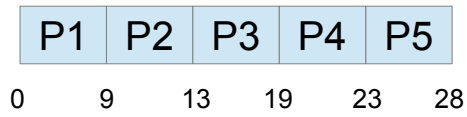
Uitwerking Tentamen Operating Systems
Dinsdag 14 juni 2016

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

Opgave 1

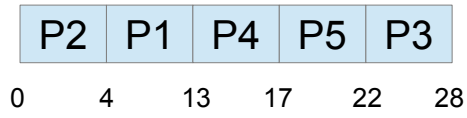
1. Let op: de GANTT charts zijn niet op schaal!

FCFS (First Come First Serve)



P1: 0 – 0 = 0
P2: 9 – 0 = 9
P3: 13 – 5 = 8 47 / 5 = 9 2/5
P4: 19 – 6 = 13
P5: 23 – 6 = 17

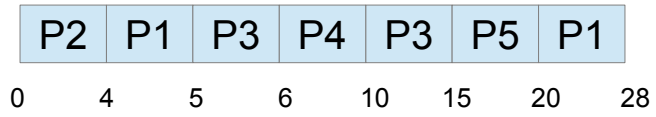
SJF (Shortest Job First, non-preemptive)



P1: 4 – 0 = 4
P2: 0 – 0 = 0
P3: 22 – 5 = 17 39 / 5 = 7 4/5
P4: 13 – 6 = 7
P5: 17 – 6 = 11

SRTF (Shortest Remaining Time First, preemptive)

Tijdstip 5: P1 heeft 8 over, P3 heeft 6 over. P3 wordt gekozen.
Tijdstip 6: P3 heeft 5 over, P4 heeft 4 over. P4 wordt gekozen.
Tijdstip 10: P3 heeft 5 over, P5 heeft 5 over, P1 heeft 8 over. FCFS to break tie: P3 wordt gekozen.

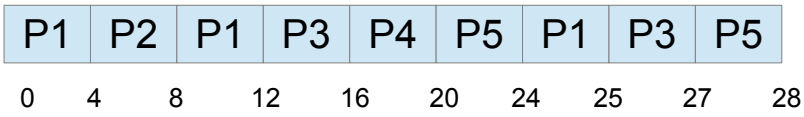


P1: (4 – 0) + (20 – 5) = 4 + 15 = 19
P2: 0 – 0 = 0
P3: (5 – 5) + (10 – 6) = 0 + 4 = 4 32 / 5 = 6 2/5
P4: 6 – 6 = 0
P5: 15 – 6 = 9

RR (Round robin), time slice = 4

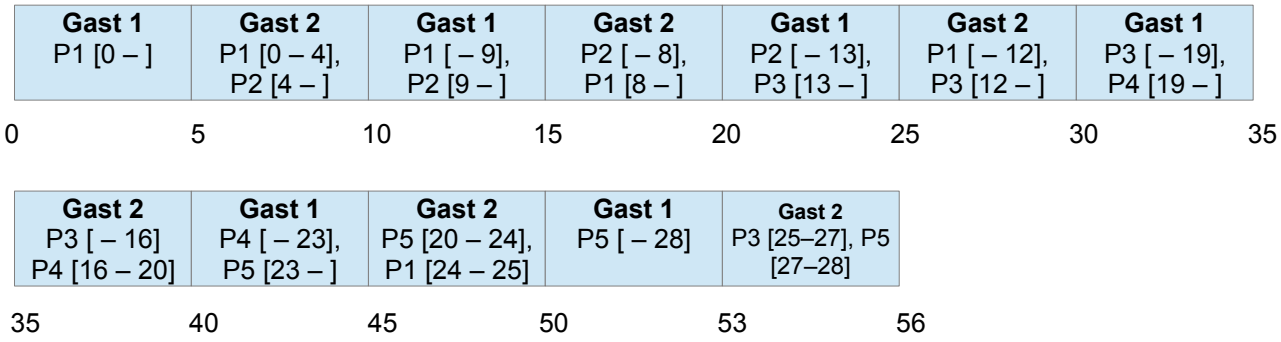
Geef bij RR altijd aan hoe er wordt omgegaan met het plaatsen van processen op de ready queue

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) + (8 – 4) + (24 – 12) = 0 + 4 + 12 = 16
P2: (4 – 0) = 4
P3: (12 – 5) + (25 – 16) = 7 + 9 = 16 63 / 5 = 12 3/5
P4: (16 – 6) = 10
P5: (20 – 6) + (27 – 24) = 14 + 3 = 17

2. Er wordt een korte uitleg verwacht dat de hypervisor steeds één van de gasten kiest en de gast daarbinnen de procesmix als onder 1. afloopt.



3. *Uitleg:* rekentijd wordt gedeeld met andere gasten, bijvoorbeeld via round-robin. Hierdoor krijgen de interactieve processen niet altijd voorrang. Als het quantum op is, zal er worden geswitched naar een andere gast. Omdat er steeds moet worden gewacht op het volgende time quantum, zal de applicatie 'stroperig' aanvoelen.

Voorstel: het probleem is dat interactieve processen steeds kunnen worden onderbroken. Een voorstel zou zijn om gasten met interactieve applicaties een hogere prioriteit te geven en de hypervisor een prioriteit-gebaseerde scheduler te laten gebruiken ipv een round-robin scheduler.

(Let op: vele andere antwoorden zijn mogelijk! In veel gevallen zal het antwoord neer moeten komen op het geven van extra informatie over de aard van de code die in de gast draait.)

Opgave 2

1. Interrupts worden gegenereerd door randapparatuur, zoals toetsenbord, muis, disk maar ook de timerchip. Moderne operating systemen zijn interrupt-driven in de zin dat deze acteren op basis van de binnenkomende interrupts. Zonder interrupts zou een OS niet worden voorzien van invoer en zou het oneindig lang niets aan het doen zijn.

2. In deze tabel worden de adressen van functies opgeslagen die een bepaalde interrupt afhandelen, voor verschillende interrupts (apparaten) wordt er een andere functie gebruikt. Het operating systeem dient deze tabel te initialiseren met de desbetreffende functies, om de aansturing van de verschillende apparaten correct te implementeren.

3. Als een user-mode proces de tabel kan aanpassen, dan heeft de operating system kernel de algehele protectie van het systeem niet meer in handen en kan het systeem worden gesaboteerd. Het is dan bijvoorbeeld mogelijk om binnenkomende data te onderscheppen, interrupts van apparaten uit te schakelen. Vooral voor de timer interrupt is dit een probleem, omdat de scheduler op die wijze kan worden omzeild.

4. Bij Direct-Memory Access krijgen controllers van apparaten direct toegang tot het geheugen van de computer. De controller kan dan direct de binnenkomende data wegschrijven in het geheugen. In plaats van een interrupt te genereren voor elke datatransfer, wordt er maar één interrupt gegenereerd, namelijk wanneer de datatransfer compleet is. Hierdoor worden er aanzienlijk minder interrupts gegenereerd, maar nog belangrijker de CPU kan zich bezighouden met belangrijkere taken.

Opgave 3

1. Het idee van het OPT algoritme is om die page te vervangen die de langste tijd niet zal worden gebruikt. Dit kan niet als zodanig in een besturingssysteem kunnen worden geïmplementeerd omdat er informatie nodig is over de toekomst, die er in de meeste (of eigenlijk alle) gevallen niet is.

2. Het antwoord **moet** een berekening bevatten van het aantal page faults, niet alleen het eindantwoord. De uitkomsten zijn: FIFO: 13 faults, LRU: 12 faults, OPT: 10 faults

3. Hier zijn vele verschillende antwoorden mogelijk, met zowel een positieve als negatieve uitkomst. Een antwoord moet bestaan uit de volgende aspecten:
– Welke pages worden vastgezet en **waarom**.
– Voorbeeld uitwerking van een algoritme toegepast op de gegeven referentiestring met deze pages vastgezet.
– Conclusie **met onderbouwing**.

Opgave 4

1. De illustratie moet het volgende bevatten:
– Directory entry met pointer naar eerste blok.
– Non-FAT: blokken waarvan de laatste vier bytes steeds naar het volgende blok wijzen.
– FAT: losstaande blokken en een tabel waar de cellen steeds naar een volgende entry wijzen. De cellen van de tabel verwijzen steeds naar het corresponderende blok.

2. Non-FAT: 1 disk seek directory entry, 6 disk seeks voor elk blok, want de blokpointer is in de blokken opgeslagen. 7 totaal.

FAT: 1 disk seek directory entry, 1 disk seek voor FAT, 1 disk seek laatste blok om aantal bytes in dat blok te bepalen (OF een argument dat dit **niet** mogelijk/nodig is). 3 totaal.

3. Voorbeelden van mogelijke voorstellen:
– Dubbel gelinkte lijst gebruiken.
– File ID + bloknummer opslaan in elk blok.

Een pseudocode voor voorbeeld 2 kan bijvoorbeeld bestaan uit: chains kunnen voor elk bestand worden afgelopen, waarbij steeds het bloknummer wordt gecontroleerd. Mocht er een fout worden gevonden, dan kan de hele disk worden gescand om alle file IDs en bloknummers terug te vinden en de chain opnieuw op te bouwen.

4. *Manier 1: de blokken vergroten:*
– Pro: Het vergroten van blokken vermindert de druk op het blokallocatie algoritme. Grotere delen van bestanden komen bij elkaar te staan, minder diskarmbewegingen nodig, leidt tot performance winst.
– Con: Bij zeer grote blokken zal er interne fragmentatie optreden.

Manier 2: de blokpointers vergroten:
– Pro: Eenvoudige manier om meer blokken te kunnen adresseren.
– Con: Alle datastructuren (ook on-disk) moeten worden aangepast.
– Con: Het opslaan van grotere pointers kost meer ruimte. Pointers van 5 of 6 bytes zijn niet praktisch, dus er zal in de praktijk vaak worden gekozen voor 8 bytes.

Opgave 5

1. Swapping vindt plaats wanneer er een tekort is aan fysiek geheugen en er ruimte moet worden gemaakt voor andere processen. In dit geval wordt een proces (of een gedeelte van een proces in de vorm van pages) tijdelijk naar de disk geschreven. Het andere proces wordt daarna van de disk gelezen en in het geheugen geplaatst. Het proces dat naar de disk is geschreven wordt later weer in het geheugen gelezen.

2. Spinlocks worden gebruikt voor het beschermen van een kritieke sectie. Vaak is er sprake van een “lock” variabele welke aangeeft of er al een proces/thread in de kritieke sectie aanwezig is. Deze variabele wordt getest en gezet bij het betreden van de sectie en gereset bij het verlaten. De variabele wordt in de regel aangepast met behulp van atomaire instructies. Het testen van de variabele wordt gedaan door in een kleine loop telkens de variabelen uit te lezen en te vergelijken. Een proces/thread blijft deze loop uitvoeren, totdat de sectie kan worden betreden. Dit fenomeen wordt “busy waiting” of “spinning” genoemd en zodoende komt de spinlock aan de naam **spin**lock.

3. Het *mounten* van het bestandssysteem is nodig om het beschikbaar te maken binnen het draaiende besturingssysteem. Het komt eigenlijk overeen met het “openen” van bestand. Bij het *mounten* wordt het bestandssysteem geopend en wordt de directorystructuur opgeslagen in dit bestandssysteem als een subboom aan de algemene directoryboom gehangen.

4. Bij het starten van pipelines worden er meerdere processen opgestart, die zijn verbonden met pipes. In elk proces wordt een andere executable image geladen en opgestart. Threads binnen eenzelfde proces draaien allen dezelfde executable image, dus in dit geval kunnen er geen verschillende programma's worden opgestart, wat wel een vereiste is voor het opbouwen van een pipeline.