Uitwerking Tentamen Operating Systems

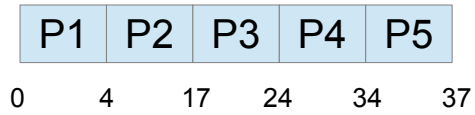
Maandag 15 juni 2015

Belangrijk: de gegeven antwoorden vormen één mogelijke uitwerking van het tentamen. Echter zijn er bij vele vragen meerdere correcte antwoorden mogelijk!

Opgave 1

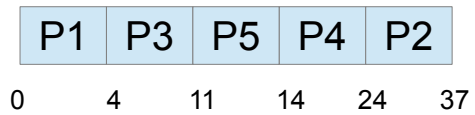
1. Let op: de GANTT charts zijn niet op schaal!

FCFS (First Come First Serve)



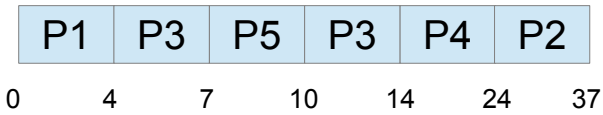
P1: 0 – 0 = 0
P2: 4 – 0 = 4
P3: 17 – 3 = 14 63 / 5 = 12.6
P4: 24 – 6 = 18
P5: 34 – 7 = 27

SJF (Shortest Job First, non-preemptive)



P1: 0 – 0 = 0
P2: 24 – 0 = 24
P3: 4 – 3 = 1 37 / 5 = 7.4
P4: 14 – 6 = 8
P5: 11 – 7 = 4

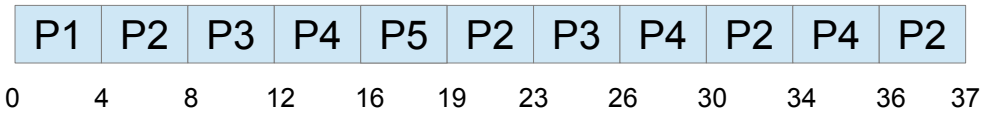
SRTF (Shortest Remaining Time First, preemptive)



P1: 0 – 0 = 0
P2: 24 – 0 = 24
P3: (4 – 3) + (10 – 7) = 1 + 3 = 4 36 / 5 = 7.2
P4: 14 – 6 = 8
P5: 7 – 7 = 0

RR (Round robin), time slice = 4

Geef bij RR altijd aan hoe er wordt omgegaan met het plaatsen van processen op de ready queue indien deze later arriveren. In het algemeen gaan wij er hier van uit dat processen die later arriveren niet vooraan de queue worden gezet, maar achteraan aansluiten.



P1: (0 – 0) = 0
P2: (4 – 0) + (19 – 8) + (30 – 23) + (36 – 34) = 4 + 11 + 7 + 2 = 24
P3: (8 – 3) + (23 – 12) = 5 + 11= 16 69 / 5 = 13.8
P4: (12 – 6) + (26 – 16) + (34 – 30) = 6 + 10 + 4 = 20
P5: (16 – 7) = 9

2. CPU utilisatie geeft aan in welke mate de CPU wordt gebruikt. Een 100% CPU utilisatie geeft aan dat de CPU volledig bezet is en zou suggereren dat de CPU 100% van de tijd nuttig bezig is. Context switches zijn echter volledig overhead. Het kan dus voorkomen dat alhoewel er sprake is van 100% CPU utilisatie er (zeer) vele context switches plaatsvinden waardoor een groot deel van de CPU tijd wordt verspild.

3. Merk op dat een hoog aantal vrijwillige context switches duidt op een hoog aantal I/O requests / sleeps en dus korte CPU bursts. Dit is een karakteristiek van een interactief of I/O bound proces. Een hoog aantal onvrijwillige context switches duidt op het feit dat de proces vooral aan het rekenen is en geen I/O requests of dergelijke uitvoert. De CPU bursts zijn lang: CPU bound proces.

We kunnen nu de scheduler zo inrichten dat we op basis van het bovenstaande bepalen of een proces of interactief of een CPU-bound proces is. We richten de twee niveaus als volgt in:

Level 1: het interactieve level, scheduler: round robin met kleine time slice.
Level 2: het background job level, scheduler: FCFS of RR met grote time slice (beide OK).

Promotie: indien gedurende een bepaald tijdinterval wordt geconstateerd dat een proces een laag aantal onvrijwillige context switches en hoog aantal vrijwillige context switches heeft.
Degradatie: indien gedurende een bepaald tijdinterval wordt geconstateerd dat een proces een hoog aantal onvrijwillige context switches en laag aantal vrijwillege context switches heeft.

Opgave 2

1. Deze faciliteit wordt gebruikt om een “user mode” en “kernel mode” te creëren. Programma's draaien in user mode en de operating system kernel in kernel mode. Via een system call wordt er van user naar kernel mode geschakeld. De kernel heeft de volledige controle over het systeem toegekend en heeft de mogelijkheid om privileged instructies uit te voeren, een user mode programma heeft die mogelijkheid niet.

2. Mogelijke consequenties:
- Applicaties kunnen in kernel geheugen schrijven.
- Applicaties kunnen in elkaars geheugen schrijven.
- Applicaties kunnen direct bij de hardware.

3. We verwachten de volgende aspecten in de beschrijving:
- Een monolitische kernel bestaat uit 1 binary / executable.
- In een monolitische kernel draait alle kernel code in kernel mode.
- Microkernel: alleen het broodnodige bevindt zich in de kernel en wordt gedraaid in kernel mode.

- Andere delen van de microkernel zijn aparte processen en draaien in user mode.
- Binnen een monolitische kernel kunnen er direct function calls worden gedaan tussen de verschillende delen. Bij microkernels zijn dit aparte processen (in user mode) en moet er een vorm van interproces communicatie worden gebruikt.

4. Een voorbeeld van een antwoord dat zou worden goedgekeurd:

We gaan uit van 4 modes en delen deze als volgt in:

- *Mode 0*: kernel-mode, hierin draait de microkernel en alles is toegestaan.
- *Mode 1*: driver-mode, driver processen kunnen niet direct bij kernel geheugen, maar mogen wel direct de hardware benaderen.
- *Mode 2*: service-mode, hierin draaien de verschillende microkernel services. Deze services zijn processen en kunnen niet direct bij de kernel. Wel hebben deze processen meer privileges dan user-mode processen.
- *Mode 3*: user-mode, normale user mode voor user processes.

Een beargumenteerde beschrijving van 3 modes (kernel, driver, service/user) of (kernel, service/driver, user) wordt ook goed gerekend.

Opgave 3

1. SSTF: Shortest Seek Time First

165, 155, 149, 125, 73, 53, 222, 272, 312, 316, 338, 387

C-SCAN: Circular SCAN, 1 richting op

165, 222, 272, 312, 316, 338, 387, **399, 0,** 53, 73, 125, 149, 155

C-LOOK:

165, 222, 272, 312, 316, 338, 387, 53, 73, 125, 149, 155

2. Disk bandbreedte is het aantal bytes dat wordt gelezen/geschreven gedeeld door de tijd tussen het eerste verzoek aan de disk en het afronden van het laatste verzoek.

Nu geldt dat hoe sneller alle verzoeken worden afgerond, hoe hoger de disk bandbreedte. Een groot deel van de tijd van disk access gaat op aan seek time. Door disk scheduling algoritmen toe te passen wordt de seek time zo klein mogelijk gemaakt, waardoor de disk bandbreedte toeneemt.

3. In dit geval is er sprake van twee disk schedulers: één in de host en één in de gast. Een I/O operatie gestart door de gast wordt dus twee keer gescheduled. Voor de gast is ook de disk drive gevirtualiseerd, de gast schrijft niet direct naar de disk drive toe. Dit kan op verschillende manieren zijn ingericht, bijvoorbeeld met een image file. De volgorde waarin de blokken van de image file zijn opgeslagen op een fysieke disk is totaal onafhankelijk van de cylindernummering van de virtuele drive. Dus alhoewel de gast probeert te schedulen op basis van cylindernummer van de virtuele drive, is de kans groot dat dit leidt tot random toegangspatroon naar een fysieke drive (*). In slechte gevallen kan de disk scheduler van de host dit niet corrigeren. De conclusie is dus dat een disk scheduler in de gast helemaal geen toegevoegde waarde heeft en beter kan worden uitgezet zodat I/O requests zonder manipulatie bij de host terechtkomen.

Alternatief bij (*), in plaats van een image file kan een virtuele drive ook zijn afgebeeld op een fysieke partitie. In dit geval houdt de disk scheduler van de gast geen rekening van I/O requests naar

andere delen van de fysieke schijf. Alhoewel in dit geval het toegangspatroon naar de fysieke partitie wel zinvol is, is het nog steeds beter de scheduling te laten gebeuren door de host die het overzicht over de hele drive heeft. In het geval van LVM e.d. kan het nog steeds zo zijn dat de cylindernummering van de logical volume niet overeenkomt met dat voor de physical volume.

Opgave 4

1. De volgende elementen worden verwacht in de uitleg:
   - Tekening virtuele adresruimte proces.
   - Tekening fysiek geheugen.
   - Fysiek geheugen moet in-memory kopie van een file op de disk bevatten. Eventueel kunnen ook de diskblokken schematisch worden weergegeven. Deze blokken kunnen in elke volgorde in fysiek geheugen staan.
   - De pages met file data moeten aaneengesloten in de virtuele adresruimte staan.
   - Relatie tussen fysieke en virtuele pages moet zijn aangeduid.
   - Eventueel kan ook een tweede virtuele adresruimte worden getekend.

2. Bij standaard system calls ziet de verwerking van files er als volgt uit:

```
fh = open(…);
while (true)
  {
      read(…);
      write(…);
   }
```

Voor elke file operatie is een system call nodig. In het geval van memory-mapped files ziet het bovenstaande er als volgt uit:

```
fh = open(...);
data = mmap(fh, …);
while (true)
  {
      tmp = data[…];
      data[…] = tmp2;
   }
```

Alleen voor het uitvoeren van open() en mmap() worden system calls gebruikt, daarna vinden alle operaties plaats middels reads/writes naar pointers naar virtueel geheugen. Voor deze operaties zijn geen system calls (en dus switches naar kernel mode) nodig! Omdat er veel minder system calls worden uitgevoerd in het geval van memory-mapped files, is de overhead vele malen lager.

3. De TLB reach geeft aan hoeveel geheugenruimte er kan worden benaderd als de TLB volledig in gebruik is en er geen TLB misses hoeven te worden veroorzaakt. De invloed die TLB reach heeft op memory mapped files is de begrenzing van de grootte van files die efficiënt, zonder telkens TLB misses te veroorzaken, kunnen worden verwerkt. Als de data waarop wordt gewerkt groter is dan de TLB reach, dan zijn TLB misses onvermijdelijk en deze hebben dure page walks tot gevolg.

Karakteristieken van programma's waarin TLB reach naar voren komt zijn bijvoorbeeld:
   - Opereert op een hot set van data groter dan de TLB reach.
   - Leest een zeer grote file (groter dan TLB reach) meerdere malen van begin tot eind, hierdoor worden meerdere malen TLB misses veroorzaakt voor dezelfde data.
   - Random access naar een zeer grote file. Waar je voor kleine files weinig TLB misses zal zien (file valt geheel binnen TLB reach), zal dit voor hele grote files niet zo zijn. In het slechte geval veroorzaakt elke file access een TLB miss.

4. Een proces dat te lijden heeft onder de invloed van TLB reach is een proces vele TLB misses genereert. At run-time kan dit worden vastgesteld door het aantal TLB misses van een proces te monitoren.

Opgave 5

1. In het geval van een crash wordt een file system (mogelijk) in een inconsistente staat achtergelaten. De metadata kan dus corrupt zijn geraakt. Door middel van consistency checking wordt het gehele file system als het ware nagelopen en worden alle fouten gecorrigeerd. Voor grote file systemen kost het proces van consistency checking zeer veel tijd. Journaling file systems lossen dit probleem op door technieken van databasesystemen te gebruiken. Voordat de system call controle terug geeft aan een programma, wordt de benodigde verandering aan de metadata vastgelegd in de transactielog. Even later zal een ander proces de transacties uit de log ook daadwerkelijk verwerken in het file system. In het geval van een crash zullen de transacties die in de log staan (en waar de system call dus return heeft gedaan), maar niet zijn verwerkt, worden verwerkt bij het booten. Dus in plaats van een langdurige consistency check bij het opstarten is alleen nog een replay van de transactie log nodig.

2. In het geval van virtualisatie draaien de gasten niet in kernel-mode. Een gevirtualiseerd operating system zal echter wel priviliged instructies moeten uitvoeren die niet in user-mode zijn toegestaan. Een virtual machine moet een exacte kopie van de originele machine zijn, dus moet het mogelijk worden gemaakt voor de gast om deze priviliged instructies correct uit te voeren. Dit kan worden bereikt met een trap-and-emulate mechanisme. Als een gast een priviliged instructie uitvoert vindt er een trap naar de host plaats. De host zal vervolgens deze instructie “emuleren”: de instructie wordt door de host in kernel-mode correct uitgevoerd, zodat het voor de gast lijkt alsof de instructie door de originele hardware is uitgevoegd. Dat wil zeggen: het resultaat is precies hetzelfde.

3. In een virtual memory management systeem is het voor read-only pages geen enkel probleem om 1 fysieke kopie van de page te delen met meerdere processen. Als een proces naar de page probeert te schrijven, zal deze schrijfactie worden afgevangen. Met de copy-on-write techniek wordt het ook mogelijk om read-write pages in eerste instantie te delen met andere processen. Dit gaat als volgt in zijn werk: in eerste instantie is een gedeelde page gemarkeerd als read-only. Als een proces naar deze page probeert te schrijven vindt er een page fault plaats. Het besturingssysteem verifieert nu of het proces naar deze page mag schrijven. Zo ja, dan wordt er van deze page een kopie gemaakt waar dit proces wel naar toe mag schrijven. Dus: in geval van een write wordt er een copy gedaan.

4. Het probleem van priority inversion ontstaat wanneer een proces met een lage prioriteit een lock heeft op een resource, wat een proces met een hoge prioriteit wil benaderen. Als een gevolg hiervan moet het proces met hoge prioriteit hierop wachten. Als nu het proces met lage prioriteit gepre-empt wordt door een proces met de medium prioriteit, dan zorgt het medium prioriteit proces ervoor dat het hoge prioriteit proces wordt geblokkeerd. Dus een proces met een lagere prioriteit zorgt ervoor dat een proces met hogere prioriteit wordt geblokkeerd.

Een mogelijke oplossing is priority inheritance: zodra een proces een lock neemt op een resource dat ook wordt gebruikt door een proces met een hogere prioriteit, dan krijgt het proces met de lage prioriteit tijdelijk die hogere prioriteit. Bij het vrijgeven van de resource wordt deze wijziging ongedaan gemaakt. Hierdoor kan het proces dat de lock heeft niet meer worden gepre-empt door het medium prioriteit proces.