Tentamen Operating Systems

Dinsdag 14 juni 2016, 10:00 – 13:00

Examinator: dr. K. F. D. Rietveld

- Het tentamen is **gesloten boek**, dus het is niet toegestaan om het tekstboek, slides of eigen gemaakte aantekeningen te gebruiken.
- Alleen rekenmachines waarin geen teksten kunnen worden opgeslagen zijn toegestaan. Het gebruik van grafische rekenmachines is **niet** toegestaan.
- De vragen mogen worden beantwoord in het Nederlands en Engels.
- Beargumenteer al uw antwoorden. Aan antwoorden zonder uitleg of uitwerking worden geen punten toegekend.

- Het aantal opgaven is 5 met een totaal van 18 onderdelen. Het tentamen bestaat uit 4 pagina's.
- Achter elk onderdeel staat tussen vierkante haken het te behalen aantal punten.
- Het totaal aantal te behalen punten is 100.

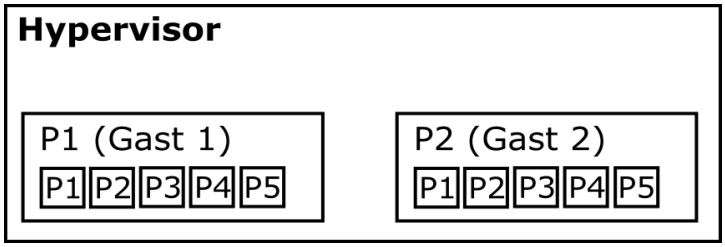
Opgave 1 – Processen & Virtualisatie [18 punten]

Beschouw de volgende processen bestaande uit een enkele CPU burst:

Proces	Arrival time	CPU burst
P1	0	9
P2	0	4
P3	5	6
P4	6	4
P5	6	5

1. Illustreer de werking van de volgende vier CPU scheduling algoritmen door de Gantt chart uit te tekenen: FCFS, SJF, SRTF en RR (time slice = 4). Bereken voor elk van deze schedules de gemiddelde wachttijd. **[5 punten]**

We bekijken nu een virtualisatie-opstelling waarin er sprake is van een hypervisor en twee gast operating systems. De hypervisor heeft één fysieke CPU tot zijn beschikking. Beiden gasten hebben elk één (virtuele) CPU en een eigen, **onafhankelijke** processcheduler. Elke virtuele CPU van een gast wordt binnen de hypervisor als een normaal proces gezien. Een schematische weergave is te zien in de volgende figuur:



2. Neem aan dat Gast 1 de procesmix als gegeven bij onderdeel 1) uitvoert met een FCFS scheduling en Gast 2 voert diezelfde procesmix uit met een RR scheduling. De hypervisor past de RR scheduler toe met een time quantum van 5. Teken de Gantt chart voor de fysieke CPU waarin is te af te lezen welke gast op elk moment actief is **en** welk proces er binnen de gast actief is. **[8 punten]**

Het komt voor dat een gast OS wordt gebruikt voor het uitvoeren van een interactieve workload, bijvoorbeeld een grafische terminalserver. Wanneer vele andere gast systemen binnen diezelfde hypervisor een computationele workload hebben, komt dit de *latency* binnen het interactieve gast OS niet ten goede. Bij het werken in de grafische interface zal er steeds een lichte vertraging te merken zijn.

3. Leg uit hoe dit komt en doe een concreet voorstel voor een mechanisme om interactieve workloads te detecteren en voorrang te geven binnen deze opstelling. **[5 punten]**

Opgave 2 – System Structures [20 punten]

1. Moderne besturingssystemen zijn “interrupt driven”. Leg uit wat hiermee wordt bedoeld. **[5 punten]**

In veel computerarchitecturen wordt er gebruik gemaakt van een Interrupt Vector, of Interrupt Descriptor Table.

2. Leg uit waar deze Vector of Table voor dient en hoe deze wordt gebruikt in de implementatie van een besturingssysteem. **[5 punten]**
3. Waarom is het van belang dat het voor processen in *user-mode* **niet** mogelijk is om deze tabel aan te passen? Licht uw antwoord toe. **[5 punten]**

Wanneer er gebruik wordt gemaakt van *interrupt-driven I/O* worden er vele interrupts gegenereerd voor grote data transfers. Direct Memory Access (DMA) is een oplossing voor dit probleem.

4. Leg duidelijk uit hoe DMA werkt en waarom dit de problemen met *interrupt-driven I/O* oplost. **[5 punten]**

Opgave 3 – Memory Management [20 punten]

Om te pogen de *page fault rate* onder controle te houden is het belangrijk om een effectief *page replacement* algoritme te implementeren. *Page replacement* algoritmen worden in de regel geëvalueerd aan de hand van het bewezen optimale algoritme genaamd OPT of MIN.

1. Omschrijf de werking van dit OPT of MIN algoritme en leg uit waarom dit algoritme niet als zodanig in een besturingssysteem kan worden geïmplementeerd. **[5 punten]**

Gegeven de volgende *reference string* van *page numbers*:

1, 4, 6, 1, 4, 1, 3, 2, 6, 1, 7, 9, 7, 7, 8, 5, 3, 7, 3, 8, 6, 7

en een fysiek geheugen bestaande uit 4 page frames.

2. Bereken het aantal *page faults* dat zal plaatsvinden bij het verwerken van deze *reference string* met de algoritmen FIFO, LRU en OPT. **[5 punten]**

Stel nu dat we de mogelijkheid hebben om één of twee pages aan te duiden die moeten worden vastgezet en niet in aanmerking komen voor *replacement*.

3. Evalueer de effectiviteit van deze functionaliteit voor de gegeven *reference string*. Welke page(s) zou u vastzetten? Leidt het gebruik van deze functionaliteit wel of niet tot een verbetering? Onderbouw uw conclusies. **[10 punten]**

Opgave 4 – File Systems [22 punten]

Het opslaan van bestanden aan de hand van *linked allocation* kan op twee manieren worden geïmplementeerd. Een pointer naar het volgende blok kan steeds in het huidige blok worden opgeslagen, of in een aparte tabel (FAT). We gaan uit van diskblokken van 512 bytes en blokpointers van 4 bytes.

1. Leg, aan de hand van duidelijke illustraties, uit hoe in beide gevallen een bestand ter grootte van 6 blokken wordt opgeslagen. **[5 punten]**

Stel dat de grootte van het bestand ("*file size*") als opgeslagen in het file control block corrupt is geraakt.

2. Bereken voor beide gevallen onder 1) hoeveel *disk seeks* er nodig zijn om de grootte van het bestand opnieuw te bepalen. Ga ervan uit dat hetzelfde blok nooit meer dan eens wordt gelezen, aangezien het blok na eenmaal lezen beschikbaar is in de *buffer cache*. **[4 punten]**

Een groot probleem met *linked allocation* is het corrupt raken van blokpointers.

3. Doe een voorstel voor een aanpassing of toevoeging aan deze opslagmethode waarmee het mogelijk wordt corrupt geraakte blokpointers te repareren. Geef ook de bijbehorende pseudocode voor het detecteren en repareren van fouten. **[7 punten]**

Voor bestandssystemen groter dan 2 TiB ($2 \cdot 2^{40}$ bytes), zijn blokken van 512 bytes en blokpointers van 4 bytes niet meer afdoende.

4. Omschrijf twee manieren om bestandssystemen groter dan 2 TiB te kunnen ondersteunen en bespreek de voor- en nadelen. **[6 punten]**

Opgave 5 – Varia [20 punten]

1. Wat wordt verstaan onder *swapping*? **[5 punten]**
2. Hoe wordt een *spinlock* geïmplementeerd en hoe komt deze aan zijn naam? **[5 punten]**
3. Wat houdt het *mounten* van een bestandssysteem in? Waarom is dit nodig? **[5 punten]**
4. Leg uit waarom het gebruik van *pipelines* in UNIX systemen zoals

   ```
   cat bestand.txt | sort
   ```

   niet kan worden geïmplementeerd met behulp van threads binnen eenzelfde proces en er hiervoor altijd meerdere processen nodig zijn. **[5 punten]**