UITWERKING TERMEN OS ZOALS GEWENST OP EXAMEN

Samengesteld door Liam volckerick voor zittijd januari 2019. Deze uitwerking bevat alle termen gevraagd van 2018-2013. Mogelijk niet volledig, daar de jaren vorderen, al is dit vrij volledig om zeker 4/5 termen te kunnen bespreken. Uitwerkingen gemaakt met het idee als oplossing van de examenvraag te zijn.

- Access Matrix:
  Het algemene model van protection kan abstract worden beschouwd als een access matrix. De rijen hiervan zijn de domeinen en de kolommen vertegenwoordigen de objecten. Elk element binnen deze matrix bevat een set aan access rights, ookwel de set aan operaties die een bepaald proces binnen dat domein op dat object kan uitvoeren. Het geeft ons dus een deftig mechanisme om dusdanig strikte controles voor statisch alsook dynamische allocatie te definieren en implementeren. Het veranderen van process op domeinen heet switch operation. Dit switchen kunnen we ook onder controle houden door alsook de domeinen naast de objecten te definieren in de kolommen. De switch operatie zal dus alleen toegelaten worden indien deze in de acces matrix de daartoe benodigde rechten voor heeft.

  Negatives:

  1)Confinement Problem. Stel dat we de entry 'control' toevoegen aan D2, D4 in de access matrix, dan zou een process in D2 het domein van D4 kunnen aanpassen tijdens executie. Copy&Owner rights kunnen deze propagatie van rechten beperken, alleen kan deze niet zorgen voor preventie van uitsluiting van de data. Dus de initiele data in een object kan niet onder garantie behouden blijven binnen dit object.

  Implementatie:
  1) Capability Lists for Domains: Ipv objecten, plaatst het domeinen in afzonderlijke rijen. Deze bevatten lijsten van objecten met hun daartoe behorende rechten. Objecten worden hier herkend als hun ware naam/adres, capability genaamd
  2) Lock Key Mechanism
  3)Global Table
  4)Acces Lists for Objects

- Acyclic Graph Directories:

  Dit is een grafe zonder cycles. Deze structuur laat directories toe subdirectories en files te delen. Ook kunnen dezelfde files en subdirectory in 2 verschillende directories plaats vinden. Enige verandering aan deze structuurs bestanden, zal direct zichtbaar zijn voor alle andere gebruikers binnen diezelfde gedeelde directory/file.

  Pluspunten: De relatieve simpliciteit van de algoritmen om deze grafe te doorzoeken alsook operaties uit te voeren op de objecten in deze structuur. Dit doordat er geen cycles zijn binnen deze structuur (niet altijd).

  Negatieven binnen de implementatie van Links:

  1) verzekeren van geen cycles. Zo niet, dan zouden er meerdere paden ontstaan naar dezelfde subdir/file binnen shared subdir/files. Dit zou onnodige overhead veroorzaken. Dit kunnen we doen door implementatie van General Graph Directory.

  2) Verwijderen van elementen (wanneer de space van allocated shared files dealloceren en daarna re-usen?)
  Solution: Gewoon de file verwijderen vanaf dat 1 iemand deze command uitvoert. Dit resulteert in hanging pointers die leiden naar nergens.
  In een system waardat sharing wordt vertegenwoordigd door sharing links, isdit veel simpeler. We verwijderen gewoon de link. Indien de file zelf wordt verwijdert in deze implementatie, resulteert dit in losgelaten links die foute pointers bevatten, indien deze niet behoorden tot een bepaalde set aan links binnen de file, zullenwe deze moeten gaan zoeken. Deze operatie kan echter zeer kostelijk zijn. Een alternatief hiervoor is het zo laten en bij de eerstvolgende oproeping van deze links zou leiden tot een exception en dus kunnen we deze link verwijderen.
  Andere manier is om de file te preserveren totdat alle links hierrond verwijdert zijn. We kunnen bv een lijst bijhouden van alle links betrekkende tot de te verwijderen file. Deze kan echter zeer groot worden, dus best is om een counter bij te houden van de geassocieerde aantal links.

Implementatie: Link: Dit is een pointer naar de andere file/subdir. De naam vd echte file zit in deze link. Link is resolved wanneer deze opgeroepen wordt tot het loceren van de echte file.

- Address binding (by the compiler, loader, at execution):

  Een programma is meestal een binary executable file op de disk. Bij executie moet dit prgramma in memory worden geplaatst en binnen een process worden geplaatst. Maar vanwaar komt deze executable? De address binnen een source file hebben meestal een symbolische aard en dus zal de compiler deze addressen binden tot realloceerbare addressen. Waarna de loader deze binding van realloceerbare addressen omzet in absolute addresses. Deze binding van instructies/data/memory allocatie en dergerlijke kan in elke stap worden uitgevoerd:

  1) Compile Time: Wanneer we weten waardat het programma in de memory zal plaats nemen, kunnen we absolute code genereren. Stel deze neemt plaats op locatie R, alles kan daarop verder bouwen in de memory, neem nu dat de locatie van R verandert, dan moet er opnieuw gecompileerd worden.

  2) Load Time: Indien in compile time niet geweten is waar men het proces in memory moet plaatsen, zal de compiler relocateable code moete genereren. Hierdoor zal finale binding uitgesteld worden tot load time. Indien het startadres verandert horen we enkel de user code te reloaden om dusdanig deze veranderde waarde te implementeren.

  3) Execution Time: Indien het proces kan worden verplaatst tijdens running(van memory segment tot andere), dan zal binding uitgesteld worden tot run time. Hiervoor is speciale hardware nodig.

- Armoured Virus:

  Een virus zodanig geprogrammeerd dat deze niet gedetecteerd kan worden door anti virussen alsnog door

onderzoekers kan begrepen worden/ontrafeld worden. Deze worden gecompressed om dusdanig detectie te vermijden. De schadelijke malware zit meestal verstopt onder de file attributes of onzichtbare file names.

Alternatieven:

1) Multipartite Virus (verspreid onder meerdere elementen van het system (boot sector, memory, files) Hierdoor moeilijk te detecteren alsnog verwijderen.

2) Polymorphic Virus: Een polymorphic virus verandert zichzelf elke keer dat deze geïnstalleerd wordt om dusdanig detectie te vermijden. De verandering verandert alleen 'the signature' van dit virus. Deze signature wordt gebruikt bij het detecteren van virussen. Meestal is dit een sequentie van bits om een viruscode te herkennen.

3) Encrypted virus: Een encrypted virus bevat encryption key samen met degecodeerd virus, opnieuw om detectie te vermijden. Het virus decrypt zichzelf en vervolgens zal deze uitvoeren.

4) Stealth: Dit virus probeert detectie te vermijden door de systemen die malware/virussen zouden detecteren aan te passen. Bv: het zou de read system call kunnen modificeren waardoor het 'het te lezen file', die werd geinfecteer, wordt gelezen, dan zal de virus de originele code teruggeven ipv de geinfecteerde code.

5) Tunneling: Dit sort virus probeert detectie te vermijden door zichzelf te installeren in de interrupt handler chain. Gelijkaardige virussen installeren zichzelf in de device drivers

– Asymmetric/Symmetric Communication(aka addressing):

Dit neemt plaats binnen het IPC, specifieker, message passing systems -> naming.

Wanneer we een communicatie link hebben tussen 2 entities, die beiden weten wie ze zijn van elkaar en er is maar exact 1 link. zal er binnen een symmetrische communicate, geweten zijn welke 2 processen onderling

tegen elkaar communiceren doorheen deze link. Alleen zo kan er communicatie ontstaan.

Asymmetric: Hier hoort er nog steeds een link te zijn tussen 2 processen, echter hoeft hier geen weet zijn van elkaars identiteiten. De sender moet weten naar wie hij zal struren, de recipient. De recipient daarentegen hoeft niet te weten van wie de boodschap kwam. Zo hebben we deze structuur in asymmetric: send(P, message) en receiver(id, message), met id een naam voor de process waarmee gecommuniceerd is.

Negatieven: De gelimiteerde modulariteit van de overblijvende proces definities. Door het aanpassen van de identifier zal het checken van alle andere proces statements ook noodzakelijk worden. Alle voorgaande declaraties van deze identifier moeten gevonden worden en handmatig aangepast worden.

– Asymmetric/Symmetric Encryption:

Er zijn 2 soorten algoritmen voor de encryptie van data.

1) Symmetric: Dezelfde key wordt gehanteerd in het encrypten/decrypten. Hierdoor moet de secrecy van 'k' (=key) beveiligd worden. De meest bekende versie hiervan is DES(data encryption standard). DES werkt op blokken aan bits tegelijkertijd, dit noemt met Block siphers. Dit is echter zeer vurnerable, omdat dezelfde key voor elke blok wordt gebruikt. Triple DES ten gevolge (2 keer encrypten, 1 keer decrypten.)

Bv: $c = E_{k3}(D_{k2}(E_{k1}(m)))$. Verder volgde AES, deze gebruikte keys van verschillende lengtes(128, 192, 256). Verdere uitwerking op block siphers: Stream siphers, indien de lengte vande communicatie overweldigend zou werken op block siphers, zal stream siphers veel efficiënter zijn. Deze encrypt een hele lijn, en dus grotere messages. Deze vorm werkt veel sneller in encrypten en decrypten van data.

2) Asymmetric: Gebruik van verschillende encryption/decryption keys. Zo wordt deze veel gebruikt bij end to end encryptie. Een ontvanger maakt 2 keys, 1 is de public key, eender wie kan die

aanvangen en onder deze key iets versturen, alleen de ontvanger kan deze decrypten. Het maakt niet uit wie dit meevolgt, aangezien zij deze encrypte messages niet zal kunnen ontcijferen, + keys horen niet meer privaat bekend te zijn. Om bidirectionele encryptie te hebben, zal de sender ook een public key moeten generen. Deze vorm is computationeel intensiever. Voorbeeld: RSA

Beide vormen worden tegenwoordig niet echt nog gebruikt, echter wel af en toe voor kleine data en authenticatie.

- Best Fit:

  Dit is een algoritme gebruikt bij dynamic memory allocation. We gaan alle vrije segmenten van de memory rangschikken van klein naar groot. Hierover itereren we dan totdat we een gevonden memory allocatie gevonden hebben waardat onze memoryblock van ons gegeven proces in past.

  Alternatieven: Worst Fit, First fit

  Negatieven: External Fragmentation bij First fit en Best fit. Dit is omdat we steeds een memory space alloceren aan het te alloceren deel zodat er zo weinig mogelijk vrije ruimte overblijft. Die vrije ruimte is nu juist ongebruikte memory omdat hier geen enkele andere memoryblock van eender welk ander process in past. Deze zijn dus ongebruikte allocaties. Solutions: Compacting (alle vrije ruimte bij elkaar in een blok), segmentatie van memory (memory blocks). Deze laatste kan echter wel leiden tot internal fragmentation.

- Block Device:

  Dit is een device dat toelaat om at random access te verlenen aan onafhankelijk, vrije en gefixeerde grootte aan datablocks. Bv: HDD's, USB, Floppy disc, etc. Voornamelijk worden deze gebruikt voor het opslaan van file systems. Echter kan ook directe access worden verleend aan programma's om file systems te creeeren en repairen

Alternatieven:

1) Character device: direct access to the hardware, het schrijven van een byte zal resulteren in het hebben geschreven van die byte op de hardware. Deze access is sequentieel, terwijl die van block devices random is.

2) Network Devices: Deze worden anders gehandhaafd dan voorgaande devices. Er kan alleen gecommuniceerd worden via een connectie in het kernel's networking subsystem.

- Block Device Interface

  Deze bevat alle nodige zaken omtrent het kunnen accessen van block devices en hiermee interactie te voeren. De devices zijn verwacht commandos zoals read() en write() te herkennen en verstaan. Bij random access devices verwacht het ook een seek() commando. Deze drie commandos vormen the raw I/O. Stel dat een programma zijn eigen buffering doet, dan is de buffering van het file system overbodig. Om dergerlijke conflicten te vermijden geeft raw device access, controle direct tot deze device over aan de applicatie. Dus is de OS buiten spraak hier. Dit heet Direct I/O.

  Ander voorbeeld is memory mapped file access. Ipv raw I/O commandos aan te bieden, geeft deze toelating tot de device storage via een series bits in de main memory. De system call die een file in memory mapped, returned een virtueel adres naar deze file. Werkelijke data transactions worden alleen gepleegd indien er toegang tot deze memory image nodig is.

  Omdat de transacties worden gepleegd door hetzelfde mechanisme als dat gebruikt is voor demand-paged virtual memory access, is memory mapped io zeer efficient

  Deze methode valt te vergelijken met het schrijven/lezen naar memory.

- Blocking

  Wanneer een applicatie een blocking system call oproept, zal deze executie van deze applicatie worden gesuspend. Deze app is dan verplaatst naar de wait

queue. Na dat de system call is vervolledigd, dan zal de applicatie worden verplaatst naar de run queue.

Tegenhanger: non-blocking, bv het ontvangen van mouse en keyboard io terwijl het deze input doet verschijnen op het beeldscherm.

- Boot Disk

  De disk die dat de boot partitie heeft staan, noemen we de boot disk. Deze bevat als het ware de inhoud van de bootstrap die zich bevind op de ROM. Omdat ROM alleen kan gelezen worden, zal bv het updaten hiervan niet zeer handig zijn om te doen. Hierdoor laden de meeste systemen een kleine bootstrap loader in de boot ROM. Deze heeft als enige taak een hele bootstrap programma te laden vanuit de disk. Bij aanpassingen van deze bootstrap is zo simple als het schrijven van de nieuwe code op deze disk. Dit programma is dan opgeslagen in de zogenaamde boot blocks.

  Windows: Master Boot Record

- Capabilities for protection

  Een capability is een object dat wordt gerepresenteerd als zijn physical name of address. Deze worden o.a. gebruikt in capability lists for domains. Capabilities werden geïntroduceerd als veilige pointers. Deze waren nodig om de nodige safety te introduceren op resources. Dit was ten tijde van de opmars van multiprogramming computer systems. Dit was een fundament voor de protection dat zich uiteindelijk op applicatie niveau ging bevinden.

  Capabilities worden onderscheden van andere date door volgende 2 kenmerken:

  1) Een tag op elk object. Dit detoneert of deze een capability was of accessible data. Meestal worden meerdere bits gebruikt om dermate het type object te onderscheiden. Zo weet de hardware met welk type het te maken heeft.

  2) De address space geassocieerd met het programma kan worden opgesplitst in 2 delen. Alleen OS kan

toegang krijgen tot de capability list. De andere lijst bevat alle andere data.

- Certificate Authority:

  Een probleem binnen het verdelen van public keys, is het kunnen bewijzen van wie allemaal een public key bevat. Dit kan opgelost worden m.b.v. digital certificates, deze worden getekend door een betrouwbare third party. Deze party bevat dan alle nodige data om dit te kunnen bewijzen voor ene toegewezen entity. Deze parties noemen we certificate authorities. Deze zijn betrouwbaar doordat ze hun public keys meedragen in hun webbrowsers vooraleer distributie begint. Deze parties kunnen dan ook certificaten maken voor public keys van andere autoriteiten en daardoor een trust factor creëren voor zichzelf.

- Circular Wait Deadlocks:

  Dit is een van de condities om een deadlock te bekomen binnen een systeem van processen die allemaal beroep doen op resources. Zo is er dan een set aan processen P0, P1, P2, ... , Pn, waardat P0 wacht op de loslating van de resource, gehouden door P1, dit analoog voor P1 die wacht op P2 etc, tot Pn-1 die wacht op Pn. Met Pn wachtende op P0.

  Alternatieven:

  1) Geen preemtie: Dit wil zeggen dat het process die een resource vasthoudt, deze vrijwillig pas loslaat nadat deze zijn taak heeft volbracht.

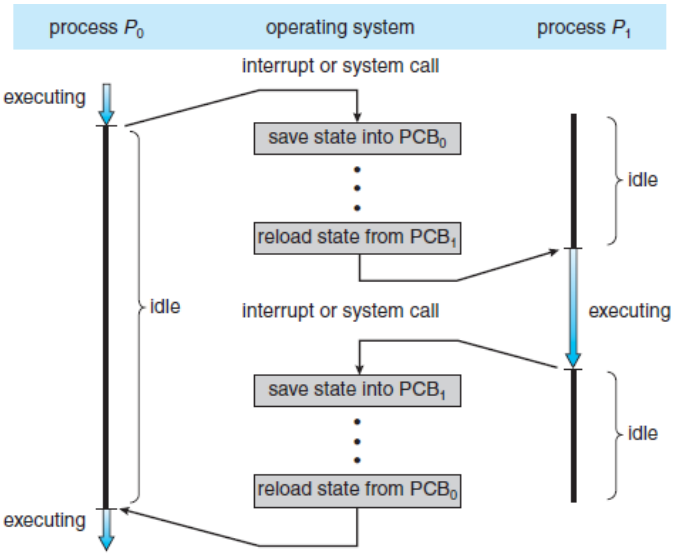
  2) Hold&wait: Een proces moet een bepaalde resource bezetten terwijl het wacht op de vrijlating van een andere resource.

  3) Mutual exclusion: Er moet minstens 1 resource zijn die in een nonsharable mode wordt gehouden. Dwz dat er maar 1 process per keer aan deze resource kan geraken. Indien een ander proces deze resource aanvraagt, zal deze moeten wachten op de vrijlating hiervan.

- C-Look x2:

  Dit is een van de vele disk scheduling algoritmen. Samen met LOOK, zoeken deze twee algoritmen allereerst naar een request alvorens te bewegen in een gegeven richting. C-Look is een realistische implementatie van C-Scan. C-Scan heeft de eigenschap dat ze altijd de hele disk afgaat waarna jobs te furfillen onderweg naar de uiteinden van de disk. C scan is circular, dwz dat ze bij het bereiken van het einde van de disk, terugspringt naar het begin. C-Scan en SCAN zijn goede algoritmen indien er een zware lading aan jobs plaats vind op de disk. C-look bevat alle aspecten van C-scan in de zin dat het altijd de dichtstbijzijnde request furfilled in een bepaalde richting. Bij het bereiken van een punt waardat er in diezelfde richting, geen requests meer plaatsvinden, springt de arm terug naar het begin van de disk. Meer bepaald, het springt naar de eerste request dat zich voordoet vanaf het begin van de disk.

- Context Switch



- Contiguous Memory Allocation

  Het geheugen wordt opgesplitst in 2 secties, 1 voor de user en 1 voor het OS. Contiguous memory alloc. Is de manier waarop processen die in de waiting state zitten, in memory worden gebracht. Zo zal elk proces zijn eigen allocatie hebben en liggend naast deze (contiguous), bevindt zich een ander proces en zijn gealloceerde

geheugenblok. Zoals net gezegd in een manier van memory allocation, het geheugen opdelen in verschillende geheugenblokken. Wanneer er zicheen allocatie vrijgeeft, kan een proces uit de input queue, geladen worden in deze partitie, zodat deze vrije ruimte benuttigt wordt.

Andere manier van partitie: Variable-Partition, het OS zal een tabel bijhouden van vrije ruimtes in het geheugen, alsook de bezette regios. In initialisatie zal deze aanduiden dat de memory leef is.

- Data Parallelism

  Deze term is een beschrijving van het verdelen van subsets aan data over meerdere cores. Waarop dezelfde berekeningen worden gevoerd. Neem bv subset {0}-{N/2} als A en subset {(N/2)+1}-{N} als B, beiden willen de som van iedere subset om dan de totale soms te berekenen van de volledige dataset, door beide subsets op verschillende cores te laten runnen kunnen we dus beide subsets tegelijkertijd berekenen.

  Alternatieven parallelisme:

  1) Task Parallelism: Ipv data over meerdere cores te verdelen, worden er threads verdeeld onder alle cores. (multithreading)

- Deadlock Prevention

  We kunnen deadlocks voorkomen door minstens 1 van zijn voorwaarden te voorkomen, ik bespreek elke voorwaarde afzonderlijk in het kort.

  1) Mutual exclusion: Het gebruik van sharable resources, zoals RO-files. Aangezien sharable files nooit in exclusief gebruik kunnen zijn, zal deze nooit voor een exclusion zorgen.

  2) Hold&Wait: We horen te verzekeren dat geen enkel proces bij het requesten van een resource, nog een resource ter beschikking houdt. Deze zou alle resources horen vrij te laten alvorens een request te issueën. Een andere policy is dat elk proces voor de executie alle resources en allocaties op voorhand request.
     Beiden hebben negatieven: Resource utilisatie zal laag zijn en weinig gebruikt. Starvation is ook

mogelijk. Want belangrijke/populaire resources kunnen altijd al gealloceerd zijn door andere processen waardoor het resquesting proces ongekende tijd zal moeten wachten.

3) No Preemtion: Indien een proces een bepaalde resource bezit, en deze vraagt naar een andere resource dat nietdirect kan gealloceerd worden naar dat vragende proces, dan zullen alle resources van dat proces worden losgelaten.

4) Circular Wait: We bouwen een hierarchie van resources door R_i voor te stellen als een resource, stel dan functie F: N->N met dan F(Ri) = een natuurlijk getal. Dan kunnen we een policy opstellen dat elk proces maar in een ascending order beroep kan doen op bepaalde resources. Stel P1 doet beroep op de disk driver, die een F-waarde heeft van 4, dan mag P1 alleen nog beroep doen op resources die een hogere F-waarde hebben. Indien P1 een bepaald resource meerder maal nodig zal hebben, zal dit binnen 1zelfde oproep moeten gebeuren. Alleen deze policy zal deadlocks niet preventen, het is dan ook aan developers om de applicaties zodanig te schrijven dat deze policies steek kunnen houden.

   Alternatief: DeadLock avoidance: Dit kan door restricties op te werken op hoe dat een proces request mag gebeuren(dead lock prevention), maar ook op de manier waarop resources kunnen worden opgevraagd. Hiervoor zal extra informatie nodig voor zijn. Bv: 2 resources, P1 als proces. 1 resource weet dat P1 pas R1 en R2 zal loslaten na het voltooien van zijn task. P2 daarentegen laat R2 direct los na het oproepen hiervan. Dus is het efficienter om P2 eerst te laten waarna P1 beide recourses mag bezitten.

- Demilitarized Zone

  Een netwerk firewall verdeelt een netwerk over meerdere domeinen. Een implementatie hiervan heeft een domein hiervan als DMZ, die semi betrouwbaar is en heeft het internet als ander onbetrouwbaar domein. Het derde domein is dan een bedrijf 's computersystemen.

Connecties zijn toegelaten tussen:

1) Internet -> DMZ

2) DMZ->Company

3) Optioneel: gecontroleerde communicatie DMZ->Company

DMZ hoort aanvalsbestendig te zijn en veilig.

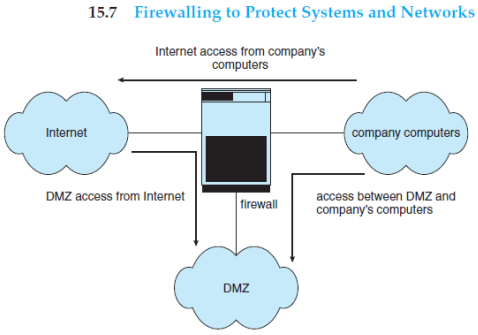


Figure 15.10 Domain separation via firewall.

– Denial of Service

Standaard aanvallen op systemen heeft als doel het afbreken van beveiligingssystemen zoals anti virussen. Aanvallen over een netwerk werken anders dan voorgaande. Aanvallen op systemen en netwerken creeren een situatie waar dat resources van het OS en user bestanden worden misbruikt. Hoe meer 'open' het OS is, hoe meer het zal toelaten en dus hoe gevoeliger het zal zijn voor dergelijke aanvallen. Meestal heeft het OS een detectiesysteem om afzenders hun ID changes te tracken, echter wanneer hackers bezitten over meerdere systemen zal dit veel moeilijker zijn om die na te gaan. Vandaar dat vele programmas persoonlijke vragen stellen bij authenticiteit, daar deze moeilijker zijn om te achterhalen als hacker. DOS aanvallen zijn niet zozeer gericht op het stelen van data, maar eerder gericht naar het uitsluiten van correct gebruik van het systeem/faciliteit. Deze aanvallen zijn makkelijker dan aanvallen om systemen te doordringen. Er zijn 2 types van DOS.

1) Aanvallen die gericht zijn op het bezetten van zoveel mogelijk faciliteiten binnen het systeem. Bv klik op site resulteert in een applet die 96% van de cpu gebruikt.

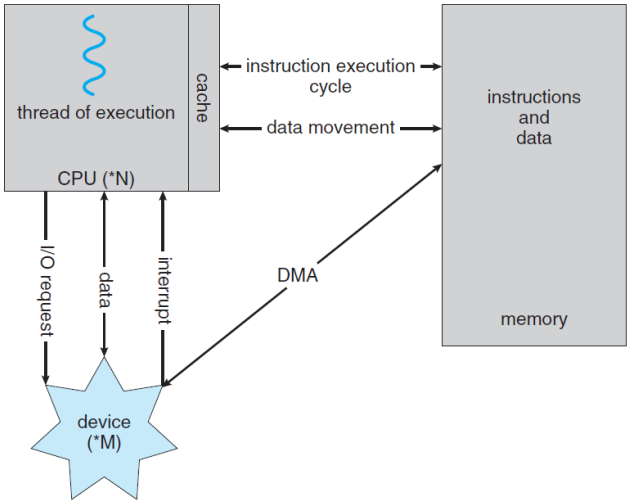
2) Tweede type is het platleggen van het netwerk van het doeleind. Hierbij wordt TCP/IP geëxploiteerd.

Over het algemeen kunnen DOS-aanvallen niet worden vermeden. Een variant hierop is DDOS, deze houdt in dat vanuit meerdere sites naar eenzelfde doeleind wordt aangevallen. Deze attacks hebben meestal betrekking tot blackmailen en het eisen van geld.

Alternatieven: Worms, Port scanning

– Device Driver

Voor elke device controller is er een device driver. Deze twee liggen in nauw verband, de device driver geeft het hele OS een uniform interface tot dat betreffende device. Bv bij het opstarten van een I/O request, zal de device driver de juiste registers laden in de device controller. Na executie, zal de device controller via een interrupt de device driver laten weten dat zijn taak erop zit. Waarna de device driver terug controle geeft aan het OS met mogelijk de gewenste data, die werd opgevraagd door de initiele I/O. Deze manier van interactie met devices is goed voor kleine hoeveelheden, echter bij bulk data kan dit enorme overhead veroorzaken. Oplossing? DMA.



– Disk Partition

Disk partition is ook wel bekend als disk slicing. Deze uitvoering bevindt zich in het formatteren van een disk. De naam zelf zegt al dat we de disk als het ware gaan partitioneren in enkele groepen van cilinders. De OS zal deze beschouwen als verschillende disks. De disk houdt

dan een partition table bij. Deze bevat alle informatie omtrent de locatie en grootte van elke partitie. Verdere stappen bevat het logisch formatteren, ook wel het creëren van de file system. Om efficientie te behouden stapelt de file system blokken bij elkaar, genaamd clusters. Disk I/O is zo gedaan via blocks, maar file system I/O gebeurt via clusters. Dit is zo zodat er meer gebruik wordt gemaakt van sequential access ipv random.

Andere implementatie: Sommige programmas laten disk partitie toe in de vorm van een grote sequentiele array van logische blokken. Dit noemt men ook Raw Disk. Raw-IO gebeurt hierop. Raw-IO bypassed: buffer cache, file locking, space allocatie, file names, directories etc.

– Dynamic Linking

We nemen aan dat een programma is geladen in het geheugen. Echter hebben vele functies nood aan libraries, die indien niet geimplementeerd zijn binnen de binaire file, dus ook gealloceerd moeten worden. Mocht deze toch toegevoegd zijn in de binary file, dan noemen we dit statically linked. Het probleem hierbij is dat elk programma een kopie bevat van overlappende libraries die door meerdere programma's gebruikt worden. Het is efficienter indien we deze eenmalig allemaal zouden moeten laden in memory. Dit doet dynamic linking. Elke library routine bevat een 'stub'. Deze bevat een stuk code dat aangeeft hoe dat men de library moet alloceren indien aanwezig, of indien niet aanwezig, hoe deze moet geladen worden. Na uitvoering van de 'stub' zal deze zichzelf vervangen door het adres naar de routine en executie zal starten.

Pluspunten: Dit is zeer handig voor wanneer herhaaldelijke code wordt uitgevoerd die beroep doen op deze library. Alle programma's die aldus dezelfde library hanteren zullen allemaal dezelfde kopie hiervan gebruiken. Dit is ook handig voor bug fixing. 1patch->alle programma's die de library hanteren zijn ook gepatched.

OS moet wel checken ofdat een gereferenceerde library reeds al gealloceerd is binnen een ander proces. Zo ja dan is het de taak van de OS om deze te verwijzen naar de requesting process.

– Earliest Deadline First Scheduling

  Algoritme voor CPU-scheduling. De naam verklapt de werking van dit algoritme. Hoe dichter bij de deadline van een task, hoe groter de prioriteit van deze task om te executeren. Analoge redenering voor deadlines die nog verder weg liggen (zullen dus later pas geëxecuteerd worden). Elke task/process moet wanneer het in 'runnable' state terecht komt aangeven wanneer zijn deadline omgaat. Hierdoor zal het algoritme dynamisch herschikken elke keer dat er een nieuwe task bijkomt.

  Alternatieven:

  1) POSIX real time scheduling: geeft extensies voor real time scheduling.

     - Sched FIFO: threads volgends fifo. Geen time slicing, dus thread met hoogste prioriteit vooraan in de Q, deze zal de cpu ter beschikking krijgen tot na voltooiing.

     - Sched RR: Gebruik van Round Robin policy. Gelijkaardig aan sched fifo, echter wordt er wel gebruik gemaakt van time slicing onder threads met gelijke prioriteit.

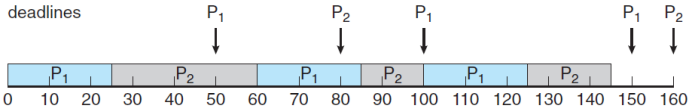
  2) Proportional share scheduling: De scheduler opereert door middel van X-aantal shares te delen onder alle applicaties. Elke applicatie kan N-shares ontvangen aan tijd, en dus N/X-tijd krijgen van de cpu. Deze scheduler werkt hand in hand met een admissie controle policy. Zodat elk de toegepaste aantal shares ontvangt naar proportie toe. Er is dan nog een andere policy die checkt opdat de totale hoeveelheid shares voldoet voor alle processen die momenteel runnen. Indien er een applicatie bijkomt die K-aantal shares vraagt, waardoor X overschreden zal worden, zal Pk geweigerd worden, uitstel ten gevolge.

  3) Rate monotonic scheduling: Een scheduler die gebruik maakt van priority based scheduling (preemption indien hogere prioriteit toetreedt). Elke periodieke task krijgt een prioriteit gebaseerd op de inverse van zijn periode.

     Dwz: Korte periode – Hoge prioriteit, Lange periode – Kleine prioriteit. Het idee is dat de cpu zick vaker

zou toekennen aan vaak retournerende taken die van korte duur zijn. Daarbij gaat die algoritme ervanuit dat de processing tijd hetzelfde is voor elke cpu cycle. Dat is: P1 & CPU-Time -> CPU-Time is cte. voor elke keer dat P1 retourneert.

Afbeelding EDF:



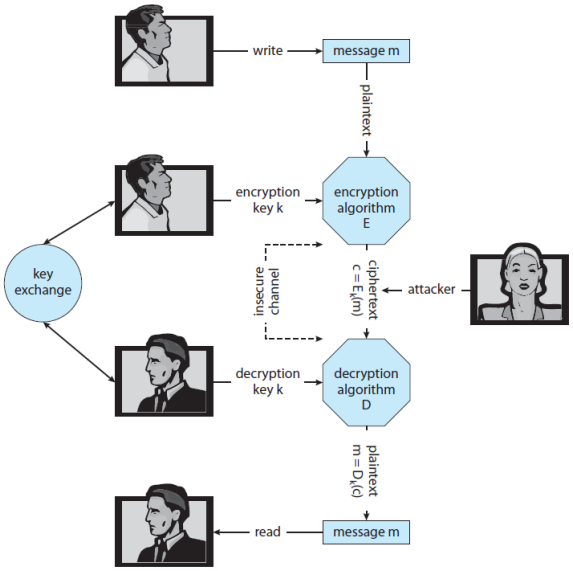
– Encryption

Lost een hoop communicatieproblemen op. Wordt gebruikt op berichten op een netwerk te sturen alsook het beveiligen van database data, folders en zijn files, volledige file systems op disks. Dit concept zorgt ervoor dat alleen geauthoriseerde entiteiten de encrypted data/message kan decrypten door het hebben van de juiste key.

Toepassingen:

1) Symmetric Encryption

2) Asymmetric Encryption



– Enhanced Second Chance Algorithm

Het is een verbetering van second chance algoritme, dit doordat het in rekening brengt van een tupel, namelijk de reference bit & modify bit.

Er zijn door samenstelling van deze tupel 4 mogelijke clauses:

1) 0,0: geen van beiden, dus noch recent gebruikt als modificeerd. Beste page om te vervangen
2) 0,1: Laatst nog gemodificeerd, niet zo goed want de page moet eerst uitgeschreven worden alvorens replacement.
3) 1,0: Recent nog gebruikt, echter nog clean. Zal waarschijnlijk nog gebruikt worden.
4) 1,1: Recent nog gebruikt en modified, dus zal eerst nog moeten uitgeschreven worden alvorens deze kan vervangen worden. Zal in toekomst waarschijnlijk alsnog gebruikt worden

- External fragmentation

  Bij dynamic memory allocation zal er ongeacht het gekozen algoritme bij First fit en Best fit ongebruikt geheugen optreden. Dit is omdat we steeds een memory space alloceren aan het te alloceren deel zodat er zo weinig mogelijk vrije ruimte overblijft. Die vrije ruimte is nu juist ongebruikte memory omdat hier geen enkele andere memoryblock van eender welk ander process in past. Deze zijn dus ongebruikte allocaties. Solutions: Compacting (alle vrije ruimte bij elkaar in een blok), segmentatie van memory (memory blocks). Deze laatste kan echter wel leiden tot internal fragmentation.

- FAT (File Allocation Table)/Linked Allocation

  Dit is een belangrijke variatie op gelinkte allocatie. Telkens wordt er aan het begin van de disk een klein volume vrijgehouden voor de tabel. De tabel bevat een entry voor elke disk block en zo geïndexeerd. Deze tabel lijkt veel op een gelinkte lijst. Elke entry in de tabel bevat een pointer naar het volgende block. Indien er een lege blok onstaat, zal deze table entry vervangen worden naar een 0. Het zoeken naar het eerste vrije block zal dus een sequential search zijn binnen deze tabel naar de eerste 0 waarde als entry. The end of file, zal een speciale waarde hebben en deze toekennen aan de laatste block van deze file.

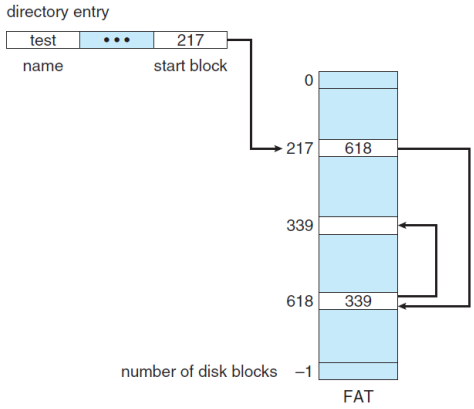
Negatieven: Deze tabel kan resulteren in vele disk head seeks.

Optimalisatie: cachen -> geen seek() van de disk head meer nodig. Maar dit is zeer kostelijk.

Positief: Random Acces time is verbetert! Gewoon aflezen van de block waarde in de tabel.

Alternatieven:

1) Indexed allocation.



– FCFS

Dit is nog een cpu-scheduling algoritme. Het idee hierachter is dat de eerste taak/proces dat beroep doet op de cpu, deze dan ook ter beschikking zal krijgen. Het is ook non preemptive. Dit algoritme makkelijk te creeren door gebruik te maken van een FIFO queue. Wanneer de CPU vrij is, zal deze zich laten toekennen aan de eerste task in de queue.

Positief: Makkelijk te begrijpen en implementeren.

Negatief:

1) Gemiddelde waiting time is nogal vrij lang. Neem bv 3 processen die samen binnen het systeem komen. De getallen na de procesnaam bevat de burst time voor elk proces: P1 = 24, P2= 4, P3 = 6. Stel dat ze P1P3P2 in de queue staan. Gemiddelde waiting time: 0+24+28/3 voor P2. Dit bedraagt 18 ms. Stel nu dat we shortest job first doen. Dan ziet de queue eruit als volgt: P2P3P1.

Gemiddelde waiting time voor de task is dan: (0+4+10)/3 = 4,66... Dit is al heelwat minder dan 18ms.

2) Convoy effect: Queue met snelle tasks moet wachten opdat een grotere task is voltooid waarna bijvoorbeeld na executie van alle I/O in de queue, de cpu uiteindelijk even werkloos zal zijn. Dit omdat een grotere task eerst werd uitgevoerd, gevolgd door short tasks. FCFS is niet preemptive.

Alternatieven: SJF, Priority Scheduling, Round Robin Scheduling, MultiLevel (feedback) Queue scheduling.

- File Handle

  Dit is de term voor in windows, in Linux ook wel bekend als file descriptor. De file handle is een naam/bit dat wordt gegeven aan een file in de file system table. Typisch wordt dit gebruikt bij het openen van de gevraagde file. Deze file handle wordt doorheen de hele uitvoering gebruikt. Wanneer de file wordt gesluiten zal ook deze file handle verwijderen uit de per-process table alsook wordt de counter binnen de system-wide op table entry. Binnen het geheugen is er een dedicated memory block voor alle file handles in op te slaan die currently gebruikt worden. De grootte van deze memory block determineert ook hoeveel file handlers er kunnen zijn at one given time.

  De meeste systemen houden alle informatie over een open file bij in memory, rather than their data blocks.

- File Type

  Bij de creatie van een file system/OS, horen we rekening te houden met het al dan niet implementeren van bepaalde file types. Indien het OS deze herkent, zal deze weten hoe deze file te herkennen en hanteren. Veel gebruikte manier voor file types te implementeren: naam van de file gevolgd door een aanhangsel, de 'file type'. Bv '.exe', '.docx'. Zo zijn er enkele file types die werkelijk kunnen worden uitgevoerd, zoals .exe, .com en .sh (shell script). Zo kunnen ook applicaties een bepaalde file type verwachten in voordracht vooraleer deze juist kan werken. Zie java files en een java compiler.

In unix wordt het file type herkent door een magic number aan het begin van sommige files.

- Fork()

  Dit is de terminologie voor UNIX based OS. Zijn windows equivalent is bekend als 'createProcess()'. Deze wordt opgeroepen wanneer een bepaald proces een nieuw proces gaat creëren en dus partent process wordt. De parent zal dan tegelijkertijd met de nieuwe process runnen. Of de parent wacht totdat sommige/alle van zijn kinderen voltooid zijn alvorens verder te zetten.

  Er zijn zo ook 2 mogelijkheden voor de adres allocaties: 1) De child is een duplicaat van de parent (zelfde data en programma) of 2) de child heeft een eigen programma geladen.

  Meestal wordt er na een fork() system call ook wel een exec() system call opgeroepen door een van de twee processen. Deze zal de memory van het proces reloaden met een nieuw binair programma (dit verwijdert de memory image van het proces dat de exec() call deed).

  Binnen dit systeem kunnen parents en children communiceren en alsnog simultaan verderzetten met hun taak. De parent kan zichzelf ook in wait() zetten indien deze moet wachten op de children.

- Free Space Management

  Het is de grote term voor het bijhouden van alle vrije disk space binnen het OS. Hiervoor hanteert het OS een free-space lijst. Deze bevat alle verwijzingen van vrije disk blocks binnen een disk.

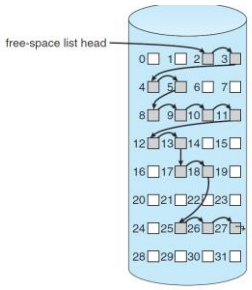
  Er zijn verschillende vormen van deze free-space list:

  1) Bit vector: aka bit map, een lange vector van bits waarin elke bit een block voorstelt. Indien de waarde == 1, dan is de block vrij, indien == 0, dan is deze block reeds gevuld.

     Positief: Makkelijke implementatie en simpliciteit voor het zoeken van free space van 1-n sequantial blocks.

  2) Linked list: Linken van alle vrije blocks op de disk. Niet efficient om altijd I/O te moeten hebben om alle vrije blocks te achterhalen, aangezien elke vrije

block een point zal bevatten naar eerstvolgende vrije block. Echter zal deze operatie nauwelijks gebeuren aangezien we meestal enkel de eerste vrije block nodig zullen hebben indien hier nood aan is. Er kan gebruik worden gemaakt van FAT



3) Grouping: modificatie van de free list, deze houdt de addressen bij van n blokken, waarvan n-1 werkelijk vrij zijn en in block n de addressen zitten van volgende n blokken die vrij zijn.

4) Counting: Ipv alle adressen bij te houden van n-1 blocks kunnen we ook adres bijhouden van de eerste vrij block met dan ook het aantal vrije opvolgende (sequentiele) blokken gevolgd op deze block. Elke entry bestaat dus uit (address, count). Elke entry is wel groter maar de grootte van de tabel zal kleiner moeten zijn indien de counter gemiddeld > 1.

5) Space Maps: Dit is een log van alle space activiteiten. ZFS creeert metaslabs van de space in manageable groottes van disk size. Elke metaslab heeft een space map met daarin ook de counter. Het gebruikt log structured file system technieken om naar de informatie bij te houden.

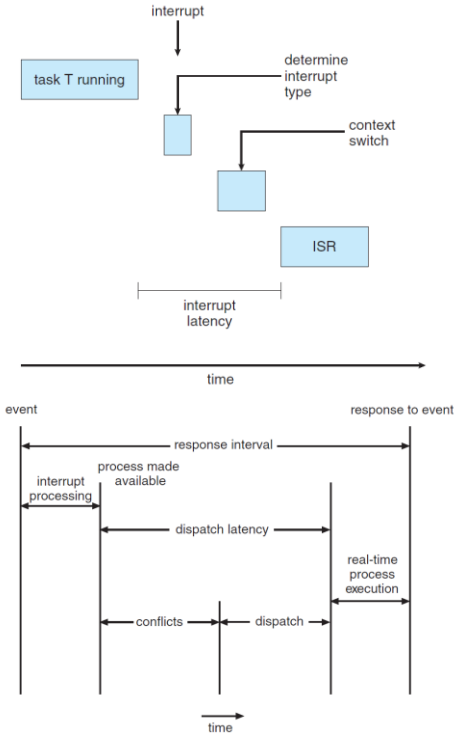
- Hard/Soft Real Time System

  Binnen Cpu scheduling zijn er 2 real time systems: Hard&soft.

  Soft: Geen garantie wanneer dat een critical section zal worden gescheduled. Garantie dat processen met critical section prioriteit hebben over die zonder.

  Hard: TOV soft, strictere noodzaken, rekening houden met deadlines, na verloop van de deadline is er geen service meer als het ware.

Problemen: Minimalisatie van de latency, aangezien real time systems zo snel mogelijk op huidige services kunnen antwoorden en dus sneller moeten processen. Er zijn 2 soorten latency: 1) Interrupt latentcy: tijd van interrupt tot executie van de interrupt service routine 2) Dispatch latency: tijd tussen de interrupt processing en de executie van het proces.

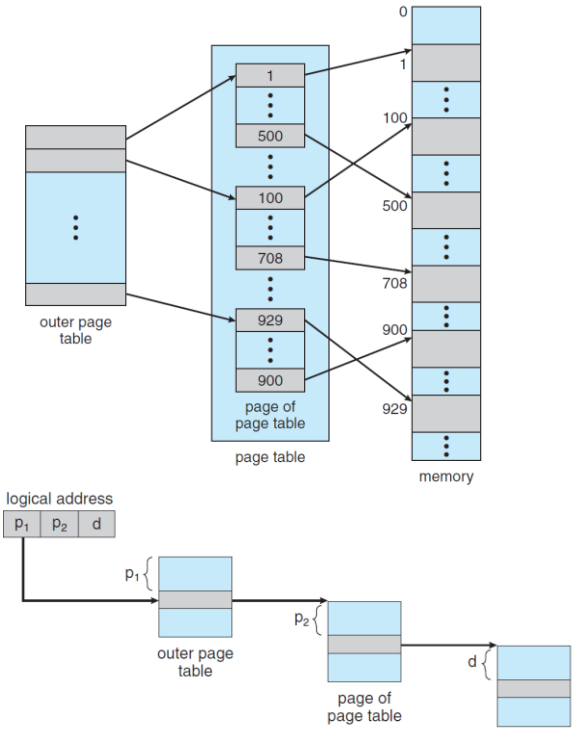


Factoren bij het bepalen van de latency: De tijdspan waarin interrupts worden genegeerd terwijl de kernel interne updates uitvoert. De tijdspan voor het stoppen van een proces om verder te gaan met een ander proces (dispatch latency) -> preemptive kernels als solution.

Dan zijn er algoritmen die hard real time volgend mbt preemption: POSIX real time scheduling, proportional share scheduling, EDFS, rate monotonic scheduling -> allemaal vormen van priority based scheduling

– Hierarchical/Hashed/Inverted Paging

Doordat vele systemen een grote hoeveelheid aan logische adressen toelaat -> page table enorm groot. We willen hier structuur binnen brengen en dit is mogelijk via het creeren van multiple level paging, zie tekening (bv van 2-level paging) Deze manier wordt ookwel forward mapped page table genoemd: van outerpage inwaards te werk gaan.



Alternatieven:

1) Hashed page table: In deze implementatie voor het kunnen handelen van adres spaces groter dan 32 bits, is de waarde van de hash value de virtuele page nummer. Een variatie voor 64 bits gebruikt clustered page tables, deze zijn handig voor spaarse adressen spaces, denk aan noncontiguous en scattered (werkt gelijkaardig alleen is elke entry in de tabel een verwijzing naar meerdere pages) Elke entry in de tabel bevat een gelinkte lijst die elementen met dezelfde hash-waarde bevatten. Elke entry in deze lijst bestaat uit {virtual page #, value van gemappede page frame, pointer naar volgende element}.

   Algoritme gaat als volgt te werk:

   Hashwaarde wordt aangemaakt van virtuele adres, virtuele adres wordt vergeleken met eerste entry van de linked list op de bepaalde hashentry, indien match wordt gekeken naar field2 -> bepaald zo physische adres, indien geen match -> verder lineair gezocht naar overeenkomende virtuele page #.

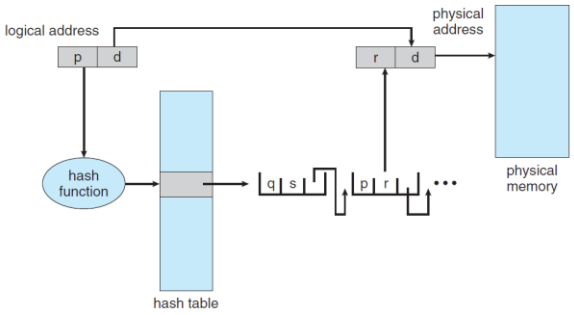


Figure 8.19 Hashed page table.

2) Inverted Page table: elk proces heeft een toegewezen page tabel, die voor elke page die het proces gebruikt als een entry beschouwt. Processen wijzen pages toe volgens virtuele adressen -> OS gebruikt deze om de physische adressen te bepalen.

   Negatieven: De page tabel kan enorm veel entries bevatten -> informatie over hoe de memory wordt gebruikt is dan de voornaamste gebruiker van de memory

   Oplosingen: Inverted paging.

   Elke entry is een virtueel adres naar de echte verwijzing in het geheugen naar de page. Heeft een beetje gelijkenissen aan dynamic binding mbt libraries. Elke entry binnen de page heeft noodzaak aan een address-space identifier (distinguishen van verschillende pages op zelfde virtuele adres maar worden gebruikt door individuele taken).

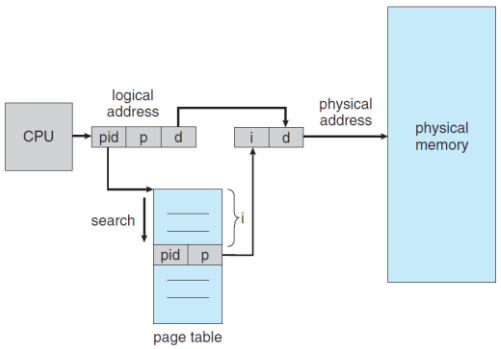


Figure 8.20 Inverted page table.

- Hold & wait

  Een proces moet een bepaalde resource bezetten terwijl het wacht op de vrijlating van een andere resource. Een van de voorwaarden voor een deadlock.

  Alternatieven:

  1) Geen preemtie: Dit wil zeggen dat het process die een resource vasthoudt, deze vrijwillig pas loslaat nadat deze zijn taak heeft volbracht.

  2) Mutual exclusion: Er moet minstens 1 resource zijn die in een nonsharable mode wordt gehouden. Dwz dat er maar 1 process per keer aan deze resource kan geraken. Indien een ander proces deze resource aanvraagt, zal deze moeten wachten op de vrijlating hiervan.

  3) Circular wait: Dit is een van de condities om een deadlock te bekomen binnen een systeem van processen die allemaal beroep doen op resources. Zo is er dan een set aan processen P0, P1, P2, ... , Pn, waardat P0 wacht op de loslating van de resource, gehouden door P1, dit analoog voor P1 die wacht op P2 etc, tot Pn-1 die wacht op Pn. Met Pn wachtende op P0.

  Oplossing: We horen te verzekeren dat geen enkel proces bij het requesten van een resource, nog een resource ter beschikking houdt. Deze zou alle resources horen vrij te laten alvorens een request te issueën. Een andere policy is dat elk proces voor de executie alle resources en allocaties op voorhand request.
  Beiden hebben negatieven: Resource utilisatie zal laag zijn en weinig gebruikt. Starvation is ook mogelijk. Want belangrijke/populaire resources kunnen altijd al gealloceerd zijn door andere processen waardoor het resquesting proces ongekende tijd zal moeten wachten.

- Indexed allocation of files

  Probeert een oplossing te bieden voor the lack of efficient direct access dat FAT met zich meedraagt. Oplossing? Alle index pointers bijhouden in 1 block, de index block.

De i^{de} entry in de index block komt overeen met de i^{de} block in de file. Dus voor het lezen van de i^{de} block wordt er gekeken naar waar de pointer in de i^{de} entry van de index block naar wijst.

Index allocation supporteert echter wel direct access, zonder external fragmentation. Het lijdt wel onder 'wasted space'. De pointer overhead van deze index block is meestal groter dan die van gelinkte allocatie (FAT). Bij FAT, 1 pointer per block. In indexed allocation: een hele index block moet gealloceerd zijn = 2 pointers, zelfs al zijn er maar enkele pointer non-null.

Probleem! Grootte index block?

1) Linked Scheme: Index block is een gewone disk block. Kan dus geschreven en gelezen worden door zichzelf -> voor grote files kunnen we meerdere index blocks linken.

2) Multilevel Index: variant van de linked representatie. First level index block point naar een set van second level index blocks. Die laatste verwijzen dan naar de file blocks. Deze multilevels kunnen doorgaan tot waar de disk space zich beperkt.

3) Combined scheme: Deze wordt voornamelijk gebruikt in UNIX, zie tekening

Negatieven: Indexed allocation lijdt onder zelfde problemen als linked allocation. De index blocks kunnen gecached worden echter kunnen de data blocks verspreid zitten over de hele grootte.

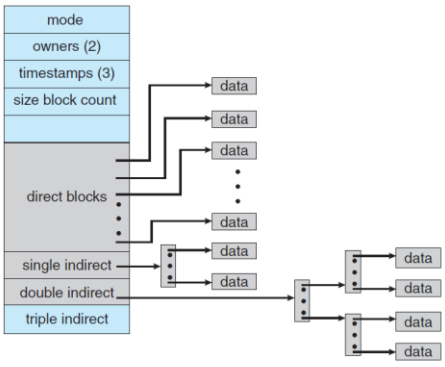


Figure 12.9 The UNIX inode.

- Interrupt:

  Dit is een signaal naar de CPU komende van de hardware/software dat de CPU vertelt dat er een dringende zaak/task te wachten staat. Dit signaal wordt mede mogelijk gemaakt door de device controllers. Hierdoor zal de CPU zijn huidige task stopzetten om dusdanig deze task met hoge prioriteit uit te voeren. De controle wordt na het alarmeren van de CPU overgegeven aan de interrupt handler/interrupt Service Routine. De interrupt handler zal vervolgens de interrupt processen en vervolgens het onderbroken programma voortzetten.

  Hier wordt er dus gebruik gemaakt van een context switch (opslaan en reloaden van een staat van de cpu zodat deze na executie van een interrupt probleemloos kan verder worden gezet).

  2 soorten interrupts:

  1) Software interrups:

     Dit is een interrupt dat onstaat binnen een proces door een uitvoerende instructie. Bv fork() & kill().

  2) Hardwarde interrupts:

     Dit onstaat uit IO op de hardware devices. Bv key presses, mouse clicks, etc.

     NonMaskable Interrupts: Dit wordt veroorzaakt door interrupts met zeer hoge prioriteit zoals hardware errors (interrupt priority levels) in de memory bijvoorbeeld. Dit is vrij logisch aangezien dergerlijke fouten direct moeten worden recht getrokken. Dit moet gebeuren voordat er een andere task met zekerheid op correcte uitvoer kan worden uitgevoerd.

     Maskable Interrups: dit is een hardware interrupt dat mag worden uitgesteld of zelfs worden rejected door het plaatsen van een bit in de bitmask.

     De Interrupt mechanisme neemt een adres als input die dan zal, leiden tot een bepaalde IHR (i. handling routine). Dit adres wordt tegenwoordig meestal geimplementeerd als de offset binnen de interrupt vector. Deze laatste bevat adressen naar alle interrupt handlers. Interrupt chaining zorgt

voor een correcte werking van de interrupt vector (elke entry heeft pointer naar volgende lijst van interrupt handlers) Binnen deze lijst wordt er dan gezocht naar de noodzakelijke handler voor de gevraagde interrupt.

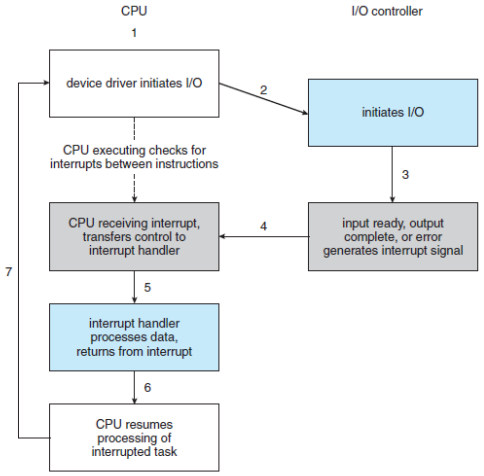
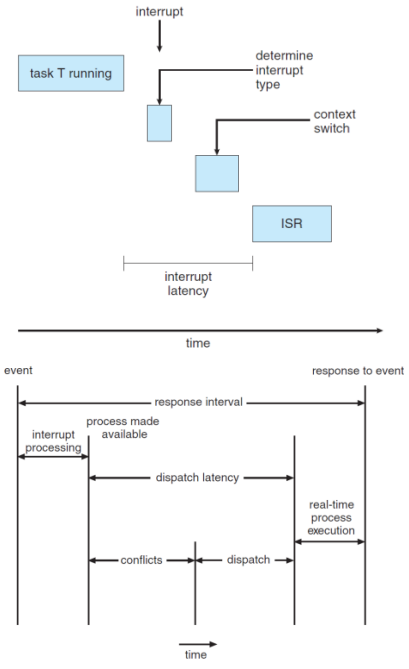


Figure 13.3 Interrupt-driven I/O cycle.

– Interrupt Latency

Tijd van interrupt tot executie van de interrupt service routine

Probleem bij cpu-scheduling in real time: Minimalisatie van de latency, aangezien real time systems zo snel mogelijk op huidige services kunnen antwoorden en dus sneller moeten processen.



Alternatief: Dispatch latency tijd tussen de interrupt processing en de executie van het proces.

- Inverted Page Table

  Inverted Page table: elk proces heeft een toegewezen page tabel, die voor elke page die het proces gebruikt als een entry beschouwt. Processen wijzen pages toe volgens virtuele adressen -> OS gebruikt deze om de physische adressen te bepalen.

  Negatieven: De page tabel kan enorm veel entries bevatten -> informatie over hoe de memory wordt gebruikt is dan de voornaamste gebruiker van de memory

  Oplosingen: Inverted paging.

  Elke entry is een virtueel adres naar de echte verwijzing in het geheugen naar de page. Heeft een beetje gelijkenissen aan dynamic binding mbt libraries. Elke entry binnen de page heeft noodzaak aan een address-space identifier (distinguishen van verschillende pages op zelfde virtuele adres maar worden gebruikt door individuele taken).

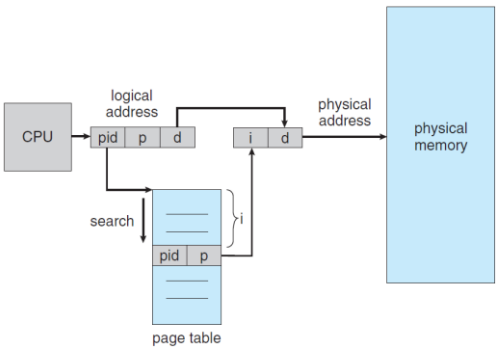


Figure 8.20 Inverted page table.

Alternatieven: Hash paging, Hierarchical paging

- Kernel Thread

  Threads die direct door de OS worden behandeld. Zo bestaat er een relatie tussen de werking van user threads en kernel threads. User threads zijn zichtbaar voor de programmeur maar onzichtbaar voor de kernel. User level threads zijn sneller te manipuleren dan de kernel threads aangezien er geen interventie nodig is van de OS.

Er bestaan meerdere modellen om deze werking te implementeren:

1) Many to one Model: Dit model mapped meerdere user threads naar 1 kernel thread. Thread management wordt gedaan door de thread library in user space -> efficient.

   Negatief: kan blokkeren indien een thread een blocking system call doet, hierdoor zal het hele proces stilstaan.

2) One to one Model: 1 user thread correspondeert met 1 kernel thread. Biedt simultaneïteit van werking aan, hier blokkeert het hele proces niet bij een blocking system call van een thread.

3) Many to many Model: Meerdere user threads worden gemapped naar evenveel of minder kernel threads. Het idee van simultaneïteit bij many to one is niet echt aan te pas komend. De simultaneïteit verbeterd naar the one to one toe, maar de dev. moet nog steeds oppassen dat deze niet te veel threads aanmaakt. Oplossing -> many to many.

– Logic Bomb (malware)

  Dit is kwaadaardige stukjes code die zich kunnen bevinden in de source van downloadbare programma's. Het is een vorm van backdoor. Deze kunnen worden geactiveerd wanneer de daartoe betreffende voorwaarden zijn voldaan. Hierdoor zal er een logic bomb ontstaan, ook wel bekend als het ontstaan van een security hole. Aangezien de code niet door safety programmas is gedetecteerd.

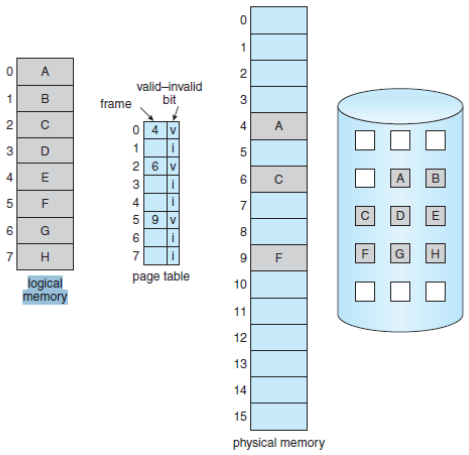
  Alternatieven;

  Trap Door, Stack&buffer overflow, Trojan horse.

– Logical Memory Space

  Binnen het concept van virtual memory worden de logische geheugenblokken gesepareerd van de fysische geheugencellen. De logische cellen zijn als het ware de representatie van het geheugen naar de programmeur toe. Dit idee laat toe dat het lijkt alsof het programma

in een sequentiële opvolging is geladen binnen het geheugen.



– Logical Record of a file

  Deze files records bevatten geen werkelijke data, maar bevatten een korte descriptie van alle records die zich bevinden in of meerdere fysiche records. Een logische file is dus een representatie van een of meerdere fysische files.

  Dit wordt onder andere gebruikt bij direct access binnen file systems. Alsook brengt dit het gemak met zich mee dat er geschreven en ook gelezen kan worden in eender welke manier dat fysische records niet zouden toelaten. Uiteindelijk zijn deze logische records referenties naar werkelijke records en wordt er door het os alles naar correcte behoeften wordt doorgetrokken naar de memory.

  Alternatief: physical record

– Look Algorithm

  Look algoritme werkt in zeker zin gelijkaardig aan SCAN, alleen verschilt het in de zin dat het actief zoekt naar taken om te doen in de richting waardat the head naartoe is aan het lezen. Indien er geen tasks meer te doen zijn zal deze veranderen van richting en dus alle taken dat zich in tegengestelde richting, t.o.v. initiële toestaan, zullen nu gehanteerd worden.

  Negatieven: Het mogelijk ontstaan van track clusters aan de uiteinden van de disk.

Variant: C-Look: Dit lost het cluster probleem op in de zin dat het hoofd alleen maar in 1 enkele richting leest. Wanneer het het einde van de disk bereikt zal deze terugspringen naar het begin van de disk.

- LRU Page Replacement

  Het grootste verschil tussen FIFO en OPT algos is dat FIFO kijkt naar de tijd van entry, OPT kijkt naar de eerstvolgende tijd van laatst gebruik. Indien onze approcimatie voor de toekomst gebaseerd wordt op de recent verleden gebeurtenissen, dan zal de page die langst niet meer gebruikt is geweest worden vervangen -> Least Recently Used. We kunnen dit bezien als het optimale algoritme dat terugkijkt in de tijd. LRU policy wordt gezien als een zeer goeie policy, echter zit het probleem in de implementatie van deze policy.

  Twee implementaties:

  1) Counter: In de page Table houden we een time-of-use veld vrij, alsook in de cpu een logische counter. De counter += 1 voor elke referentie. Telkens wanneer ene referentie is gemaakt naar een page, zal de counter waarde overgedragen worden naar de time-of-use entry. Zo hebben we altijd 'de tijd' van de laatste referentie naar deze page. Overflow mogelijk van de counter (bevindt zich in een register, daarom).

  2) Stack: Een stack bijhouden met page numbers. Telkens wanneer een page is gerefereerd, wordt deze verwijdert en op de top van de stapel geplaatst. Hierdoor is de MRU altijd on top of the stack. Best implementeren met doubly linked list.

  Echter zeer intensief en weinig support om deze implementaties tot realiteit te wekken in de huidige day and age. Meestal ondersteunt de hardware wel een 'reference bit'. -> LRU Approx. Page Replacement. Deze wordt steeds geplaatst wanneer deze onlangs nog eens is gerefereerd. 1 onlangs nog gebruikt, 0 niet recent gebruikt. Dit levert een approx. tot de basis van LRU replacement algoritme.

- Mailboxing of IPC

  Dit is een andere benaming voor message passing, een methode waarop processen communiceren met elkaar.

Binnen message passing zijn er 2 commandos die zeker moeten werken send en receive van een parameter 'message'. De message kan oftewel van fixed of variabele size zijn. Om te kunnen communiceren moet er een link onstaan tussen 2 processen. De logische implementatie hiervan kan op meerdere manieren:

Direct/indirecte communicatie

Syncroon/assyncroon communicatie

Automatische/expliciete buffer

Alternatief: Shared memory communication

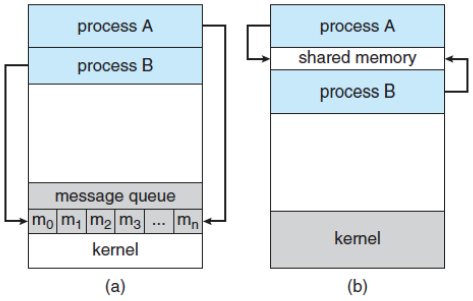


Figure 3.12 Communications models. (a) Message passing. (b) Shared memory.

- Maskable Interrupts

  Dit is een hardware interrupt dat mag worden uitgesteld of zelfs worden rejected door het plaatsen van een bit in de bitmask.

  De Interrupt mechanisme neemt een adres als input die dan zal, leiden tot een bepaalde IHR (i. handling routine). Dit adres wordt tegenwoordig meestal geimplementeerd als de offset binnen de interrupt vector. Deze laatste bevat adressen naar alle interrupt handlers. Interrupt chaining zorgt voor een correcte werking van de interrupt vector (elke entry heeft pointer naar volgende lijst van interrupt handlers) Binnen deze lijst wordt er dan gezocht naar de noodzakelijke handler voor de gevraagde interrupt.

- Master File Directory

  Elke keer dat een job wordt gestart of elke keer wanneer een user aanmeld op zijn computersysteem zal er gezocht worden naar zijn persoonlijke user file

directory, deze bevat alle data die behoort tot de daartoe betreffende user. Elk van deze UFD's behoort tot de grotere klasse van MFD. Elke entry van de MFD is een UFD. MFD kan ook wel worden bezien als de root van de hele tree dat plaatsvindt binnen een file system.

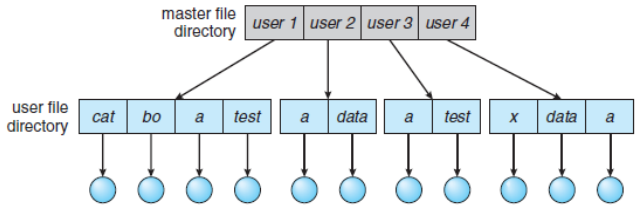


Figure 11.10 Two-level directory structure.

– Micro Kernel

Deze methode verandert de structuur van de OS door alle non essentiële componenten van de kernel te verwijderen en deze te implementeren als system&user-level programma's. Hierdoor ontstaat er een kleinere kernel. De voornaamste taak van de microkernel is dus het creëren van een environment waar dat er communicatie plaatsvindt tussen een client program en verscheidene services dat ook worden geklaard in user space. Message passing is een voorbeeld van communicatie.

Pluspunt: extenderen van de os is zeer makkelijk
Geeft ook meer veiligheid en relabiliteit aangezien de meeste processen als user runnen en niet binnen de kernel. Aangezien bij het falen van een proces niets van het OS aangetast zal worden.

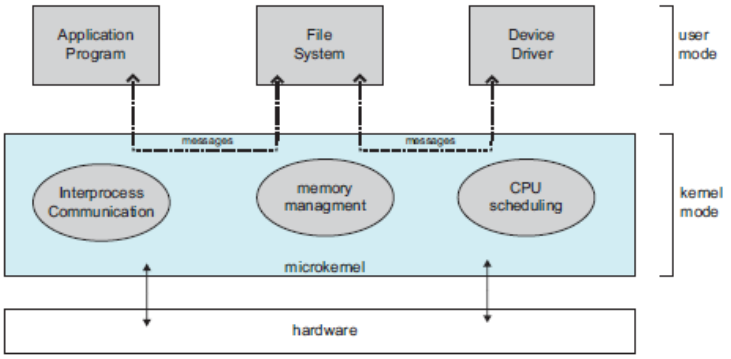
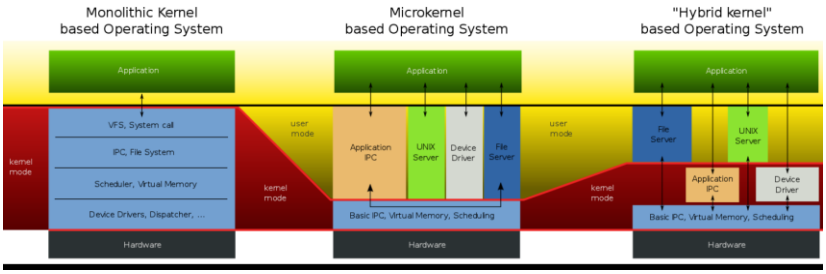


Figure 2.14 Architecture of a typical microkernel.

– Monolithic Operating System

Dit is een OS-architectuur waar dat het hele systeem werkt in kernel-space. Het verschilt van andere architecturen in de zin dat het alleen een hoog leven virtuele interface doet ontstaan tussen OS en hardware niveau.

Alternatieven: Microkernel, Hybrid kernel



– Mount Table

De Mount table houdt een tabel bij met alle mounted devices binnen het computer systeem. Vooraleer je beroep kan doen op enige vorm van storage space, moet jij of het OS deze beschikbaar maken voor gebruik via de file system. Deze tabel bevat dus alle toekenningen van mount blocks (ookwel alle storage devices).

– Naming of IPC

Binnen inter process communicatie bestaat er een manier genaamd message passing. Deze bevat 2 commando's: Send(mesg), Receive(mesg). Bij naming zullen we deze functies een extra parameter aangeven, deze parameter zal de naam zijn van de daartoe behorende proces. Zo zal het verzenden als volgt eruit zien: Send(P, message). Met P een naam van een bepaald proces. De vorm van communicatie is een directe vorm. Beide partijen vernoemen elkaars naam.

Tussen beide partijen hoort er ook een link plaats te vinden. De voorwaarden voor een link zijn:

1) Een link ontstaat wanneer 2 processen met elkaar willen communiceren.

2) Een link is geassocieerd met exact 2 processen.

3) Tussen elk paar van processen kan er maximaal 1 link plaats vinden.

Alternatieven tot directe communicatie: indirecte communicatie, bv sturen naar een mailbox.

- Need To Know Principle

  Een proces zou alleen toelating mogen hebben tot de resources waarover het proces de noodzakelijke autorisatie van bezit. Verder zullen de processen alleen toegang mogen hebben tot de resources waardat het noodzaak voor heeft. Dit laatste is bekend als het need to know priciple. Dit is handig om de schade te beperken indien er zich een foutief proces zou optreden.

  Bv: stel dat P een compiler oproept om bepaalde source code te compilen. De compiler zal dus alleen de noodzakelijke resources hanteren bij het vervolledigen van deze taak. Hier de noodzakelijke sectie van de source code en indien nodig enkele libraries.

- Non-Pre-emptive Scheduling

  We zeggen dat een process non preemptive is wanneer:

  1) Het proces switched van running naar waiting. Bv I/O request of het invokeren van een wait().

  2) Wanneer een proces beëindigd.

  Binnen non preemption zal het toegewezen proces de cpu niet vrijlaten totdat deze is beëindigd. Terwijl bij preemption deze door processen met hogere prioriteiten wel verstoten kunnen worden.

  Preemption veroorzaak race condities wanneer resources over meerdere threads worden gedeeld. Preemption affecteert alsook de design van de operating system kernel.

  FCFS is een voorbeeld van een nonpreemptive algoritme voor disk scheduling.

- One-To-Many Threading Model

  Dit model mapped meerdere user threads naar 1 kernel thread. Thread management wordt gedaan door de thread library in user space -> efficient.

  Negatief: kan blokkeren indien een thread een blocking system call doet, hierdoor zal het hele proces stilstaan. Er is ook nauwelijks sprake is van simultaneïteit. Dit omdat alle user threads worden herleid naar 1 kernel

thread. Er zal dus maar 1 thread tegelijk uitvoeren voor een bepaald proces.

- OpenMP parallel regions

  The simplest way to create parallelism in OpenMP is to use the parallel pragma. A block preceded by the omp parallel pragma is called a parallel region.

  Het model bevat het volgende:

  1) Immediately preceding the parallel block, one thread will be executing the code. In the main program this is the initial thread.

  2) At the start of the block, a new team of threads is created, and the thread that was active before the block becomes the master thread of that team.

  3) After the block only the master thread is active.

  4) Inside the block there is team of threads: each thread in the team executes the body of the block, and it will have access to all variables of the surrounding environment. How many threads there are can be determined in a number of ways; we will get to that later.

- PCB

  Elk proces is vertegenwoordigd als een proces control block. A.k.a. task control block. Deze bevat veel informatie gerelateerd aan een specifiek proces inclusief onderstaanden:

  1) Process state: Deze kan bestaan uit: new, ready, running, waiting, halted etc.

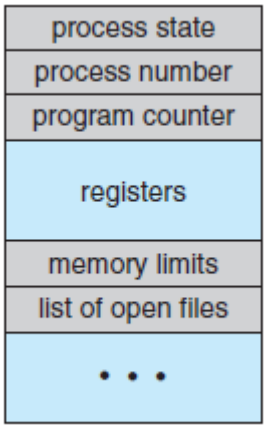
  2) Program counter: bevat het adres van de eerstvolgende instructie die zal moeten worden geëxecuteerd voor dit proces.

  3) CPU-registers: Deze kunnen allerlei typen aan informatie bevatten. Deze bevatten o.a. accumulatoren, registers, stackpointers, etc. samen met de program counter moeten deze opgeslagen worden bij een context switch, zodat achteraf correcte executie kan lopen.

4) CPU-scheduling information: dit bevat een proces priority, pointers naar de scheduling Q's en elke andere scheduling parameter.

5) Memory Management info: Kan informatie bevatten over o.a. de waarden van de base en limit registers, zo ook bv. de page tables.

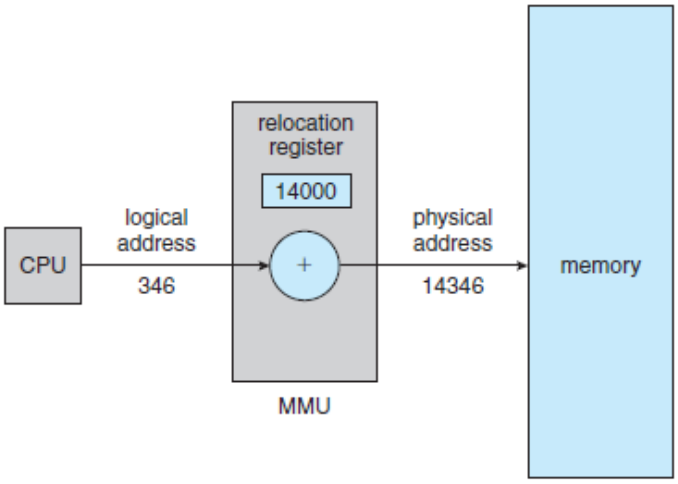
Tegenhanger: TCB(thread)



– Physical Address Space

De adressering gedaan door de cpu wordt beschouwd als de logische adressen. De verwijzingen van de logische adressen naar de memory, daar waar de memory adres registers zijn gedeclareerd, noemen we het fysische adres. Logische adressen/virtuele adressen wijzen naar entries binnen het fysische.

De mapping gedaan tussen beide adresseringen gebeurt door de memory management unit. Er zijn vele manieren om die mapping te implementeren.



- Polymorphic Virus

  Een polymorphic virus verandert zichzelf elke keer dat deze geïnstalleerd wordt om dusdanig detectie te vermijden. De verandering verandert alleen 'the signature' van dit virus. Deze signature wordt gebruikt bij het detecteren van virussen. Meestal is dit een sequentie van bits om een viruscode te herkennen.

  Alternatieven: Multipartite Virus, Encrypted virus, Stealth, Tunneling en armoured virus.

  Zie armoured virus voor meer info over deze.

- POSIX Thread

  /Geen idee

- Priority-Based scheduling

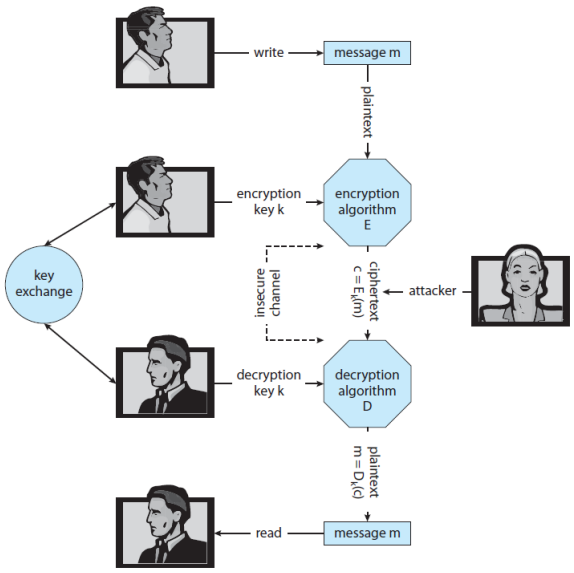
  Real time scheduler respecteert de prioriteit van elke task en dus is dit een algoritme met preemption. Zie schedulers onder Earliest Deadline First

- Private & Public key

  Dit komt voor bij asymmetric encryption.

  Gebruik van verschillende encryption/decryption keys. Zo wordt deze veel gebruikt bij end to end encryptie. Een ontvanger maakt 2 keys, 1 is de public key, eender wie kan die aanvangen en onder deze key iets versturen, alleen de ontvanger kan deze decrypten. Het maakt niet uit wie dit meevolgt, aangezien zij deze encrypte

messages niet zal kunnen ontcijferen, + keys horen niet meer privaat bekend te zijn. Om bidirectionele encryptie te hebben, zal de sender ook een public key moeten generen. Deze vorm is computationeel intensiever. Voorbeeld: RSA



- Process State

  Tijdens executie van een proces verandert de state van dat proces. De staat van dat proces wordt bepaald door de huidige activiteit van dat proces.

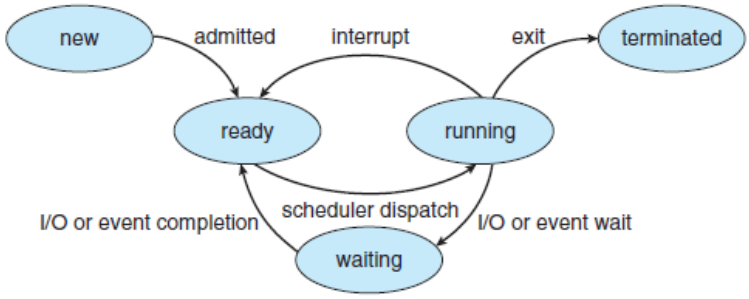


Figure 3.2 Diagram of process state.

- PTBR (Page table base register)

  Elk OS heeft zijn manier van het bijhouden van page tables. Een manier hiervoor is het gebruiken van registers voor page tables op te slaan. Dit gaat voornamelijk bij relatief kleindere tabellen. Bv 256 entries. Tegenwoordig kunnen deze tabellen tot 1 miljoen entries bevatten. Hierdoor zullen we de page table in memory houden en een PTBR wijst naar de

page table. Het veranderen van een page table zal alleen noodzakelijk het register moeten aanpassen. Dit is positief voor de context switch time.

Negatief: De tijd die het erover doet om een user memory locatie te vinden. Bv: we willen locatie L accessen. Eerst moeten we indexeren binnen de page table, met gebruik van de waarde in de PTBR, met in gedachte nemend de offset van L. Deze taak doet een memory access. Deze zal ons een frame number teruggeven wat gecombineerd is met de page offset om dusdanig het werkelijke adres te bepalen. Dus met dit systeem doen we 2 MEMORY ACCESSES OM 1 BYTE te vinden. 1 voor de page table entry en 1 voor de byte -> memory access is vertraagd met een factor van 2. Swapping komt zelfs over het algemeen gunstiger uit.

- Raw Disk:

  Sommige OS laat toe aan sommige programma's om een disk partition te gebruiken waardat er een grote sequentiele array aan logische blocks in staan. Deze toegang heeft geen betrekking tot ene file system data structuur. Deze array noemen we de raw disk.

  Raw-IO gebeurt hierop. Raw-IO bypassed: buffer cache, file locking, space allocatie, file names, directories etc.

- Relative/Absolute Path Names

  Deze termen vinden plaats binnen de file system. Deze paden bepalen waardat de elementen zich bevinden in deze structuur. Relative path names beginnen van een bepaalde directory. Bv: /images/italy Dit terwijl absolute path names beginnen vanaf het begin van de tree: root/user/images/italy.

- Replay Attack

  Dit is de term voor malicious or fraudulent repeat of a valid data transmission. Het is een vorm van een attack, deze zal dus proberen de security te breken/bypassen/breachen. Een voorbeeld hiervan is het herhaaldelijk opvragen van een som van geld. Dit gebeurt dan via message modificatie zodanig dat de boodschap zo realistisch mogelijk zal overkomen bij de onschuldige.

Alternatieven: man in the middle attack (hacker zet zichzelf in het midden van de communicatie flow, masquerading zichzelf als verzender aan de ontvanger en vice cersa), session hijacking (actieve communicatie sessie wordt onderschept door de hacker, dit kan leiden tot man in the middle), masquerading.

Het tegengaan van deze attacks moet op 4 nieveaus gebeuren:

1) Fysisch
2) Humaan
3) OS
4) Netwerk

- Requirements for deadlock

4) Geen preemtie: Dit wil zeggen dat het process die een resource vasthoudt, deze vrijwillig pas loslaat nadat deze zijn taak heeft volbracht.
5) Hold&wait: Een proces moet een bepaalde resource bezetten terwijl het wacht op de vrijlating van een andere resource.
6) Mutual exclusion: Er moet minstens 1 resource zijn die in een nonsharable mode wordt gehouden. Dwz dat er maar 1 process per keer aan deze resource kan geraken. Indien een ander proces deze resource aanvraagt, zal deze moeten wachten op de vrijlating hiervan.
7) Circular wait: Dit is een van de condities om een deadlock te bekomen binnen een systeem van processen die allemaal beroep doen op resources. Zo is er dan een set aan processen P0, P1, P2, ... , Pn, waardat P0 wacht op de loslating van de resource, gehouden door P1, dit analoog voor P1 die wacht op P2 etc, tot Pn-1 die wacht op Pn. Met Pn wachtende op P0.

- Round Robin Scheduling

Dit is een algoritme dat speciaal is ontworpen voor time sharing devices. Het is gelijkaardig aan FCFS scheduling + preemption. De scheduler itereert over de

Q (FIFO) heen en alloceert de cpu tot elk proces voor maximaal 1 tijdskwantum. Na dat de cpu deze taak gedurende 1 tijdskwantum heeft gehanteerd zal er 1 van 2 onderstaanden gebeuren:

1) De task had minder dan 1 tijdskwantum nodig en zal de CPU vrijwillig vrijgeven.

2) Indien de task langer dan 1 tijdskwantum zou willen werken zal er een timer afgaan en een interrupt oproepen, hierna volgt een context switch en zal de task achter op de Q geplaatst worden.

Probleem: de grootte van de timekwantum bepalen, een grote tijd zal resulteren in fcfs alike behaviour. Een korte tijd (1ms) zal resulteren in zeer veel context switches indien er veel tasks zijn. -> we willen large timeQuantum met respect tot de context switch.

Alternatieven:

Multilevel (feedback) queue scheduling, FCFS, SJF (priority scheduling)

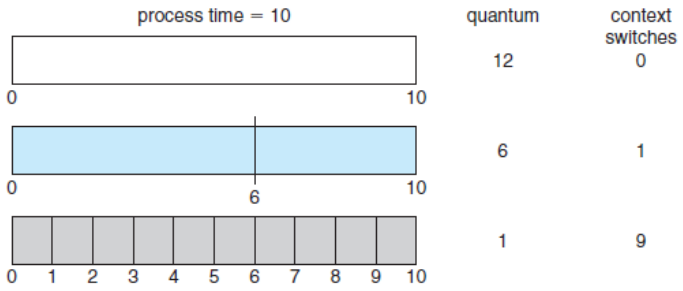


Figure 6.4 How a smaller time quantum increases context switches.

– Sandbox

Het is een manier om een systeem te beveiligen tegen virussen. Net zoals antivirussoftware die actief zoekt naar familiale patronen van virussen waarna het systeem zal gedesinfecteerd worden. Een manier om virussen ook tegen te gaan, is een programma laten runnen in een sandbox, dit is een gecontroleerd/emuleerd deel van het systeem. Het antivirus zal het programma zijn werk laten doen zonder inferentie van de antivirus. Het antivirus zal het gedrag v.h. programma nagaan vooraleer deze code wordt losgelaten unmonitored.

Alternatieven:

1) Safe computing (prevention)

2) Niet openen van suspicious emails.

3) Het herformatteren van de disk op veilige wijze. Voornamelijk richtend op de bootsector.

- SCAN Aglorithm

  Zie C-LOOK

- Scheduling criteria

  Er zijn enkele criteria waardat CPU scheduling algos rekening mee moeten houden binnen hun implementatie. Het optimale algoritme zou dus een perfecte combinatie moeten zijn van onderstaande:

  1) CPU-utilisiatie:

     We willen de cpu zo goed mogelijk gebruiken en utilisatie maximaliseren. Gemiddeld ligt de utilisatie van cpu tussen 40-90%.

  2) Throughput:

     De aantal tasks dat zijn vervolledigd binnen een tijdseenheid noemt men throuhput.

  3) Turnaround time:

     De tijd tussen de invoeging van een proces in de Q met de uiteindelijke tijd wanneer het aan bod komt in de CPU om uit te voeren.

  4) Waiting time:

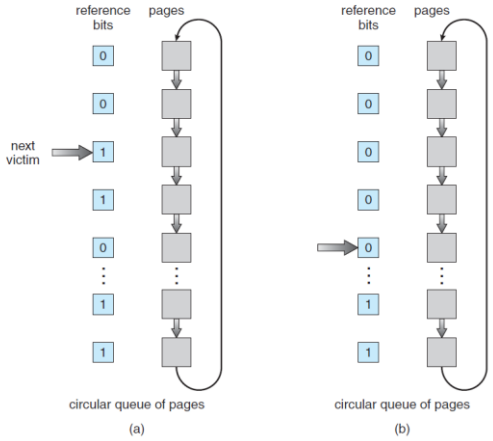
     De sommatie van de perioden waardat een proces zich in een waiting state bevindt in de ready queue. Dit door mede I/O.

  5) Response time:

     Turnaroundtime is niet bepaald het sterkste punt van de meeste algos, response tijd is de tijd die verstrijkt tussen de submissie van een proces en het verschijnen van het uiteindelijke resultaat aan de user.

- Second-Chance Algorithm

  Second chance replacement is een FIFO replacement algoritme. Wanneer een page is geselecteerd, inspecteren we zijn referentie bit. Bij 0 -> replace page. Bij 1 -> we geven het een tweede kans en gaan verder naar de volgende FIFO-pagina. De referentie bit wordt alsook gereset naar 0. De tijdstamp van entry wordt ook gezet naar dat huidige moment. De implementatie hiervan kan gezien worden als een clock algorithm. Zie tekening



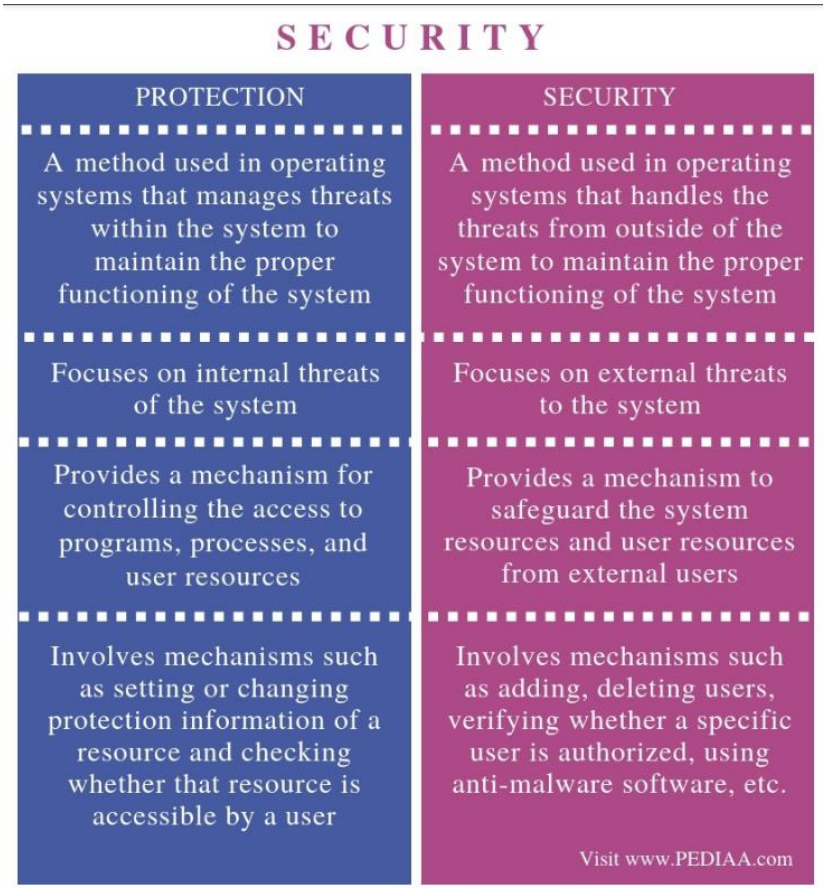
- Sector Slipping/Sector Sparing

  Dit is een alternatief voor sector sparing. Het is een oplossing voor het handelen van bad block recovery. Doordat er veel wordt geschreven en gelezen op disks, kan het zijn dat er bad blocks onstaan, deze zijn niet meer bruikbare locaties om hierin data op te slaan. Op simple disks wordt er manueel gescanned naar bad blocks terwijl deze disk wordt geformatteerd. Hierna worden badblocks geflaggen als unusable. Indien badblocks occureren tijdens normale executie, zal er manueel teruggezocht worden naar deze en deze blokken afsluiten van het OS (linux: badblocks commando). Data dat zich bevond in de bad blocks is meetal verloren.

  Nieuwere disks gaan hier slimmer mee te werk. De controller houdt een lijst bij van alle bad blocks. Deze lijst werd geinitialiseerd bij het low level formateren en wordt doorheen zijn lifespan geupdate. Tijdens deze low level formatting houdt deze enkele sectoren verborgen voor het OS, waarna de controller elke badblock logisch kan omwisselen met deze gespaarde sectoren.

Sector Slipping: Als alternatief op sector sparing kan het ook zo zijn dat er een fout optreedt in logisch block 3. Waarna de eerstvolgende vrije block zich bevindt in 20. Sector slipping remapped al deze blocks, zo zal 19 remappen naar 20, enz totdat 3 zich remapped in 4. 3 wordt dan aangeduid al seen badblock. Dit is uiteraard alleen mogelijk bij kleine fouten. Indien er zich een harderror voordeed, is alle data uit block 3 verloren en zal er een manuele repair moeten plaatsvinden vanuit de laatste backup.

- Security vs Protection



- Sequential access of memory

Informatie wordt geprocessed in order, de ene record na de andere. Deze manier van access is de meest gebruikte.

Alternatieven: Direct Access: file is gemaakt uit logische records. Read & write gebeurt snel in geen particular volgorde. Zijn goed voor directe toegang tot grotere

hoeveelheden aan datagroottes. Bv Databases. Binnen DA moeten de operaties zodanig aangepast worden zodat ze een block nummer nemen als parameter. Dit kan ook bereikt worden door een nieuwe functie: position_file(n), deze wordt opgeroepen alvorens read/write wordt opgeroepen. De parameter zal een relatieve block nummerzijn aangezien de OS logische adressen bijhoudt die dan verwijzen naar absolute adressen. Het gebruik van relatieve block numbers resulteert in de OS die beslist waardat de file hoort te komen -> ALLOCATION PROBLEM. Dit verhelpt de user van segmenten van de file system te accessen die niet behoren tot zijn file.

- Signal/IPC (inter process communication)

  Een signal wordt gebruikt in het UNIX systeem om te laten weten dat er een bepaalde gebeurtenis heeft plaats gevonden. De aanname van een signal kan oftewel synchroon als asynchroon. Eenmaal de signal is geleverd aan een proces, zal deze behandelt moeten worden.

  Synchronous signals: delen door 0, illegale memory acces (deze laatste wordt opgelost door bv het programma te sluiten).

  Asynchronous signals: signalen gecreëerd door gebeurtenissen buiten een environment van een bepaald proces.

  Signal Handler -> Default of user-defined (user defined override de default handler)

  Probleem: in multithreading, waar zal de signal heen moeten?
  1) Deliver the signal to the thread to which the signal applies.
  2) Deliver the signal to every thread in the process.
  3) Deliver the signal to certain threads in the process.
  4) Assign a specific thread to receive all signals for the process.
- SSTF (Disk Scheduling)

  SSTF (Shortest seek time first) scheduling kiest altijd het dichts bij liggende als volgende, gelijkaardig aan shortest job first met process scheduling, maar een probleem bijvoorbeeld is aging. Aging houdt in dat processen die verder gelegen liggen nooit ter sprake komen. Dit kan opgelost worden door een counter/clock

bij te houden die alle tasks onder hoede houdt van zijn arrival time tot het moment dat die er nog inzit. Wanneer de time limit bereikt is, zal ongeacht de prioriteit van deze taak, deze taak uitgevoerd worden.

Alternatieven: FCFS, SCAN, C-SCAN, LOOK, CLOOK.

- Static Linking

  We nemen aan dat een programma is geladen in het geheugen. Echter hebben vele functies nood aan libraries, die indien niet geimplementeerd zijn binnen de binaire file, dus ook gealloceerd moeten worden. Mocht deze toch toegevoegd zijn in de binary file, dan noemen we dit statically linked. Het probleem hierbij is dat elk programma een kopie bevat van overlappende libraries die door meerdere programma's gebruikt worden. Het is efficienter indien we deze eenmalig allemaal zouden moeten laden in memory. Dit doet dynamic linking.

- Stealth Virus

  Dit virus probeert detectie te vermijden door de systemen die malware/virussen zouden detecteren aan te passen. Bv: het zou de read system call kunnen modificeren waardoor het 'het te lezen file', die werd geïnfecteerd, wordt gelezen, dan zal de virus de originele code teruggeven i.p.v. de geïnfecteerde code.

  Alternatieven: Multipartite Virus, Encrypted virus, Polymorphic virus, Tunneling en armoured virus.

- Swap Space

  Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance. The main goal voor het design en implementatie van swap space is to provide the best throughput for the virtual memory system.

  Systemen die bijvoorbeeld swapping implementeren zullen deze voornamelijk gebruiken bij het behouden van een volledige afbeelding van het proces. Het is zo ook veiliger om de swap space die er is te overschatten i.p.v. te onderschatten (onderschatting -> crashen van

bepaalde processen en aborten van anderen). Het negatieve hieraan is dat er disk space wordt gebruikt voor niets, i.p.v. hier files in te kunnen plaatsen.

Swap space kan worden geplaatst in een aparte raw paritite. Hier wordt er gebruik gemaakt van een swap-space storage manager die allocatie en deallocatie doet van storage blocks voor de raw partition.

Implementatie:

sunOS: Swap space is only used as a backing store for pages of anonymous memory, which includes memory allocated for the stack, heap, and uninitialized data of a process.

A swap area may be in either a swap file on a regular file system or a dedicated swap partition. Each swap area consists of a series of 4-KB page slots, which are used to hold swapped pages. Associated with each swap area is a swap map—an array of integer counters, each corresponding to a page slot in the swap area.

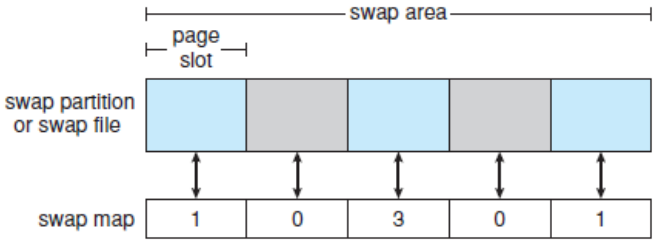


Figure 10.10 The data structures for swapping on Linux systems.

- System Call

System calls are interfaces through which a process communicates with the OS. Each system call has a unique name associated with it (Open, Read, Fork, etc). Each of these names maps to a unique system call number. Each system call in turn causes a software interrupt to occur. Note that multiple system calls can be mapped to the same interrupt.

All the arguments to the system call are pushed into the user stack of the process which invokes the system call. The system call number is pushed as the last argument.

File system calls are used by a process when it has to create, delete or manipulate Data files that reside on

the disk (file system). There are seven file system calls. An interrupt is associated with each system call. All the necessary arguments for a system call are available in the user stack with the system call number as the last argument.

Voorbeelden van file system calls: open close delete create etc.

Process system calls are used by a process when it has to duplicate itself, execute a new process in its place or when it has to terminate itself. There are three process system calls. An interrupt is associated with each system call. All the necessary arguments for a system call are available in the user stack with the system call number as the last argument.

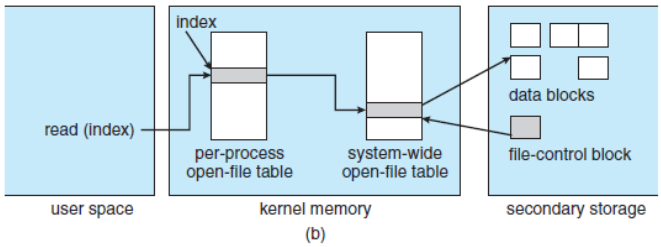
Voorbeelden van process system calls: fork, exec, exit

- System Wide Open Table

  This data structure maintains details about all open files in the system. It is located from words 1344 to 1471 of the memory (in Page 2). System Wide Open File Table consists of a maximum of 64 entries. Therefore, there can be at most 64 open files in the system at any time. Each entry of the System Wide Open File Table occupies 2 words:

  - FAT index: It stores the index of the corresponding file in the FAT. An invalid entry is denoted by -1.
  - File Open Count: File Open Count is the number of open instances of the file. When this becomes zero, the entry for the file is invalidated in the System Wide Open File Table.

  The Per-Process Open File Table in the PCB of each process stores information about files opened by the corresponding process. Each entry in the Per-Process Open File Table has the index to the file's entry in the System-wide Open File Table.



– TestAndSet() (mutual exclusion)

The important characteristic of this instruction is that it is executed atomically. Thus, if two test and set() instructions are executed simultaneously (each on a different CPU), they will be executed sequentially in some arbitrary order. If the machine supports the test and set() instruction, thenwe can implement mutual exclusion by declaring a boolean variable lock, initialized to false.

Alternatieven:

Compare&swap():

The compare and swap() instruction, in contrast to the test and set() instruction, operates on three operands; it is defined in Figure 5.5. The operand value is set to new value only if the expression (*value == exected) is true. Regardless, compare and swap() always returns the original value of the variable value. Like the test and set() instruction, compare and swap() is executed atomically.

Code compare&swap:

```
do {
   while (compare_and_swap(&lock, 0, 1) != 0)
      ; /* do nothing */

      /* critical section */

   lock = 0;

      /* remainder section */
} while (true);
```

Figure 5.6 Mutual-exclusion implementation with the `compare_and_swap()` instruction.

Code voor test&set:

```
do {
   while (test_and_set(&lock))
      ; /* do nothing */

      /* critical section */

   lock = false;

      /* remainder section */
} while (true);
```

```
boolean test_and_set(boolean *target) {
   boolean rv = *target;
   *target = true;

   return rv;
}
```

- Theft of Service (Unauthorized use of resources, such as theft of CPU cycles)

  Alternatieven: Breach of confidentality/integrety/availability, denial of service

- Thread Library

  A thread library provides the programmer with an API for creating and managing threads. There are two primary ways of implementing a thread library.

  The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space.

  The second approach is to implement a kernel-level library supported directly by the operating system. In this case, code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.

  3 grootste thread libraries vandaag: POSIX Pthreads,Windows, and Java.

- Time Quantum x2

  Voornamelijk gebruikt bij Round robin scheduling. Ook wel bekend als time slice. (10-100 ms) Dit is een tijdsunit waardat het proces in kan werken waarna er geswitched zal worden.

- TLB

  TLB of Translation Look-aside Buffer is een speciale, kleine snelle, lookup hardware cache van de pagina

tabel. Elke ingang van de TLB bestaat uit een key en een waarde. Wanneer het wordt aangeboden met een item, wordt het item vergeleken met alle keys tegelijk. De TLB wordt gebruikt voor een snellere vertaling van virtuele adressen naar fysieke adressen, dit is ter vervanging van het controleren van alle elementen van een page table.

- Trap(Interrupt)

  Ookwel bekend als een system interrupt, dit is een instructie met een operand die de gewenste kernel service aanvraagt. Eerst context switch naar kernel mode, waarna gedispatched wordt naar de kernel routine die de gewenste interrupt kan handlen. Een trap heeft een vrij lage prioriteit vergeleken met interrupts van devices. Denk maar aan de situatie waardat er gekozen moet worden tussen een system call van een app. of de service toewijzen aan een device controller voordat zijn fifo Q overflowd en data verliest.

- Trap Door(backdoor)

  Het creëren van een breach door het security systeen van een bepaald systeem is noodzakelijk om de kernel en user leven te kunnen manipuleren, uit standpunt van een hacker. Het is aantrekkelijk om ongeautoriseerd binnen een bepaald systeem te geraken maar het zou beter zijn, als hacker, om een back-door achterwege te laten. Deze zal een hole vormen in de security wall, deze zal altijd informatie lekken ook wanneer de originele exploit is gepatched. Er zijn verschillende implementaties van een back-door:

  1) Trojan horse:

  2) Trap Door:

  3) Stack&buffer overflow:

- Trojan Horse

  A Trojan horse or Trojan is een type malware dat vermomd is als legitieme software. Trojaanse paarden kunnen worden ingezet door cyber-dieven en hackers die proberen toegang te krijgen tot de systemen van gebruikers. Gebruikers worden meestal misleid door enige vorm van sociale Engineering in het laden en uitvoeren van Trojaanse paarden op hun systemen. Eenmaal geactiveerd, kan Trojaanse paard de

cybercrimineel in staat stellen te bespioneren op u, het stelen van uw gevoelige gegevens, en krijgen backdoor access tot uw systeem. Deze acties kunnen zijn:

Deleting data

Blocking data

Modifying data

Copying data

Disrupting the performance of computers or computer networks

In tegenstelling tot computervirussen en wormen, zijn Trojaanse paarden niet in staat om zichzelf te repliceren.

Alternatieve backdoors: trap door, stack&buffer overflow

- Two Level threading

  Dit is een variant op de many-to-many model, waar er toch van meerder user level threads naar minder kernel threads wordt overgegaan alsook de user level thread gebonden te zijn aan een kernel thread.

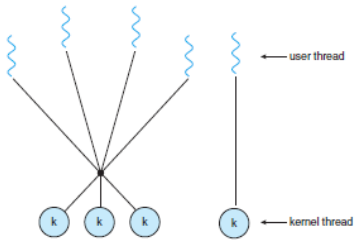


Figure 4.8 Two-level model.

- User Thread

  Een user thread is normaal gezien gecreerd door ene threading library, scheduling is ook beheert door de threading library (Which runs in user mode). Alle user threads behoren tot het proces dat deze heeft opgeroepen/geinvokeerd. Het grootste verschil met kernel threads valt te zien in het gebruik van multiprocessor systems, user threads die volledig beheerd zijn door de threading library kunnen niet in

parallel runnen, dit impliceert wel dat ze op een singleprocessor goed runnen. Door dat kernel threads gebruik maken van de kernel scheduler, kunnen verschillende kernel threads runnen op verschillende CPUs. Elk systeem implementeert threading anders.

Many to many, one to one (laat wel toe binnen multiprocessors meerdere threads simultaan te laten runnen), many to one.

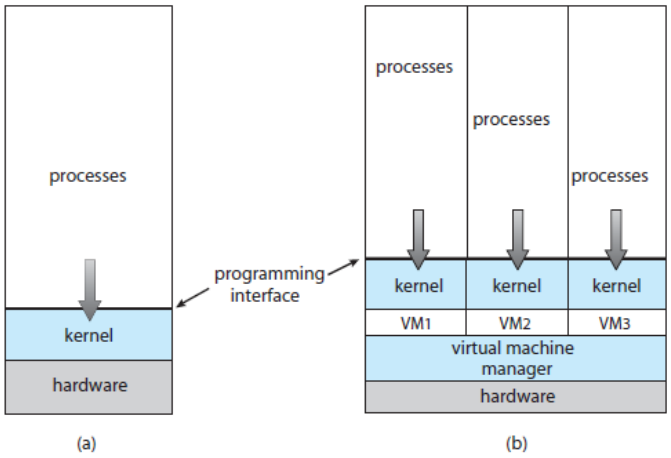
Positief: very portable

- Virtual Machine Manager

  Ookwel bekend als de hypervisor. Deze creëert en runned virtuele machines door een interface aan te bieden dat identiek is aan de host. Elke guest process (meestal een OS) krijgt een perfecte kopie van de host. Een enkel machine kan dus meerdere OS's kunnen.

  De implementaties van VMM's varieren enorm. Onderstaande opties geven een beeld hiervan:

  1) Type 0 hypervisors: Dit zijn hardware based solutions die support providen voor virtuele machine creatie.
  2) Type 1 hypervisors: Operating system alike software die gebouwd is om virtualisatie te providen.
  3) Type 2 hypervisors: Dit zijn standaard OS'en die dat VMM features kunnen aanbieden aan guest OS'en
  4) Paravirtualizatie: een techniek om zodanig de guest OS in samenwerking te laten werken met de VMM voor optimalisatie
  5) Programming-Environment Paravirtualizatie: creëren van een geoptimaliseerd virtueel systeem (oracle java, .NET)
  6) Emulators: Applicaties geschreven voor een bepaalde hardware environment kan gerunned worden op een compleet andere hardware environment
  7) Application containment



- Volume Control Block

  Volume control block is a standard Operating System queue that's maintained in the system heap. It contains a volume control block for each mounted volume. A volume control block is a nonrelocatable block that contains volume-specific information.

- Waiting Time

  Binnen het concept van I/O requests horen we in de tijd dat het proces wacht op io, iets nuttig te doen met de cpu. Anders zou deze gewoon idle zijn. Hier komt multiprocessing van te pas. The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue.Waiting time is the sum of the periods spent waiting in the ready queue.