher Prozess laufen soll - User Orienterie (Responsiveness) - System Orienterie (Effektivetetes runzen der CPU) - OS entscheidet, welcher Prozesse sausgelagertivenglagert werden soll - Long Term (New und Exist) Prozesse das mit eine Men vom Exist sind oder fretig sind - meist User oder OS welcher Prozesse sausgelagertivenglagert werden soll - Long Term (New und Exist) Prozesse statet - Preemtives Scheduling bereichnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu - OS kann laufende Prozesse and einen Prozesse sind ter Scheduling verfahren enter Prozesse sind exist in Scheduling verfahren nehmen einen Prozesse sind ter CPU bis er fertig ist - Sie er sich sebbst blockhold Sie er sich sebst blockhold Sie er sie sie er sich sebst blockhold Sie er sie sie er sich sebst blockhold Sie er sie		Ziele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness)
- User Orientier (Responsiveness) - System Orientier (Effektivestes nutzen der CPU) - Nedium Term (Blocked Suspendet und Ready Suspendet und Ready Suspendet worden soll - Long Term (New und Exist) Prozesse die langer nicht mehr laufen müssen/können - O'S entscheldet, welcher Prozess ausgegerteingelagert werden soll - Long Term (New und Exist) Prozesse die firsch sind oder fertig sind - miest User oder OS weichter Prozesse site stantet - OS kam laufende Prozesse dabetzen (in Ready Scheduling Verlanden von Scheduling Verlanden einen Prozess nicht von der CPU bis er fertig ist - Bis er sich selbst blooklert - Die er sich selbst blooklert - Bis er		- Short Term (Blocked, Ready, Running) gut für Prozesse die demnächst benötigt werden
- Medium Term (Blocked Suspendet und Ready Suspendet (wober dias Lettere blod ist)) gut für Prozesse die langer nicht mehr laufen müssen/können - OS entscheidet, werher Prozess ausgelagertreingelagert werden soll - Long Term (New und Exist) Prozesse die frisch sind oder fertig sind - meist User oder OS welcher Prozesse staret Prozentives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu - OS kann laufende Prozesse sähertset - Nicht Prozesse stant die CPU Monopolisieren - Nicht Prozesse sahn die CPU Monopolisieren - Nicht Prozesse sich der Ausschlaft (in Ready schieben) - Es kann passieren, dass ein Prozesse sicht von der CPU bis er fertig ist - Bis er sich selbst blockliert - Bis dez Besprach von der Prozesse sich der Prozesse sich der Prozessese - Bis dez Besprach von der Prozesses werden aus er berintat des höhereingestuften Prozesses - Scheduling FCFS - Bindste Prozess Net Nicht Preemity - Verhungern möglich - Ber Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_lodo+cycles_walled)/cycles_todo - Vorteit. lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran - Scheduling SRT - Der Prozess mit der geingsten todo Zeit wird priorisiert - Bis er der Bis der Scheduling Scheduling SRT - Bis er der Bis der Scheduling S		- User Orientiert (Responsiveness)
- Long Term (New und Exist) Prozesse die frisch sind oder fertig sind - meist User oder Oz welcher Prozesse startet - Preemtives Scheduling - Preemtives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu - OS kann laufende Prozesse skartetet - Preemtives Scheduling - Nicht Preemtive Scheduling - Nicht Preemtives Scheduling - Nicht Preemtive Scheduling - Prozess in der Priorität liegt vor dem höherstufigen erledigt wird. Das nennt man Priority Inversion - Nicht Preemtive - Kein Vertunger möglich - Scheduling FCFS - Einfache queue Scheduling FCFS - Einfache queue Scheduling SPN - Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert - Wein Vertungern möglich - Scheduling SPN - Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert - Wein Vertungern möglich - Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cycles_walted)/cycles_todo - Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran - Scheduling SRT - Der Prozess mit der geringsten todo Zeit wird priorisiert - Scheduling SRT - Der Prozess mit der geringsten todo Zeit wird priorisiert - Round Röbin - Preemtiv - Kein Vertungern möglich - Round Röbin - Preemtiv - Kein Vertungern möglich - Round Röbin - Preemtiv - Kein Vertungern möglich - Round Röbin - Preemtiv - Kein Vertungern möglich		- Medium Term (Blocked Suspendet und Ready Suspendet (wobei das letztere blöd ist)) gut für Prozesse die länger nicht mehr laufen müssen/können
- Preemtives Scheduling berechnet das Scheduling nach jedem Timer-interrupt für alle Prozesse neu - OS kann laufender Prozesse absetzen (in Ready schieben) - Nicht Preemtive Scheduling - Nicht Preemtive - Nicht Preemtiv - Nicht Preem		- Long Term (New und Exist) Prozesse die frisch sind oder fertig sind
- Kein Prozess kann die CPU Monopolisieren - Nicht Preemtive Scheduling - Nicht Preemtive Scheduling - Bis er sich selbst blocklert - Bis er selbs	Frozess scrieduling	- Preemtives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu
- Bis er sich selbst blockleirt Es kann passieren, dass ein höherstufiger Prozess auf einen niedrigeren gestuften Prozess warten muss. Daher kann es passieren, dass ein Prozess, der zwischen diesen beiden Prozessen in der Priorität liegt vor dem höherstufigen erledigt wird. Das nennt man Priority Inversion Lösung: Der niedrigere Prioritäts Prozess erbt die Priorität des höhereingestuften Prozesses First Come First Served Nicht Preemtive Kein Verhungern möglich Scheduling FCFS Einfache queue. Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cycles_walited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Röbin Preemtiv Kein Verhungern möglich Per Prozess mit der geringsten todo Zeit wird priorisiert Kein Verhungern möglich Round Röbin Preemtiv Kein Verhungern möglich		- Kein Prozess kann die CPU Monopolisieren
der zwischen diesen beiden Prozessen in der Priorität liegt vor dem höherstufigen erledigt wird. Das nennt man Priority Inversion Lösung: Der niedrigere Prioritäts Prozess erbt die Priorität des höhereingestuffen Prozesses First Come First Served Nicht Preemtive Kein Verhungern möglich Scheduling FCFS Einfache queue. Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Nicht Preemtiv Verhungern möglich Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Preemtives/Nicht Preemptives Scheduling	- Bis er sich selbst blockiert
First Come First Served Nicht Preemtive Kein Verhungern möglich Scheduling FCFS Einfache queue. Shortest Process Next Nicht Preemtiv Verhungern möglich Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Verhungern möglich Round Robin Preemtiv Kein Verhungern möglich		
Nicht Preemtive Kein Verhungern möglich Einfache queue. Shortest Process Next Nicht Preemtiv Verhungern möglich Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Priority Inversion und Inheritance	
Shortest Process Next Nicht Preemtiv Verhungern möglich Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cycles_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich		Nicht Preemtive
Nicht Preemtiv Verhungern möglich Scheduling SPN Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_walted)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Scheduling FCFS	
Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cycles_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich		Nicht Preemtiv
Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Scheduling SPN	-
Schedulung HRRN Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich		Nicht Preemtiv
Shortest Remaining Time Preemtiv Verhungern möglich Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Schedulung HRRN	
Scheduling SRT Der Prozess mit der geringsten todo Zeit wird priorisiert Round Robin Preemtiv Kein Verhungern möglich	Concusting and Civil	Shortest Remaining Time Preemtiv
Preemtiv Kein Verhungern möglich	Scheduling SRT	
Scheduling RR Prozesse werden im Kreis abgefahren		Round Robin Preemtiv
	Scheduling RR	Prozesse werden im Kreis abgefahren

	More Resources Available to Schedule
	Möglichkeit 1: - Prozesse bleiben auf 1 Prozessor (weniger Probleme mit der Cache verwaltung)
	- Mögliche ungleichverteilung von Prozessen
	Möglichketi 2: - Prozessoren werden als Pool angesehen
	- Prozess kann auf vielen Prozessoren laufen - Cache muss gehandlet werden bei Prozessorwechsel
	Real-Time-Scheduling
M Edward Dall Town October 1	- Der Korrekte Systemstatus hängt von der Einhaltung der Deadlines an
Multipleprozessors and Real-Time Scheduling	- Scheduling muss mit betrachtung und Priorität der Deadlines geschehen Zwei Methoden:
OS Memory Management	Paging - Unterteilung der Memorysegmente in gleich große Einheiten Segmentierung - Unterteilung der Memorysegmente in Varriable lange Einheiten
OS Wellory Wallagement	- Feste größe der Memorysegmente
	- Auslagerung von Pages ist einfach, da jede Page gleich groß ist - Verteilung von Pages kann nicht sequenziell sein
Pagin	- Bei Pages kann interne Fragmentierung entstehen - Memory ist nicht Perfekt ausgenutzt, da vielleicht eine Page nicht komplett gefüllt ist doch selben Platz braucht
	- Segmentgröße Varriert nach benötigtem Platz - Blöcke müssen aber sequenziell geschrieben werden
Segmente	- Externe Fragmentierung entsteht -> Beim auslagern von Segmenten können Lücken entstehen, die von weiteren Segmenten nicht Perfekt gefüllt werden können
	Wann muss eine neue Page/Segment geladen werden Zeitliche- und Örtliche-Lokalität
	- Wo werden Daten hingepackt die geladen werden? - Bei Pages einfach, da jede Lücke groß genug ist
Probleme mit Virtueller Memory	- Bei Segmenten muss erst benötigter Platz gefunden werden
	- First Fit - erster Platz, der groß genug ist für das Segment, wird belegt
	- Next Fit - erster Platz, der groß genug ist für das Segment, wird belegt. Untschied ist nur, dass die Suche nach neuen Plätzen immer dort beginnt wo das vorheriger abgelegt wurde
Dynamiasha Plaziarunga Algarithman	- Best Fit
Dynamische Plazierungs Algorithmen	- Platziere das Segment in die Stelle, die am besten Passt - FIFO
	- LIFO - LRU (Least recently used)
	- LFU (Least frequently used)
	- LRD (Least reference density) - Verbindung von LRU und LFU
Beispiele für Swapping Algorithmen	- Ratio zwischen wann es reingeladen wurde und wann es zuletzt benutzt wurde Der Page Table wird vom OS angelegt und behält die Position im Memory für jede Pages eines Prozesses.
	Will jetzt ein Prozess auf eine bestimmte Adresse zugreifen muss zunächst im Page Table nachgeschaut werden, ob die Page in der Memory ist.
Page Table	- Wenn ja -> Nehme die Position der Page und Addiere den Offset der Adresse - Wenn nein -> Lade Page ins memory und wiederhole "Wenn ja"
	Virtuelle Adresse besteht aus:
	- Pagenumber - Offset
	Wie in "Page Table" erklärt: Wenn die Page sich im Pagetable befindet wird zunächst der Basepointer zu dieser Adresse geholt. Daraufhin wird der Offset Addiert,
Umrechnung der Virtuellen Adresse	um die korrekte Adresse zu bekommen
	Page Table können eine Gewisse größe übersteigen, sodass sie nicht mehr in einer Page gehalten werden können. Um dies zu unterstützen kann man Hierarchien von Page Tables erstellen. Im ersten Page Table (root table) steht dann nur die Adresse der Page, die entweder direkt die Position der Page mit dem Datum enthält
Hierarchische Page Table	oder die Position des nächsten Page Tables. Dies wird solange gemacht, bis die Page mit dem Datum gefunden wurde. Im schlimmsten fall müssen immer n-1 Pages geladen werden (alle bis auf den root table, da auf diesen bei jedem Zugriff zugegriffen werden muss)
	Der Translation Lookaside Buffer (TLB) enthalt die direkte Position des Datum innerhalb des Memory, wenn sie sich dort befindet und bereit angefragt wurde.
	Bei einer Anfrage wir zunächst im TLB geschaut, ob die Virtuelle Adresse dieses Prozesses schon einmal angefordert wurde: Ja -> Die logische Adresse befindet sich im TLB und kann direkt ausgelesen werden
	Nein -> Es wird geschaut, ob die Page sich im Hauptspeicher befindet: Ja -> Die Adresse wird umgerechnet, geladen, und die übersetzte Adresse im TLB gespeichert
Translation Lookaside Buffer	Nein -> Die Page wird aus dem Hintergrundspeicher geladen und dann wird die Adresse im TLB gespeichert
	- Programmed I/O - Die CPU beschäftigt sich aktiv mit dem Auslesen der I/O daten ins Memory und blockiert somit alle weiteren Prozesse
	- Interrupt I/O - I/O Gerät läd die Daten in den geteilten Speicher und meldet sich bei der CPU zurück, sobald alle Daten geladen wurden
I/O Managment	Direct Memory Access (DMA) Extra Stück Hardware, welches sich mit dem Auslesen der Daten des I/O's beschäftigt
	- I/O Device meldet sich bei der CPU, dass es Daten besitzt, mittels eines Interrrupts
	- CPU stößt das Laden der Daten über den DMA an und behandelt daraufhin weitere Prozesse - DMA läd Daten Blockweise oder als Stream (je nach implementation) in den Memory
Direct Memory Access	- Nach beendigung des ladens eines Blocks meldet sich der DMA Controller bei der CPU das er fertig ist. Single Bus - Deatached DMA
	- Ein einziger Bus über den alles verbunden ist.
	- Sehr einfach aber Sehr uneffizient Single Bus - Integrated DMA
	- Alles läuft über einen Bus jedoch jede I/O (group) besitzt einen eigenen DMA - Sehr effizient doch kostspielig
Direct Memory Access Implementationen	I/O bus
Shoot mornory Access implementationell	- Eigener Bus über den alle I/O's angelegt sind der über einen DMA an den Hauptbus verbunden ist Dadurch, dass I/O eindeutig langsamer ist als die CPU bietet sich hier das Prinzipt des Buffering an:
	- Die I/O läd mehrere Daten vor währenddessen die CPU noch alte Daten verarbeitet
	Blockorientiert: - Daten werden in festen Blocks eingelesen und ausgelesen
I/O Duffering	Streamorientiert:
I/O Buffering	Datenblöcke habe keine feste größe und können beliebig ein/ausgelesen werden - FIFO
	- SSTF - Der Block wird als nächstes geschrieben wofür sich der Schreibkopf an wenigsten bewegen muss
HO Oshari Par ANGEST ST	- Scan
I/O Scheduling (Writing into Memory)	- Der Schreibkopf bewegt sich stehts von einer Seite zur nächsten und bearbeitet Anfragen immer nur in eine Richtung RAID wird genutzt um ein mögliches Ausfallen einer Festplatte abzufangen. Hierzu werden z.B in RAID Level 5 die Blöcke verteilt auf verschiedene Disks geschreiben.
PAID	Pro Festplatte gibt es eine Disk die für die Parität zuständig ist. Fällt eine Festplatte aus kann die Disk über die Parität wiederhergestellt werden.
RAID	Die Parität entsteht darduch, dass man alle Blöcke XOR nimmt. Fällt eine Festplatte aus kann durch das XOR über alle und der Parität alle wieder hergestellt werden.
	Files bzw. jegliche im Filesystem dargestellten Objekte sind in Inodes Codiert.
	Diese Inodes enthalten Metadaten wie:
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte)
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Prosition in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein.
Inodes	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält
Inodes Inode Struktur	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Prosition in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. - Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält - Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. - Zwei verschiedene Referenzen auf eine Inode können auch existerien
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. - Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. - Zwei verschiedene Referenzen auf eine Inode können auch existerien - Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke abgespeichert. Hierzu gibt es 3 Varrianten:
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) - Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält - Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes Zwei verschiedene Referenzen auf eine Inode können auch existerien - Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke abgespeichert. Hierzu gibt es 3 Varrianten: - Sequilized Allocation - Eine Datei wird immer hintereinander abgespeichert. Blöcke müssen Sequenziell geschreiben werden
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. Zwei verschiedene Referenzen auf eine Inode können auch existerien Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke abgespeichert. Hierzu gibt es 3 Varrianten: - Sequilized Allocation - Eine Datei wird immer hintereinander abgespeichert. Blöcke müssen Sequenziell geschreiben werden → Easy Random Access (Basepointer + Offset) aber zu unflexible Struktur - Chained Allocation
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. Zwei verschiedene Referenzen auf eine Inode können auch existerien Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke dagsepeichert. Hierzu gibt es 3 Varrianten: - Sequilized Allocation - Eine Datei wird immer hintereinander abgespeichert. Blöcke müssen Sequenziell geschreiben werden -> Easy Random Access (Basepointer + Offset) aber zu unflexible Struktur
	Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. Zwei verschiedene Referenzen auf eine Inode können auch existerien Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke abgespeichert. Hierzu gibt es 3 Varrianten: - Sequilized Allocation - Eine Datei wird immer hintereinander abgespeichert. Blöcke müssen Sequenziell geschreiben werden -> Easy Random Access (Basepointer + Offset) aber zu unflexible Struktur - Chained Allocation - Ein Block wird egal wo im Hintergrundspeicher abgelegt und verweißt auf den nächsten Block

	Der Boot Prozess findet in einer Kaskade statt:
	- BIOS initialisierung - Boot stage 1
	- Load and start Bootstage 2 - Boot stage 2
	- Load Kernal - Kernal
	- Switch into Protected Mode
	- Start first Prozess (init) - Init Porzess:
Boot Prozess	- Start System functionalitys
UEFI/EFI	Der BIOS wurde durch den neuen Standard UEFI/EFI ersetzt um gewisse Sicherheitlücken zu stopfen. Unteranderem kann dadurch nur noch software im nicht Protected ausgeführt werden, die eine besondere Signatur besitzt
UNIX Daemons	Daemons sind Programme die durchgängig im Hintergrund laufen ohne Interaktion des Benutzers zu benötigen
	Passive Attak: - Belauschen von Informationen ohne Handlung auf das System
	Aktive Attak:
Attak Types Password Creation	- Maipulation von Daten Der Passowort String wird mit einem Salt durch eine sehr langsame Hashfunktion gejagt und abgespeichert. Beim einlogen werden nur die Hashes überprüft
Backdoors	Schlupflöcher innerhalb von Programmen um sich Zugang zu verschaffen. Diese müssen vom Entwickler eingebaut worden sein
Logic Bomb	Wenn sie ausgeführt wird wird das System so manipuliert, dass es zum Absturz des Programms/Systems kommen kann
Trojanisches Pferd	Schleuse Schadsoftware mit Nützlicher Software ein
Virus Bots	Infiziert einen Rechner und Repliziert sich selbst Software die genutzt werden kann um Systeme femzusteuern
BUG	Johnste die gemitzt werden die üb ann un opsierine entzüsteleum Besteht aus 7 layern die üb ann die üb Annach entzüsteleum miteinander Kommunizieren:
	7. Application 6. Presentation
	5. Session
	4. Transport Layer 3. Network Layer
ISO/OSI Modell	2. Datalink Layer 1. Physical Layer
	Ähnlich zum ISO/OSI Modell, wird jedoch, anders als das ISO/OSI Modell, in realität benutzt:
	5. Application 4. Transport Layer
	3. Internet Layer
TCP/IP Modell	Network Access Network Access
	Der Physical Layer befasst sich damit das Digitale Signal in ein Physicalisches Signal Umzuwandeln. Hierzu bekommt es eine gewisse Bandbreite zugeteilt auf der es Arbeiten kann. Je höher die Bandbreit-Range ist, desto genauer kann das Signal übertragen werden.
	Auf dieser Ebene findet keine direkte Fehlerbehebung statt sondern nur eine kleine Minderung des Rauschens indem bei jedem Hop über ein weiteres Gerät
	das Signal einmal vom Physicalischen zum Digitalen zum Physicalischen übersetzt wird um kleine Anomalien zu eliminieren.
Physical Layer	Zu den Daten muss zusätzlich eine Taktung übertragen werden, damit der Reciever am ende Weiß, wann jeweils ein Hoch (1) bzw. Tief (0) ausgelesen wurde. Durch dem Clockdrift-Phänomän muss dieser ständig wieder angepasst werden. Eine Lösung hierfür ist die Manchester Codierung
r flysical Layer	In der Manchaster Codierung wird die Information der Nachicht und die Clockinformation gleichzeitig übertragen.
	Eine 0 bzw. eine 1 wird nicht mehr über ein Hoch oder Tief codiert, sondern über den Wechsel von Hoch->Tief bzw. Tief-Hoch. Dadurch entsteht selbst bei einem gleichbleibenen Signal bsp: 100000000000000000000000000001 die Information des Clockstatus erhalten.
Manchaster Codierung	Problem: Da mehr Information gleichzeitig übertragen wird kann nur noch die halfte der Informationen übermittellt werden (geschwindigkeit = Bandbreite/2)
Analoges Signal	AM = Amplituden Modeliert FM = Frequenz Modeliert
7 maiogos olginai	Der Datalink Layer kümmert sich um:
	- die Einteilung der übergeben Bits in Frames - Error Erkennung/Behebung
	- Acknowledgement Protocol
Datalink Layer	Flow Control Mitteilung and Sender, in welcher Geschwindigkeit er Packete schicken kann/darf
	- Unterteile Bits in Frames
	- Frames haben eine Gewisse Bitfolge als Start/End - Das ende ist oft eine CRC Prüfsumme
	- Header besteht aus Flagbits - Um sicher zugehen, dass die Payload keine Headerconfiguration besitz wird Bitstuffing betrieben
Datalink Layer - Framing	- Bsp: Header = 11111 Message 1011111001 => 10 111101001 (setze 0 nach 4 aufeinander folgenden 1sen)
Error Control:	Beim versenden der Nachicht wird eine CRC Prüfsumme gebildet und als Trailer der Payload hinzugefügt. Stimmt der an der Destination übertragener CRC Wert nicht mit dem Übertragenen Wert überein kann versucht werden das Package zu recovern, ansonsten wird es gedroppt
	Um über das selbe Medium mehrere Signale zu übermitteln muss man diese nach einem Speziefischen verfahren versenden um sicher zu stellen, dass jedes Signal unique ist Hierzu gibt es verschiedene Varianten:
	- Time Division Multiplexing:
	- Man verschickt Datenpackete Zeitverschoben über das selbe Medium um Eindeutigkeit zu erhalten - ALDHA und Slotted ALDHA sind zwei Varrianten dafür
	- Frequenzy Division Multiplexing: - Man nutzt verschiedene Frequenzen um verschiedene Signale zu Codieren
	- Code Division Multiplexing:
	- Space Division Multiplexing:
Medium Access - Multiplexing	ne Custom versebiskon dana Potes vana sia valeba kakari. Es avietint kaisa Caardisianus
ALOHA	Das System verschicken dann Daten wann sie welche haben; Es existiert keine Coordinierung. Überlappen sich zwei Signale (auch nur um ein Bit) wird das gesamte Packet verworfen und als Ungültig gekenzeichnet.
	Das System verschickt Daten wenn es welche hat, hält aber dabei ein Zeitrahmen ein. Daraus folgt, dass sich Packete entweder ganz oder gamicht überschneiden und nicht mehr Halbüberschneidungen stattfinden.
Slotted ALOHA	Dies benötigt jedoch eine Timerabstimmung unter den verschiedenen Systemen
Repeater/Hub	Nimmt Physicalische Signale, übersetzt sie einmal von Physikalisch -> Digital -> Physicalisch und schickt sie wieder verstärkt weiter
Switch	Genutz um mehrere Terminals untereinander zu verbinden ohne ein Router Netzwerk zu benutzen. Hierbei Identifizieren sich die verschiedenen Geräter mit ihrer MAC Zu begin weiß ein Switch nicht, welche MAC Adresse an welchem Port verbunden ist. Dies wird erst zugeteilt, wenn dieses Gerät eine Information schickt.
Switch Bookward-I	Schickt jetzt z.B. A eine Nachicht nach B, weiß der Switch jedoch nicht auf welchen Port es diese Nachicht schicken soll. Also wird zunächst die Nachicht an alle Terminals
Switch - Backwardslearning	verschickt. Meldet sich B nun zurück kann diesem Port die Adresse von B zugeteilt werdne und bei der nächsten Nachicht für B wird nur noch an diesen Port geschickt. Router werden genutzt zur Kommunikation innerhalb eines Netzwerk bzw. zwischen Netzwerken (Diese Router werden Gateways genannat).
Router	Innerhalb eines Netzwerkes werden Intra-Domain-Routing-Methoden benutzt. Zwischen Netzen Inter-Domain-Routing-methoden.
	Für Intra-Domain-Routing kann die Methode Open-Shortes-Path-First (OSPF) genutzt werden. Vorraussetzung ist, dass jeder Router das gesamte Routing Netz als Information vorhanden hat. Dies geschicht dadurch, dass ein Router, sobald er einen neuen Nachbarn hat, oder ein Nachbar wegfällt, diese Information an alle seiner Nachbarn weiterleitet
	die dann durch das gesamte Netz Propagieren. Die "länge" einer Verbindung ergibt sich durch das Messen eines Round-Trip signals.
Router - Intra-Domain-Routing - OSPF	Wird nun eine Anfrage geschickt um innerhalb dieses Autonomen-Netzes zu kommunizieren wird der Dijkstra Algorithmus genutzt um den schnellsten Weg zu finden.
Router - Inter-Domain-Routing	Für die Kommunikation zwischen verschiedenen Netzwerken wird an den verschiedenen Gateways ein Boarder-Gateway-Protocol verwendet. Jeder der Router besitzt eine Tabelle mit Präfixen und dazugehörigen Bitmasken. Unser Packet wird an den Router weitergeleitet, der die beste Metrik für unser IP besitzt.
	Das IP wird genutzt um jedem Terminal in einem Netwerk eine eindeutige Adresse zuzuweisen. Hierbei gibt es entweder IPv4 oder IPv6
	Im Netzwerklayer werden zwei wertere Protokolle verwendet um Zusatzinformation zu bekommen: ARP und ICMP
Internet Protocol (IP)	Das IP ist Hierarchisch aufgebaut. Das heißt verschiedene Netwerke werden in verschiedene Gruppen zusammengefasst. Je höher die Gruppe desto weniger IPs besitzt diese. Die höchste ist die A Gruppe und es steigt auf bis D (E ist reserviert für die zukünftige Nutzung)
	Das ARP Protokoll wird verwendet um die MAC Adresse eines gewissen Terminals herrauszufinden. Hierzu wird an alle Teilnehmer des Systems gerufen:
ARP	"Wer von euch hat diese IP?!?" und derjenige der als erstes Antwortet wird genommen. Falls keiner in diesem System die IP besitzt meldet sich der Router mit seiner MAC
ICMP	Das ICMP Protokoll wird verwendet um eine Rückmeldung an den versender zu schicken, falls das Packet zwischendurch verloren geht/die Time-To-Live abläuft. Dies ist ein kann und kein muss!
	Das TCP Protokol ist Teil des Transportlayers. Es ist ein verbindungsorientiertes und sichers Protokol. D.h., dass eine zweiseitige Verbindung besteht durch die sichergestellt wird, dass alle Pakete in der richtigen Reihenfolge ankommen. Hierfür setzt jeder Empfänger einen Buffer fest, den er mit erhaltenden Packeten füllt.
	Um sicherzustellen, dass die Verbindung sowohl auf beiden Seiten richtig eingerichtet bzw. abgebaut wird benötigt man einen Timed 3 Part Handshake.
	Connection esteblishment: - SYN, SYN ACK, ACK
	Connection finish: - FIN, FIN ACK, ACK
TCP	Ob alle Pakete angekommen sind wird durch ein dementspechendem ACK Signal verifiziert. Hierfür hat das jeweilige ACK die Sequenznummer+1 des vorherigen ACK Signals. SYN und FIN entspechen genau einem seq Wert
	Das UDP Protokol ist ein Packetorientiertes einseitiges Verbindungsprotokol. Ob Daten in richtiger Reihenfolge bzw. Anzahl ankommen ist nicht garantiert -> Verlorende Packete werden nicht berücksichtigt.
UDP	Vertorande Packete worden nicht herücksichtigt

Das TCP Protokol sieht vor, dass der Reciever nicht mit Packeten überflutet wird.

Um die Sendegeschwindigkeit des Senders zu Kompensieren setzt der Empfänger einen gewissen Buffer fest, der linear gefüllt wird. Erhält der Empfänger Packages out-of-order so Speichert er diese in den Buffer und reicht die Packete erst an die Application weiter, wenn alle Packagages in der richtigen Reihenfolge vorhanden sind.

Flowcontrol in TCP

Es gibt zwei Methoden um sicher zu gehen, dass dieser Buffer nicht geflutet wird: Der Sender fragt Speicherplatz an oder der Empfänger sendet keine Acknowledgements mehr.