

## IN1805 Operating System concepten Oefen opgaven

### Opgave 1

Een OS is vaak ontworpen overeenkomstig een lagenstructuur.

- a. Beschrijf kort (in enkele zinnen) de essentie van een gelaagde opbouw. (Welk verband is er tussen de lagen?)
- b. Wat is het belangrijkste voordeel van een gelaagde structuur? Licht uw antwoord kort toe.
- c. Men kan in principe net zoveel lagen definiëren als men wil. Wat is de belangrijkste reden om het aantal lagen niet te klein te maken, en wat is de belangrijkste reden om het aantal lagen niet te groot te maken?  
Motiveer uw antwoord.

### Opgave 2

In een Operating System kan men 3 soorten schedulers onderscheiden, namelijk: long term, medium term en short term.

- a) Beschrijf kort ieder type. Geef daarbij steeds aan welk soort beslissingen betreffende scheduler neemt en op welke tijdschaal deze scheduler werkt.
- b) Beschrijf kort wat de functie is van een Process Control Block (PCB). Geef twee voorbeelden van informatie die in een PCB thuishoort.

### Opgave 3

Onderstaande vragen hebben betrekking op de toewijzing van procestijd aan processoren.

- a. Wat wordt verstaan onder de short-term scheduler?
- b. Wat wordt verstaan onder een preemptive scheduling policy?
- c. Bij de scheduling policy 'Shortest Job First' (SJF) wordt gekozen voor het 'kortste' proces. Beschrijf op welke wijze de processor tot de keuze van het kortste proces kan komen. (Met andere woorden: Hoe 'weet' de scheduler hoeveel tijd ieder proces nodig zal hebben?)
- d. Is SJF preemptive of nonpreemptive?
- e. Wat is het sterke punt van SJF? Licht uw antwoord kort toe.
- f. Is SJF een geschikte policy in geval van een time-sharing systeem?

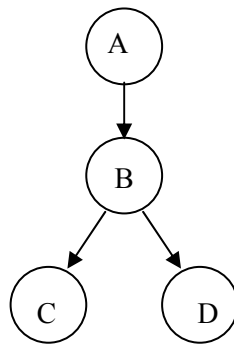
### Opgave 4

Gegeven een time-sharing systeem met een groot aantal ( $> 100$ ) terminals die allemaal hetzelfde type applicatie uitvoeren. De bijbehorende CPU-bursts zijn allemaal ongeveer even groot (er is een variatie van maximaal 20 % rond het gemiddelde). Men moet een scheduling policy kiezen voor de CPU scheduling. Men kiest verrassend voor een nonpreemptive scheduling policy, maar men is het nog niet eens over Shortest Job First (SJF) of First Come First Served (FCFS).

- a) Leg uit waarom de keuze voor een nonpreemptive policy hier verrassend is, en waarom dit hier wel een goede keus kan zijn.
- b) Wat is in deze situatie het belangrijkste voordeel van SJF? Motiveer uw antwoord.
- c) Wat is in deze situatie het belangrijkste voordeel van FCFS? Motiveer uw antwoord.
- d) Welk van deze methoden zou U in dit geval aanbevelen en waarom?

### Opgave 5

- De code die wordt uitgevoerd bij het wijzigen van de waarde van een semafoor door middel van Wait (of P) en Signal (of V), mag maar door één proces tegelijk doorlopen worden. (Met andere woorden: er moet wederzijdse uitsluiting gelden). Geef kort maar precies weer, hoe men binnen deze code wederzijdse uitsluiting kan bereiken. (Hierbij kan dus geen beroep worden gedaan op semaforen, want deze moeten juist geïmplementeerd worden.)
- In welke gevallen (voor welk doel?) is het zinvol een semafoor te initialiseren op 1, wanneer op 0 en wanneer op 2 of meer?  
Geef steeds een voorbeeld.
- De acties A, B, C en D zijn afhankelijk en kunnen alleen worden uitgevoerd in de volgorde zoals weergegeven in onderstaande synchronisatiegraaf. (d.w.z. B kan pas beginnen als A klaars is, C en D kunnen beginnen als B klaar is)



Geef aan hoe met behulp van semaforen de gewenste synchronisatie verkregen kan worden.  
Geef de semaforen namen en initiële waarden.

Geef aan waar in ieder proces op welke semaforen een Wait of Signal plaatsvindt.

### Opgave 6

Gegeven een systeem met N processoren ( $N > 1$ )

In het systeem zijn 2 processen actief die gebruik maken van hetzelfde programma. Het programma houdt door middel van een shared variable TELLER bij hoe vaak het is aangeroepen. Het programma is in assembler geschreven. Het ophogen van de TELLER met behulp van register R2 is als volgt gecodeerd.

```
.  
.  
load  R2,TELLER  { inhoud van TELLER naar R2 } [1]  
add   R2,1       { hoog R2 met 1 op          } [2]  
store R2,TELLER  { inhoud R2 naar TELLER   } [3]  
.
```

- Laat aan de hand van een tijdschema (een scenario), zien dat als de 2 processen concurrent worden uitgevoerd, het resultaat wel eens anders zou kunnen zijn dan de programmeur zich had voorgesteld.
- Als het systeem beschikt over een TestAndSet instructie, waar en hoe moet deze dan worden toegepast? Motiveer uw antwoord.
- Kan voor de oplossing van het gesignaleerde probleem ook gebruik worden gemaakt van DisableInterrupts in plaats van TestAndSet?  
motiveer uw antwoord.

### Opgave 7

Gegeven een systeem waarin een printer en een tape-unit aanwezig zijn.  
Op grond van te gebruiken resources zijn jobs in te delen in 3 klassen:

- klasse A: jobs die alleen de printer nodig hebben
- klasse B: jobs die alleen de tape-unit nodig hebben
- klasse C: jobs die de printer en de tape-unit nodig hebben.

Stel men heeft de volgende policy bedacht. Een job van klasse C mag alleen starten als bij de start van de job de printer en de tape-unit beide vrij zijn. Ze worden dan meteen aan deze job toegewezen. (Start en toewijzen vinden plaats door middel van een ondeelbare actie). Jobs van categorie A en B mogen altijd starten, dus zonder dat zij eerst de printer of de tape-unit toegewezen hebben gekregen.

- a. Wordt door deze policy deadlock voorkomen?  
Motiveer uw antwoord.
- b. Wat kunt U in de boven beschreven situatie zeggen over de kans op starvation?  
Motiveer uw antwoord.

### Opgave 8

Gegeven een systeem met 2 soorten tape-units: type A en type B. Tapes zijn niet uitwisselbaar tussen de verschillende units, dus moet bij iedere aanvraag het type (A of B) worden opgegeven.

Er zijn in totaal aanwezig 5 units van type A en 5 units van type B.

Het bankiersalgoritme wordt gebruikt om te beoordelen of een bepaalde aanvraag op een bepaald moment wel of niet gehonoreerd kan worden.

Op een gegeven tijdstip  $t_0$  zijn er 3 processen P1, P2 en P3 in het systeem aanwezig. De toewijzing van de tape-units is op dat moment als volgt:

proces	toegewezen		maximum (claim)	
	A	B	A	B
P1	0	1	3	4
P2	3	1	4	4
P3	1	2	1	3

gevraagd:

- a. Laat zien dat dit een veilige toestand is.
- b. Zou volgens het bankiersalgoritme op tijdstip  $t_0$  een vraag van P2 om een tape-unit van type A worden toegekend? Motiveer uw antwoord.
- c. Als volgens het bankiersalgoritme een bepaalde toekenning niet mag, en de toekenning zou toch gedaan worden, betekent dit dan dat er een deadlock optreedt? Licht uw antwoord toe.

### Opgave 9

Vergelijk 2 hypothetische systemen X2 en X4. De virtuele adresruimte in beide systemen is even groot. Het systeem X2 heeft een paginagrootte van 2 Kbytes, X4 heeft pagina's van 4 Kbytes.

Het reële geheugen is in beide systemen even groot. Ook de Translation-Look-Aside buffers zijn even groot. Als in beide systemen hetzelfde operating system wordt gebruikt (uiteraard zijn de delen die afhangen van de paginagrootte wel verschillend), en op beide systemen ook dezelfde processen draaien, wat kunt U dan onafhankelijk van de paginavervangings policy zeggen over de verschillen met betrekking tot de volgende grootheden?

Motiveer uw antwoord steeds kort.

- a. Omvang van de pagetables
- b. De omvang van een eventuele inverted pagetable
- c. Het aantal adresvertalingen per tijdseenheid
- d. De hitratio van de Translation-Look-Aside buffers
- e. interne geheugenverliezen
- f. externe geheugenverliezen

### Opgave 10

Gegeven een systeem met een fysiek geheugen van maximaal 16 Mbytes. De logische adresruimte bedraagt 1 Gbytes. (hierbij geldt:  $1\text{ G} = 2^{10}\text{ M}$ ,  $1\text{ M} = 2^{10}\text{ K}$ ,  $1\text{ K} = 2^{10}$ ). Er wordt gebruik gemaakt van paging.

De grootte van 1 page bedraagt 2048 bytes.

Een virtueel (logisch) adres ziet er als volgt uit

pagenr	offset
--------	--------

- a. Uit hoeveel bits bestaat een virtueel adres?  
Hoeveel bits zijn er nodig voor het pagenr?  
Hoeveel bits zijn er nodig voor de offset?  
Geef een korte toelichting.
- b. Wat is in dit systeem de minimale omvang van een entry in een pagetable? (Dit moet een geheel aantal bytes zijn)  
Geef een korte motivering.
- c. Stel een proces P gebruikt alleen adressen helemaal onder in het virtuele geheugen (dat wil zeggen in pagina 0) of helemaal boven in het virtuele geheugen (dat wil zeggen in de pagina met het hoogste pagina nummer).  
Hoe groot zou dan de pagetable worden bij de meest eenvoudige organisatie?  
Geef de berekening.
- d. Beschrijf een manier voor het organiseren van paging en pagetables, zodat de pagetable in een situatie zoals beschreven onder c. minder ruimte in het reële geheugen inneemt.  
Bereken bij de gekozen oplossing hoeveel ruimte er in de genoemde situatie voor pagetable van proces P in main storage nodig is.  
Laat de berekening zien.

### Opgave 11

Er bestaan verschillende methoden voor toekenning van schijfruimte aan files.

- a. Beschrijf kort de methoden contiguous (aaneengesloten) en indexed (geïndexeerd).
- b. Vergelijk deze methoden op de volgende aspecten:
  - 1) toegangsmogelijkheden (sequentieel en direct access)
  - 2) benodigde aantal seeks
  - 3) ruimteverlies
  - 4) gebruiksgemak

### Opgave 12

- a. De tijd die verloopt tussen het starten van een opdracht om een blok van schijf te lezen tot het moment waarop de leesopdracht klaar is bestaat uit 3 delen. Geef een korte omschrijving van deze 3 componenten en een typische grootte van elk van deze componenten.
- b. Wanneer er in de queue voor een bepaald device een aantal I/O-requests staan te wachten, moet de device driver bepalen in welke volgorde deze requests zullen worden afgehandeld. Enige policies hiervoor zijn:
- First Come, First Served (FCFS)
  - Shortest Seek Time First (SSTF)
  - SCAN
- Beschrijf elk van deze 3 methoden kort, en laat aan de hand van de volgende request-queue zien, wat de totale seektijd voor elk van deze policies is. (Seektijd uit te drukken in aantal over te steken tracks.) Neem aan dat de cylinders genummerd zijn van 0 tot en met 199, en dat bij de start de leeskop zich bevindt op cylinder 100.
- De requestqueue bevat de requests voor de volgende cylinders (in volgorde van binnenkomst; het oudste request links): 10, 140, 45, 180, 95
- c. Tot nu toe is in deze opgave alleen rekening gehouden met een niet veranderende request queue. In een normaal systeem kunnen er tijdens uitvoering van een I/O-opdracht nieuwe requests aan de queue worden toegevoegd. Welk van de de bovengenoemde methoden kunnen dan aanleiding geven tot starvation en welke niet? Motiveer uw antwoord kort.
- d. SCAN kent ook een circulaire variant. Leg kort uit wat dit inhoudt en in welk opzicht dit een verbetering is t.o.v. SCAN.

## IN1805 Operating System Concepts

### Uitwerking oefen opgaven

#### Uitwerking opgave 1

- a. Gelaagde opbouw betekent dat de functionaliteit van een systeem laag voor laag van beneden naar boven wordt opgebouwd. Iedere laag voegt enige functionaliteit toe aan het geheel van de onderliggende lagen. Een laag kan alleen gebruik maken van functies in de onderliggende lagen. In het ideale geval ziet laag N alleen het interface naar laag N-1
- b. Door deze structuur kan laag voor laag worden gebouwd en getest. Eventuele fouten zijn sneller te localiseren.
- c. De overhead neemt toe als het aantal lagen toeneemt, daarom het aantal lagen niet te groot maken. Bij een te klein aantal lagen is er teveel functionaliteit per laag, deze wordt daardoor complexer en meer foutgevoelig.

#### Uitwerking opgave 2

- De long-term scheduler houdt zich bezig met het kiezen van batchjobs die in executie genomen gaan worden. Deze werkt op een tijdschaal van minuten tot vele uren.
- De medium-term scheduler houdt zich bezig met de vraag welke processen tijdelijk uit het geheugen verwijderd moeten worden (door middel van een swap out) als het Multiprogrammeringslevel te hoog is geworden. Ook het besluit een dergelijk proces weer naar binnen te halen (swap in) hoort bij de medium term scheduler. Werkt op een tijdschaal van minuten.
- De short-term scheduler kiest uit de ready queue een proces dat de CPU krijgt. De short-term scheduler komt in de regel vele malen per seconde in actie (na iedere interrupt)
- b. Het Proces Control Block (PCB) is voor het OS de representatie van een proces. Het bevat alle informatie die het OS nodig heeft om het proces te kunnen managen.  
Voorbeelden van informatie in het PCB zijn:  
processtatus,  
inhoud van program counter en andere registers bij de laatste interrupt,  
accounting en scheduling informatie (b.v. prioriteit).

#### Uitwerking opgave 3

- a. De short-term scheduler selecteert een proces uit de ready-queue om daaraan de processor toe te kennen.
- b. Een preemptive scheduling policy maakt het mogelijk de processor van een proces af te nemen en toe te kennen aan een ander proces, terwijl het eerste proces de processor nog kon gebruiken.
- c. De scheduler kan niet weten hoeveel tijd ieder proces nodig zal hebben, maar voor ieder proces kan een schatting worden gemaakt van de tijd die het in de komende CPU-burst zal vragen. Hierbij wordt gebruik gemaakt van een exponentieel gemiddelde, waarbij de duur van de vorige burst en het verdere verleden in een bepaalde gewichtsverhouding worden meegenomen, in formulevorm:  
 $T_{n+1} = \alpha t_n + (1-\alpha)T_n$ , waarbij  $T_i$  de schatting is voor de  $i$ 'de CPU-burst,  $t_j$  de gemeten lengte van de  $j$ 'de CPU-burst, en  $\alpha$  de gewichtsverhouding.
- d. Er bestaat zowel een preemptive als een non-preemptive SJF.
- e. preemptive SJF geeft de **kortste gemiddelde** wachttijd (d.w.z. 'waiting while ready').  
Toelichting: een kort proces verplaatsen van ná een lang proces naar vóór een lang proces, vermindert de wachttijd van het korte proces meer, dan dat het de wachttijd van het lange proces doet toenemen.
- f. nonpreemptive SJF is zeker niet geschikt voor een time-sharing systeem, want voor een time-sharing systeem is preemption vereist. Preemptive SJF is wel bruikbaar, maar heeft het gevaar van starvation voor CPU-gebonden processen.

#### Uitwerking opgave 4

- a. Een nonpreemptive policy is hier verrassend omdat een timesharing systeem in de regel programmerende gebruikers kent die allemaal verschillende activiteiten uitvoeren. zij kunnen b.v. programma's draaien die zeer grote CPU-bursts bevatten (b.v. een oneindige lus als gevolg van een programmeerfout.) Een preemptive scheduling policy is dan nodig omdat andere gebruikers anders een slechte responstijd krijgen. In de situatie zoals beschreven in de opgave kan een non-preemptive scheduling policy werken, omdat de alle CPU bursts ongeveer even groot zijn.
- b. Het belangrijkste voordeel van SJF is dat het gemiddeld de kortste responstijd geeft.
- c. Het belangrijkste voordeel van FCFS is dat het eenvoudig te implementeren is: namelijk een queue waarvan steeds de eerste wordt geselecteerd, en waar nieuwe processen achteraan sluiten.
- d. FCFS is aan te bevelen vanwege de eenvoudige implementatie. Het belangrijkste nadeel van FCFS: het relatief lang moeten wachten vvan kleine processen is hier minder belangrijk omdat de CPU bursts

Uitwerking opgave 5

- a. Gebruik shared variabele Sembusy initieel (FALSE)  
De de semafooroperaties (dus Wait en Signal) beginnen met:  
    While TestAndSet(Sembusy) do no-op;  
  
en eindigen met:  
    Sembusy=FALSE;
- b. Semaforen initialiseren op:  
    0 ten behoeve van synchronisatie,  
    b.v. proces B moet wachten tot proces A klaar is,  
    proces A eindigt met Signal(SemA)  
    Proces B begint met Wait(SemA)  
  
    1 ten behoeve van wederzijdse uitsluiting  
    kritieke sectie begint met:  
    Wait(mutex)  
    kritieke sectie eindigt met Signal(mutex)  
  
    2 of meer als de semafoor wordt gebruikt als teller, b.v. als er van een resource 2 instances zijn.  
    Voorbeeld er zijn 2 buffers. Semafoor FreeBufCnt wordt geïnitialiseerd op 2.  
    Bij in gebruiknemen van buffer Wait(FreebufCnt).  
    Bij vrijgeven van buffer Signal(FreeBufCnt)
- c. Semafoor Aklaar initieel 0, Semafoor Bklaar initieel 0  
    A eindigen met Signal(Aklaar)  
    B beginnen met Wait(Aklaar)  
    B eindigen met Signal(Bklaar), Signal(Bklaar)  
    C en D ieder beginnen met Wait(Bklaar)

### Uitwerking opgave 6

- a. Mogelijk scenario:  
(tussen de haakjes steeds de nieuwe waarde van R2)

Neem aan shared variabele TELLER heeft waarde n

A		B	
load R2,TELLER	(n)	load R2,TELLER	(n)
add R2,1	(n+1)	add R2,1	(n+1)
		store R2,TELLER	(n+1)
store R2,TELLER	(n+1)		

Resultaat is dus dat de TELLER na afloop van beide acties maar met 1 is opgehoogd, in plaats van de verwachte 2.

- b. TestAndSet zou als volgt moeten worden toegepast.  
Definieer een shared boolean variabele LockTeller, initiële waarde FALSE.

Vóór de executie van [1] zou de Entry code moeten worden uitgevoerd die als volgt kan worden beschreven.

```
While TestAndSet(LockTeller) do no-op; end;
```

Na afloop van [3] zou de Exit code moeten worden toegepast, die als volgt kan worden beschreven.

```
LockTeller := FALSE;
```

Toelichting:

LockTeller is FALSE betekent dat de kritieke sectie niet op slot is. Het eerste proces dat hem niet op slot aantreft mag erin, maar moet hem wel meteen op slot zetten.

TestAndSet zet in een ondeelbare actie LockTeller op TRUE en geeft de oude waarde terug. Als een proces ziet dat de oude waarde FALSE was, 'weet' het proces zeker dat hij degene was die de sectie niet op slot aantrof en hem op slot heeft gezet. Hij mag dus de sectie binnengaan. Als een proces ziet dat de oude waarde TRUE is, 'weet' het proces dat hij in feite niets veranderd heeft en dat hij er dus niet in mag.

- c. DisableInterrupts schakelt alle interrupts van een processor uit. Dit lost het probleem in dit geval niet op want als een ander proces gebruik maakt van een andere processor kan nog steeds het onder a. gesignaleerde probleem optreden. DisableInterrupts zou in geval van een systeem met maar 1 processor wel een (zij het dure) oplossing zijn.

### Uitwerking opgave 7

- a. Deadlock wordt op deze wijze voorkomen.  
Dit is te zien aan de hand van een resource allocation diagram. Als er geen circulair pad kan voorkomen, kan er geen deadlock optreden.

Een proces kan alleen onderdeel zijn van een circulair pad wanneer het een resource heeft en tegelijkertijd wacht op een ander resource. Deze situatie kan hier niet optreden omdat processen die meer dan één resource nodig hebben niet kunnen starten als deze niet beschikbaar zijn.

- b. De kans op starvation hangt af van de toewijzingspolicy. Als deze liberaal is, en de printer of de tape steeds onmiddellijk toewijst aan klasse A of B jobs, zal een klasse C job kunnen lijden aan starvation. Hij zal misschien nooit kunnen starten omdat er altijd wel of een A job of een B job aanwezig is die de printer of de tape nodig heeft.

Ook het omgekeerde is mogelijk (maar minder waarschijnlijk) als de policy heel voorzichtig is en steeds als er een C job wil beginnen, wacht met het toekennen van de tape en de printer aan B of A jobs.

### Uitwerking opgave 8

- a.      Aanwezig aantal units      A: 5,      B: 5  
         op t0 toegewezen          A: 4,      B: 4  
         op t0 nog beschikbaar      A: 1,      B: 1

Voor toegewezen (alloc) en nog nodig (need) en claim (max) geldt:

	alloc		need		max
	A	B	A	B	A B
P1	0	1	3	3	3 4
P2	3	1	1	3	4 4
p3	1	2	0	1	1 3

(note: alloc + need = max )

safety algoritme:

work := available = (1,1)

**ronde 1:**

need<sub>3</sub> ≤ work want (0,1) ≤ (1,1)

work := work + alloc<sub>3</sub> = (1,1) + (1,2) = (2,3)

**ronde 2:**

need<sub>2</sub> ≤ work want (1,1) ≤ (2,3)

work := work + alloc<sub>2</sub> = (2,3) + (3,1) = (5,4)

**ronde 3:**

need<sub>1</sub> ≤ work want (3,3) ≤ (5,4)

work := work + alloc<sub>1</sub> = (5,1) + (0,1) = (5,5)

(note: work komt nu overeen met alle aanwezige units)

Hiermee zijn alle processen geëindigd. Dus de toestand op t0 is veilig.

- b. Als P<sub>2</sub> om een tape zou vragen en die zou krijgen, dan zouden alloc en need er als volgt uitzien:

available (0,1)

	alloc		need	
	A	B	A	B
P <sub>1</sub>	0	1	3	3
P <sub>2</sub>	4	1	0	3
P <sub>3</sub>	1	2	0	1

Het safety algoritme, zou geven:

work := available := (0,1)

**ronde 1:**

need<sub>3</sub> ≤ work

work := work + alloc<sub>3</sub> = (1,3)

**ronde 2:**

need<sub>2</sub> ≤ work

work := work + alloc<sub>2</sub> = (5,4)

**ronde 3:**

need<sub>1</sub> ≤ work

work := work + alloc<sub>1</sub> = (5,5)

Hiermee alle processen geëindigd, dus dit is toegestaan.

- c. Als een bepaalde toewijzing niet is toegestaan maar toch plaats zou vinden, betekent dit alleen dat de toestand niet meer veilig is: d.w.z. er **kan** een deadlock ontstaan, maar dat is **niet zeker**.

### Uitwerking opgave 9

- a. De pagetable bij X2 is tweemaal zo groot als pagetable bij X4, want er is een entry per page en X2 heeft tweemaal zoveel pages als X4
- b. Een inverted pagetable zal bij X2 ook tweemaal zo groot zijn als bij X4, want er zijn tweemaal zoveel frames in hetzelfde fysieke geheugen als bij X4.

- c. Het aantal adresvertalingen per eenheid van tijd bij beide systemen is even groot. Dit aantal wordt bepaald door het uitgevoerde programma en is niet afhankelijk van de paginagrootte.
- d. De hitratio van de TLB zal bij X4 hoger zijn. Want, stel er kunnen N entries in de TLB, dan zullen deze N entries in het geval van X4 een groter deel van het logische geheugen overdekken dan in het geval van X2.
- e. De interne fragmentatie bij X4 is groter dan die bij X2, omdat er gemiddeld per programma een halve page verloren gaat.
- f. De externe fragmentatie is bij paging altijd 0.

#### Uitwerking opgave 10

- a. de virtuele adresruimte is 1 Gbyte, d.i.  $2^{30}$  bytes.  
Dit betekent dat er 30 bits nodig zijn om een virtueel adres aan te geven.  
Een page is 2048 bytes, d.i.  $2^{11}$ , dus er zijn 11 bits nodig voor de offset.  
Voor het aangeven van het pagenummer zijn er dus  $30-11=19$  bits
- b. Een entry van de pagetable moet bevatten: een framenummer en enkele statusbits.  
Het aantal mogelijke frames bedraagt: maximaal fysieke geheugen / pagesize =  $16 \times 2^{20} / 2^{11} = 2^{24} / 2^{11} = 2^{13}$ ,  
Er zijn dus 13 bits nodig voor het framenummer  
Het dichtstbijzijnde veelvoud van 8 is 16. Bij 3 statusbits is dus de minimale omvang van een entry in de pagetable dus 16 bits, ofwel 2 bytes.
- c. Bij de meest eenvoudige organisatie zou de pagetable een omvang krijgen van : aantal mogelijke pages x entrysize =  $2^{19} \times 2 = 2^{20} = 1$  Mbyte.
- d. Het paging mechanisme zou gebruik kunnen maken van paging op 2 niveaus. De pagenummers in dit voorbeeld zouden gesplitst kunnen worden in 10 bits die de plaats weergeven van een entry in een superpagetable, en 9 bits die de plaats aangeven in de door de superpagetable aangewezen pagetable.  
De superpagetable bevat  $2^{10} = 1024$  entries  
Een gewone pagetable bevat  $2^9 = 512$  entries  
In het in c. beschreven voorbeeld, zou nodig zijn 1 superpagetable van 1024 entries en 2 gewone pagetables van 512 entries. Per entry zou 2 bytes nodig zijn, dit betekent in totaal:  $1024 * 2 + 512 * 2 + 512 * 2 = 4096$  bytes.

#### Uitwerking opgave 11

- a. Bij contiguous allocation worden de blokken die horen bij een file, aaneengesloten (en in de juiste volgorde) opgeslagen op de schijf.  
Bij indexed allocation worden de blokken die horen bij een file, verspreid over de schijf opgeslagen. De plaats waar een blok zich bevindt, wordt bijgehouden in een indextabel. Via het bloknummer kan men in de indextabel de locatie van het blok op de schijf vinden.
- b. 1) Zowel bij contiguous allocation als bij indexed allocation zijn beide access methoden mogelijk  
2) Voor sequentieel access kunne in geval van indexd allocation meer seeks nodig zijn. De blokken hoven bij indexed allocation immers niet naast elkaar te liggen.  
Voor direct access zijn in beide gevallen evenveel seeks nodig, namelijk 1 per blok. Dit geldt onder voorwaarde dat het index blok (de index tabel) in het primaire ggeheugen is ingelezen.  
3) Bij contiguous allocation kan zowel interne als externe ffragmentatie optreden. Interne fragmentatie treedt op wanneer men meer ruimte heeft gereserveerd dan nodig is voor de file. Externe fragmentatie is er als gevolg van het dynamisch creeren en verwijderen van files vvan verschillende grootte, waardoor verspreid over de schijf vrije ruimten van verschillende grootten ontstaan.  
Bij indexed allocation is er ggeen interne of externe fragmentatie, er is el extra schijruimte nodig voor index blokken.
- c. Indexed allocation is gemakkelijker voor de gebruiker. De gebruiker hoeft niet van te voren te weten hoe groot zijn file zal worden; files zijn voortdurend uit te breiden; er is niet een regelmatige compaction actie nodig.

### Uitwerking opgave 12

- a. De tijd tussen het starten van een I/O opdracht en het beëindigen: is op te delen in:  
**seek tijd:** dit is de tijd die het kost om de lees/schrijfkop naar de juiste cylinder te bewegen, b.v. 10 ms.  
**latency tijd** (=rotational delay): de wachttijd voordat de goede sector zich onder de lees/schrijfkop bevindt, b.v. 10 ms  
**transfertijd:** dit is de tijd die het kost om de data daadwerkelijk te lezen en te transporteren naar de controller, b.v. 1 ms
- b. First Come First Served (FCFS)  
I/O-requests staan in een queue. Nieuwe requests worden achteraan gezet. Wanneer weer een request verwerkt kan worden, wordt hiervoor de eerste in de rij genomen.  
Volgorde van afhandeling van het voorbeeld (kop bij de start op 100): 100 10 140 45 180 95, over te steken dus  $90+130+95+135+85 = 535$  cylinders
- Shortest Seek Time First (SSTF)  
Er wordt steeds naar de inhoud van de hele queue gekeken. De request die het dichtstbij de huidige positie van de lees/schrijfkop ligt, wordt gekozen. Volgorde van afhandeling:  
100 95 140 180 45 10, over te steken cylinders dus:  
 $5 + 45 + 40 + 135 + 35 = 260$
- Scan  
De kop beweegt steeds over het volledige traject van binnen naar buiten en omgekeerd. Als de kop een cylinder passeert waarvoor een request uitstaat, wordt deze request uitgevoerd.  
Volgorde van afhandeling (aanneem kop bij start op 100, op weg van laag naar hoog, cylinders genummerd van 0 t/m 199)  
100 140 180 (199) 95 45 10, over te steken cylinders dus:  $40 + 40 + 19 + 104 + 50 + 35 = 288$
- c. Van de gegeven methodes kan alleen SSTF aanleiding geven tot starvation. Als er steeds nieuwe verzoeken binnenkomen voor cylinders in de buurt van de plek waar de lees/schrijfkop zich bevindt, kunnen requests voor verderweg gelegen cylinders onbepaald lang moeten wachten .  
Bij FCFS is het zeker dat een request aan de beurt komt als het (eindige) aantal requests voor hem is afgewerkt.  
Bij Scan weet je zeker dat iedere cylinder op zijn beurt wordt afgehandeld, dus iedere request komt binnen een bepaalde tijd aan bod.
- d. De circulaire variant van Scan houdt in dat I/O-requests alleen worden uitgevoerd terwijl de kop in bepaalde richting beweegt (b.v. van laag naar hoog), en niet op de terugweg.  
Het voordeel hieraan is een gelijkmatiger verdeling van de wachttijden. Bij de niet-circulaire variant wordt een cylinder dicht bij een keerpunt steeds snel na elkaar tweemaal behandeld en daarna heel lang niet, dus een grotere variatie in wachttijden