**CH 5….**

1. **First come, first served (FCFS):**

فكرته

المهمة التي تأتي أولا تنفذ أولا وبشكل كامل أي أنها تبقى في المعالج حتى تنتهي

عيوبه

متوسط وقت الانتظار يعتمد بشكل كبير على ترتيب وصول المهام وكذلك أن مهمة قصيرة قد تنتظر مهمة طويلة في أنظمة الـ (time sharing)

مميزاته

بسيط

* The code for FCFS scheduling is simple to write and understand.
* On the negative side, the average waiting time under the FCFS policy is

often quite long.

* There is a **convoy effect** as all the other processes wait for the one big process to get off the CPU.
* Note also that the FCFS scheduling algorithm is nonpreemptive. Once the

CPU has been allocated to a process, that process keeps the CPU until it releases

the CPU, either by terminating or by requesting I/O.

**E.G.**

Using the FCFS algorithm answer the following question using the table below.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst time |
| P2 | 0.0 | 12 |
| P3 | 3.0 | 8 |
| P4 | 5.0 | 4 |
| P1 | 10.0 | 10 |
| P5 | 12.0 | 6 |

* Develop Gantt showing the execution of the processes in the order mentioned.
* Waiting time for each process and average waiting time.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P3 | P4 | P1 | P5 |

0 12 20 24 34 40

P1: 24 ms.

P2: 0 ms.

P3: 12 ms.

P4: 20 ms.

P5: 34 ms.

Total= 90 ms. Avg. = 90/5=18 ms.

**2-Shortest-Job-First (SJF) Scheduling:**

**فكرته**

أن المهمة التى لاتستغرق وقت طويل في المعالج تدخل اولا وتنقسم هذه الطريقه الى نوعان:

النوع الاول :

( nonpreemptive)

عند قدوم عدد من المهام الى المعالج ياخذ الـمهمة التي لاتاخذ وقت طويل ولكن عند قدوم مهمة الوقت اللازم لها اقل من التي مع المعالج فانه يتم تجاهلها ويكمل المعالج عمله الى ان ينتهي

النوع الثاني :

(preemptive)

نفس الطريقه السابقه ولكن الفرق هو عند قدوم مهمة الوقت اللازم لها اقل من التي مع المعالج فان المعالج يوقف العمليه ويضع المهمة التي معهافي ([ready queue](http://os2a.pbwiki.com/ready+queue))وياخذ المهمة القادمه وهكذا

عيوبه:

صعوبة برمجته وذلك لعدم معرفة الوقت اللازم لبقاء المهمة الجديدة في المعالج.

مميزاته

 مثالي

* The SJF scheduling algorithm is provably optimal, in that it gives the

minimum average waiting time for a given set of processes.

The real difficulty with the SJF algorithm is knowing the length of the next

CPU request.

SJF scheduling is used frequently in long-term scheduling

* Although the SJF algorithm is optimal, it cannot be implemented at the

level of short-term CPU scheduling. With short-term scheduling, there is no

way to know the length of the next CPU burst. One approach to this problem

is to try to approximate SJF scheduling. We may not know the length of the

next CPU burst, but we may be able to predict its value.

The SJF algorithm can be either preemptive or nonpreemptive.

* A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst.
* Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first**

scheduling.

**E.G.**

Using the Non-preemptive SJF algorithm answer the following question using the table below.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst time |
| P2 | 0.0 | 12 |
| P3 | 3.0 | 8 |
| P4 | 5.0 | 4 |
| P1 | 10.0 | 10 |
| P5 | 12.0 | 6 |

* Develop Gantt showing the execution of the processes in the order mentioned.
* Waiting time for each process and average waiting time.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P2 | P4 | P5 | P3 | P1 |

0 12 16 22 30 40

P1: 30-10=20 ms.

P2: 0 ms.

P3: 22-3=19 ms.

P4: 12-5=7 ms.

P5: 16-12=4ms.

Total= 50 ms. Avg. = 50/5=10 ms.

**E.G.**

Using the preemptive SJF algorithm answer the following question using the table below.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst time |
| P2 | 0.0 | 12 |
| P3 | 3.0 | 8 |
| P4 | 5.0 | 4 |
| P1 | 10.0 | 10 |
| P5 | 12.0 | 6 |

* Develop Gantt showing the execution of the processes in the order mentioned.
* Waiting time for each process and average waiting time.

Answer:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P2 | P3 | P4 | P3 | P5 | P2 | P1 |

0 3 5 9 15 21 30 40

P1: 30-10=20 ms.

P2: 0 ms.

21 – 3 =18 ms.

18 + 0 = 18 ms.

P3: 3 – 3 =0 ms.

9 – 5 =4 ms.

4 + 0 = 4 ms.

P4: 5 -5 = 0 ms.

P5: 15 – 12 = 3 ms.

Total= 45 ms. Avg. = 45/5= 9 ms.

**3-Priority Scheduling**

فكرته

يعتمد على (A priority number) وهو عباره عن رقم صحيح ياتي مع الـمهمة يمثل الاولوية أي ان المهمة التي معها عدد صغير له اولويه اكبر(في نظام يونكس مثلا) عن مهمة التي معها رقم كبير  أي أن لها الحق في الدخول للمعالج

عيوبه

التجويع (Starvation)هو ان الأقل أولوية قد لاتنفذ ابدا لذلك الحل هو تقليل الرقم الذي يمثل الـمهمة من فتره الى اخرى الى ان يصبح ذو اولويه عليا فينفذ (هذا بالنسبه للمهمة التي ليست لها اولويه عليا وذلك حتى لاتظل بدون تنفيذ)

* A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low priority processes waiting indefinitely.
* Absolution to the problem of indefinite blockage of low-priority processes is **aging**. Aging involves gradually increasing the priority of processes that wait in the system for a long time.

**E.G.**

Using the Priority algorithm answer the following question using the table below.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst time |
| P2 | 0.0 | 12 |
| P3 | 3.0 | 8 |
| P4 | 5.0 | 4 |
| P1 | 10.0 | 10 |
| P5 | 12.0 | 6 |

* Develop Gantt showing the execution of the processes in the order mentioned.
* Waiting time for each process and average waiting time.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P4 | P2 | P5 | P1 | P3 |

0 5 16 22 32 40

P1: 22 ms.

P2: 4ms.

P3: 32 ms.

P4: 0 ms.

P5: 16 ms.

Total= 74 ms. Avg. = 74/5= 14.8 ms.

**4-Round Robin:**

فكرته:

الطريقه هذه لاتنظر لأهمية الـمهمة جميعهم متساوون .  يحدد وقت للمعالج ولنقل 20 عادة يكون 10 - 100  milliseconds

ومعنى ذلك ان المهمة سواء كانت لها أولوية او لا تاخذ 20 من الزمن فقط وتخرج من المعالج حتى ولو لم تنتهي إذا انتهت لا توجد مشكلة اما اذا لم تنتهي فانها تخرج وتوضع في اخر الطابور

* The **round-robin (RR)** scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to

enable the system to switch between processes. A small unit of time, called a

**time quantum** or **time slice**, is defined. A time quantum is generally from 10

to 100 milliseconds in length.

The average waiting time under the RR policy is often long.

* The performance of the RR algorithm depends heavily on the size of the time quantum. At one extreme, if the time quantum is extremely large, the RR policy is the same as the FCFS policy. In contrast, if the time quantum is extremely small (say, 1 millisecond), the RR approach can result in a large number of context switches.

**E.G.**

Using the RR algorithm answer the following question using the table below.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst time |
| P2 | 0.0 | 12 |
| P3 | 3.0 | 8 |
| P4 | 5.0 | 4 |
| P1 | 10.0 | 10 |
| P5 | 12.0 | 6 |

* Develop Gantt showing the execution of the processes in the order mentioned.
* Waiting time for each process and average waiting time.

Answer:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P1 | P2 | P4 | P1 |

0 5 10 14 19 24 29 32 37 40

P1: 0 + (24 - 5)+ (37 - 29)= 0 + 19 +8 = 27 ms.

P2: 5 + (29-10)= 5+19= 24 ms.

P3: 10 ms.

P4: 14 + (32 - 19)= 14 + 13= 27 ms.

P5: 19 ms.

Total= 107 ms. Avg. = 107/5= 21.4 ms.

**5-shortest-job-first Scheduling**

**انه من الممكن تحليل اداء احد الخوارزميات لمجموعة من العمليات اذا توفرت المعلومات المناسبة المتعلقة بالعمليات .**

**فمثلا لتقييم خوارزمي (SJF)ينبغي الحصول على البيانات المتعلقه بزمن وصول العملية وزمن تنفيذها.**

**لكي يكون اوضح ,لناخذ مجموعة العمليات الاربع المبينة في الجدول :**

|  |  |  |
| --- | --- | --- |
| **زمن المعالجة** | **زمن الوصول** | **العملية** |
| **A** | **0.0** | **5** |
| **B** | **1.0** | **2** |
| **C** | **2.0** | **5** |
| **D** | **3.0** | **3** |

* **عند الزمن 0, العملية التي دخلت الى النظام هي A فقط وبالتالي فهي التي تنفذ.**
* **عند الزمن 1,تصل العملية B.عندها يتبقى للعملية A اربع وحدات زمنية للتنفيذ ولB وحدتان لذلك تبدا العملية B بالتنفيذ عند الزمن 1 .**
* **عند الزمن 2, تدخل العملية C النظام .تستمر B في التنفيذ لان لها ادنى عدد من الوحدات الزمنية من بين العمليات الثلاث .**
* **عند الزمن 3, تنتهي B وتدخل العملية D الى النظام.ومن بين العمليات A و C و D, تملك D زمن التنفيذ المتبقي الاقصر وبالتالي فهي تبدا بالتنفيذ.**
* **عندما تنتهي العملية D عند الزمن 6, يتم اكمال تنفيذ العملية A.**
* **وعندما تنتهي العملية A عند الزمن 10,تنفذ العمليه C.**

ويوضح المخطط التالي توقيت تنفيذ العمليات:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | A | D | B | A |

15106 3 1

**ويمكن حساب الزمن الدوري لكل عملية بطرح زمن دخول العملية الى النظام من زمن انتهائها** .

**والان نحسب ازمنة الدورة للعمليات السابقة:**

**العملية A: 10-0=10**

**العملية B: 3-1=2**

**العملية C: 15-2=13**

**العملية D: 6-3=3**

**اذا الزمن الدوري المتوسط يساوي (10+2+13+3)/4 أي 7**

**ويمكن حساب زمن الانتظار بطرح زمن التنفيذ من زمن الدورة, وبالتالي :**

**العملية A: 10-5=5**

**العملية B: 2-2=0**

**العملية C: 13-5=8**

**العملية D: 3-3=0**

**اذا متوسط زمن الانتظار هو (5+0+8+0)/4 أي 3.25**

**تم تنفيذ العمليات الاربع في غضون 15 وحدة زمنية. وبالتالي الانتاجية تساوي 15/4 وحدة زمنية لكل وظيفة.**

**CH 7…**

**Resource-allocation graph of Deadlock**

الجمود يمكن ان يوصف بدقة اكثر بواسطة الرسم البياني( (graph ويسمى (system resource-allocation graph) . ويتكون من مجموعة من القمم (vertices) ومجموعة من الحواف(edge) .

مجموعة القمم تقسم إلى صنفين مختلفين :

1-p={p1,p2,…..,pn} وهذه المجموعة تتكون من كل العمليات الفعالة في النظام.

2-R={R1,R2,….,Rn} وهذه المجموعة تتكون من كل أنواع الموارد في النظام.

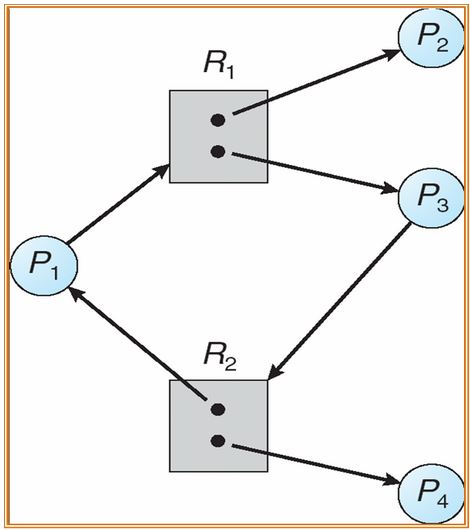
والإنتقال المباشر من العملية Pi إلى نوع الموردRj يكتب (*(Pi*® *Rj*.وتعني أنPi طلبت طلب منRj وهي حاليا تنتظر من المورد , وتسمى طلب حافة (request edge) .

والإنتقال المباشر من المورد Rj إلى العملية Pi يكتب(*(Rj* ® *Pi*وتعني أن طلبRj قد خصص لـ . Pi

وتسمى تخصيص الحافة (assignment edge).

تمثل العملية بـ دائرة , ونوع المورد بـ مستطيل .

نوع المورد ممكن ان يملك اكثر من طلب ويمثل الطلب بـ نقطة داخل المستطيل .



من هذه الصورة نعرف ان :

\*مجموعة P(process), R(resource type), E(edge):

-         p={p1,p2,p3}

-         R={R1,R2,R3,R4}

-         E={p1®R1, p2®R3, R1®p2, R2®p2,R2®p1, R3®p3}

 \*طلب المورد:

-     طلب 1 في R1

-     طلب 2 في R2

-     طلب 1 في R3

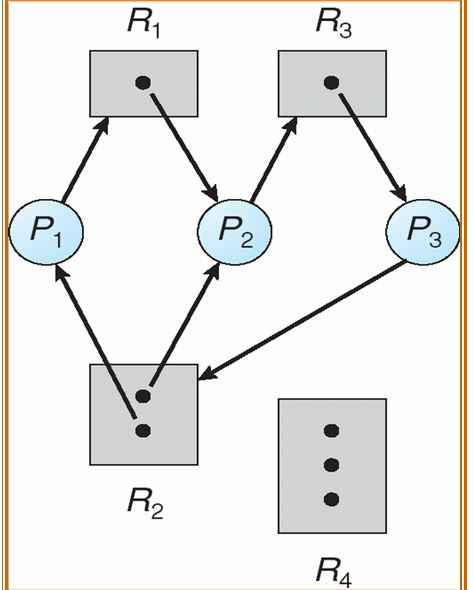
-     طلب 3 فيR4

\*لو الرسم لا يحتوي على دائره ---- لايوجد جمود

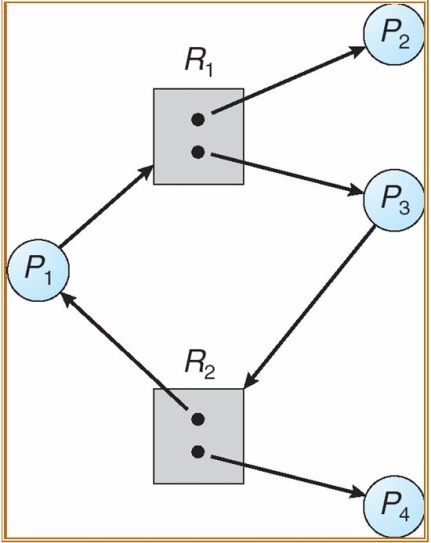
\*لو الرسم يحتوي على دائرة :

    - إذا وجد طلب واحد في كل مورد ---يوجد جمود

    -إذا وجد أكثر من طلب في كل مورد---لا يوجد جمود



يوجد جمود.



لايوجد جمود

* Deadlocks can be described more precisely in terms of a directed graph called a **system resource-allocation graph**.
* If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

Deadlock Prevention

* Mutual Exclusion

at least one resource must be non-sharable. Sharable resources, in contrast, do not require mutually exclusive

access and thus cannot be involved in a deadlock. Read-only files are a good

example of a sharable resource.

* Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must

guarantee that, whenever a process requests a resource, it does not hold any other resources.

* Both these protocols have two main disadvantages. First, resource utilization

may be low, since resources may be allocated but unused for a long period.

* Second, starvation is possible. A process that needs several popular

resources may have to wait indefinitely, because at least one of the resources

that it needs is always allocated to some other process.

* No Preemption

To ensure that this condition does not hold, we can use the following protocol. If a process is holding

some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is

currently holding are preempted.

This protocol is often applied to resources whose state can be easily saved

and restored later, such as CPU registers and memory space. It cannot generally

be applied to such resources as mutex locks and semaphores.

* Circular Wait

One way to ensure that this condition never holds is to impose a total ordering of

all resource types and to require that each process requests resources in an

increasing order of enumeration.

**Convert the following resource allocation graph to wait for graph, then&**

**Decide if there is deadlock or not? Why?**



**Answer:**

There is no deadlock because there is no cycle in the chain of waits



**CH 9…**

**طرق page replacement :**

**استخدام memory reference و تمثل بمجموعة من الأرقام وذلك للسهولة في التعامل.**

**1-FIRST IN FIRST OUT :**

**تعتمد على وقت وصول page .**

**يكون اختيار الضحية على أساس أول page دخل في memory هو الذي يكون الضحية.**

**بما أن هذه الطريقة تعتمد على Belady’s Anomaly تعني كلما زاد عدد frame زاد عدد page fault وهذا يخالف الأساس الذي يعتمد عليه page replacement .**

**ويميز هذه الطريقه أنها من أسهل الطرق.**

* **Suppose we have the following page accesses:1 2 3 4 2 3 4 1 2 1 1 3 1 4 and**

**That there are three frames& within our system. Using the FIFO replacement**

**algorithm, show in the following table the configuration of the tree frames for**

**each step, and provide the number of page faults.**

**Answer:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Step 6 | Step 5 | Step 4 | Step 3 | Step 2 | Step 1 |
| 3 | 3 | 4 | 4 | 4 | 1 |
| 4 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 3 | 3 | 3 | 3 |

**Page faults=8**

**2-OPTIMAL ALGORTHIM:**

**إنها الطريقة المثاليثة.ويكون عدد page fault أقل شيء.**

**ولكن إنها صعبة في التنفيذ لأنها تعتمد على المستقبل.لذلك تستخدم فقط في قياس مدى جودة algorithm عن طريق مقارنتها مع algorithm المستخدم.**

**أساس في اختيار الضحية هو الframe الذي سوف يستخدم بعد مده زمنيه طويله نسبيا**

* The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
* The FIFO page-replacement algorithm is easy to understand and program.
* its performance is not always good.
* On the one hand, the page replaced may be an initialization module that was used a long time ago and is no longer needed. On the other hand, it could contain a heavily used variable that was initialized early and is in constant use
* **Belady’s anomaly**: for some page-replacement algorithms, the page-fault rate may ***increase*** as the number of allocated frames Increases.

**3-LEAST RECENTLY USED (LRU) ALGORITHM :**

**هذه الطريقة قريبة من النتائج optimal algorithm .**

**تعتمد على أقدم وحدة تم استخدامها هي التي تستبدل**

**يتم تنفيذها عن طريق:**

**1-إضافة عنصر جديد في page table يسمى counter مشترك لكل   pagesيمثل access time, حيث يكون لكل page عداد,وكلما استخدمنا page زدنا العداد بواحد.**

**إذا أزلنا page من memory وأعدناها مرة أخرى إلى memory فإن قيمة counter نفس قيمة access time**

**إن أقل قيمة counter لِ page في الجدول هي التي سوف يحدث لها استبدال.**

**2-عن طريق stack .**

* we can replace the page that ***has not been used*** for the longest period of time. This approach is the **least** **recently used (LRU) algorithm**.
* We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.
* The LRU policy is often used as a page-replacement algorithm and

is considered to be good.

* The major problem is ***how*** to implement LRU

replacement. An LRU page-replacement algorithm may require substantial

hardware assistance. The problem is to determine an order for the frames

defined by the time of last use. Two implementations are feasible:

• **Counters**. In the simplest case, we associate with each page-table entry a

time-of-use field and add to the CPU a logical clock or counter. The clock is

incremented for every memory reference. Whenever a reference to a page

is made, the contents of the clock register are copied to the time-of-use

field in the page-table entry for that page.

•**Stack**. Another approach to implementing LRU replacement is to keep

a stack of page numbers. Whenever a page is referenced, it is removed

from the stack and put on the top. In this way, the most recently used

page is always at the top of the stack and the least recently used page is

always at the bottom

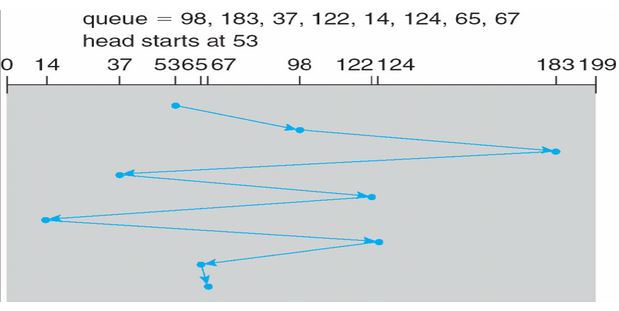
**CH 12…**

**الجدولة بحسب زمن الوصول(FCFS scheduling):**

      (FCFS) هو اختصار (First Come First Serve) أي (الأول في الوصول يخدم أولاً) وهو أبسط شكل لجدولة القرص ،كذلك فإنه عادل جوهرياً ولكن ليس الأسرع في التنفيذ دائما ً فمثلا لوكان صف مهام الإدخال والإخراج لأجزاء من الإسطوانات كالتالي (بالترتيب):.

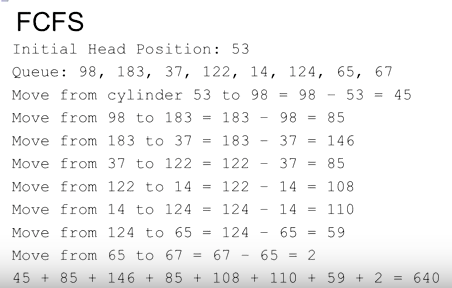
                                                        76,65,124,14,122,37,183,98

وكان رأس القراءة والكتابة عند الأسطوانة رقم53 فإنه في البداية سينتقل من 53 إلى 98 بحيث يمر خلال 45 أسطوانة (98-53=45) ثم ينتقل من 98الى 183 ليمر خلال 85 أسطوانة وهكذا حتى يصل إلى الأسطوانة رقم 97 (كما هو في الصورة التالية)، وبذلك يكون رأس القراءة والكتابة قد إنتقل خلال 640 أسطوانة لإنهاء مهام الإدخال والإخراج للقرص.



 وتظهر المشكلة في هذا النوع من الجدولة عندما انتقل رأس القراءة والكتابة من الأسطوانة 37 إلى 122 ثم عاد إلى 14ثم إلى 124، وستكون الفعالية أفضل لو أنه أنهى العمل من 37 و 14ثم انتقل للعمل على 122 و 124 وبالطبع هذا سيقلل من عدد الأسطوانات التي سيمر عليها وبالتالي ستزداد فعالية التنفيذ.

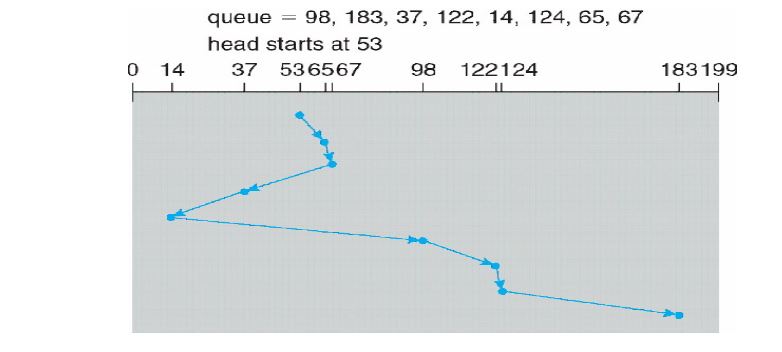
* The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm.



الجدولة بحسب زمن البحث (:(SSTF scheduling

(SSTF) هو إختصار((shortest-seek-tine-first أي (الأقصرفي زمن البحث أولاً) ويعتمد هذا النوع على خدمة أو تنفيذ المهام الخاصة بالأسطوانات الأقرب إلى موضع رأس القراءة والكتابة قبل الإنتقال إلى الأسطوانات البعيدة ويتم ذلك باختيار المهام ذات زمن البحث الأقصر ولأن هذا الأخير يزداد بازدياد عدد الأسطوانات التي سيمر عليها رأس القراءة والكتابة فإن الإختيار سيقع دائماً على المهمة ذات الموقع الأقرب من موقع الرأس.

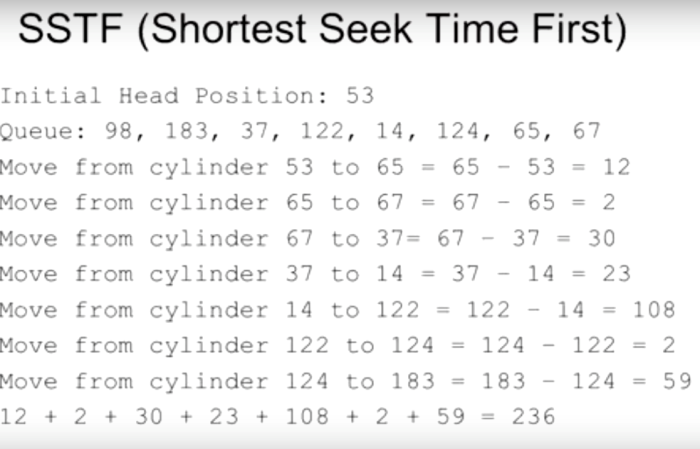
ففي مثالنا السابق ستكون أقرب أسطوانة من الموقع المبدأي للرأس 53 هي الأسطوانة 65 ثم 67 ثم 37 ثم 14 ثم 98 ثم 122 ثم 124 ثم 183 (كما هو في الصورة التالية) وبذلك يكون إجمالي الأسطوانات التي انتقل الرأس خلالها هو 236 وهو ما يقارب ثلث عدد الأسطواناتالتي مر خلالها الرأس في (FCFS) وبالتالي فهذا النوع أكثر فعالية من النوع السابق.



 ولأن المهام قد تصل في أي وقت أثناء العمل على مهمة أخرى فإن أحد المهام قد تظل تنتظر أغلب الوقت فمثلاً لو حدث أن وصلت مهمة خلال تنفيذ المهمة من أو إلى الأسطوانة 14 وكانت هذه المهمة الجديدة من أسطوانة(لتكن 20) أقرب إلى الرأس من التالية (لتكن 186) فإن الجديدة ستنفذ وستضطر التالية للإنتظار، ولو تكرر ذلك لعدة مرات فإن المهمة من أو إلى 186 ستنتظر طويلاً وربما إلى مالانهاية وهذا ما يسمى بالتجويع (starvation).

وبالرغم من أن هذا النوع أو الخوارزمية أفضل من (FCFS) إلا أنه ليس الأفضل على الإطلاق ففي المثال ستكون الفاعلية أفضل لو انتقل الرأس إلى 37 أولاً ثم إلى 14 ثم 65 ثم 67 ثم 98 ثم 122 ثم 124 ثم 183 وسيكون كذلك إجمالي الأسطوانات التي سيمر عليها 208 وهو أقل كذلك.

* SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause starvation of some requests



* Explain the SSTF scheduling algorithm, and why it may cause

starvation, Given the example queue 98, 183, 37, 122, 14, 124, 65, 67 And

the head is at 53?

Answer: SSTF is Shortest Seek Time First If the head is at 53 then the shortest distance is 65 so it will serve 65, and then 67 and so on 65, 67, 37, 14, 98, 122, 124, 183

It may cause starvation because if a shorter request keeps coming on the queue it will be served first and the longer seek time request will not get served and suffer from starvation.

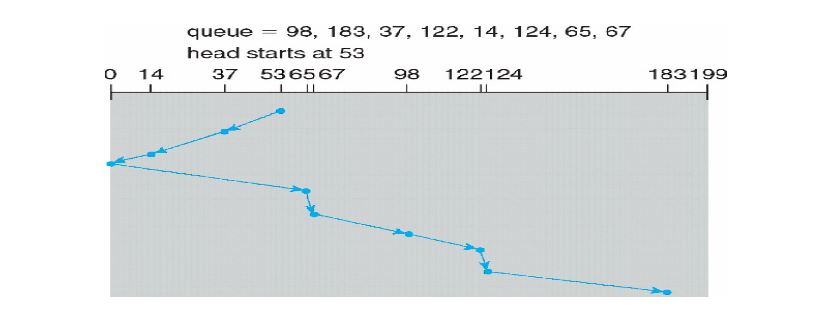
* What is a disadvantage of the SSTF scheduling algorithm?

Answer: Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal. SSTF may cause starvation of some requests. If a continual stream of requests arrives near one another, a request of a cylinder far away from the head position has to wait indefinitely.

**الجدولة بالمسح (SCAN scheduling):**

في هذا النوع من الجدولة يبدأ الذراع بتحريك رأس القراءة والكتابة من أحد نهايتي القرص وحتى النهاية الأخرى، وفي أثناء المرور خلالهما يتم تنفيذ المهام حال الوصول إلى الأسطوانة التي تحتاجها وهكذا حتى يصل الرأس إلى النهاية الأخرى لتكون بداية لرحلة جديدة من المسح خلال أسطوانات القرص وتنفيذ المهام التي تستلزم أحدها وتتكرر هذه العملية إلى مالا نهاية. ولأن الذراع هنا يعمل كعمل المصعد في المباني بحيث يخدم الأدوار التي يمر بها فإن هذا النوع من الجدولة يسمى بخوارزمية المصعد (Elevator Algorithm).

في مثالنا السابق لابد لنا من معرفة اتجاه تحرك الذراع أولاً لجدولة المهام وبافتراض أنه ينتقل باتجاه الأسطوانة صفر فإنه سينتقل من موقعه 53 إلى الأسطوانة 37 ثم 14 ثم سيصل إلى صفر ليعاود التحرك بالاتجاه المعاكس حتى يصل إلى 65 ثم 67 ثم 98 ثم 122 ثم 124 ثم 183 لينفذ المهام التي تحتاج كل منها (كما هو في الصورة التالية).

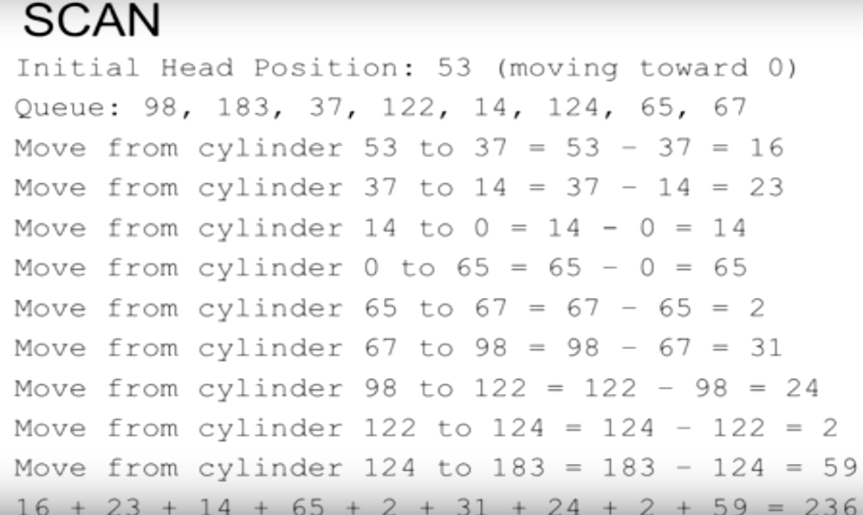


* In the **SCAN algorithm**, the disk arm starts at one end of the disk and moves

toward the other end, servicing requests as it reaches each cylinder, until it gets

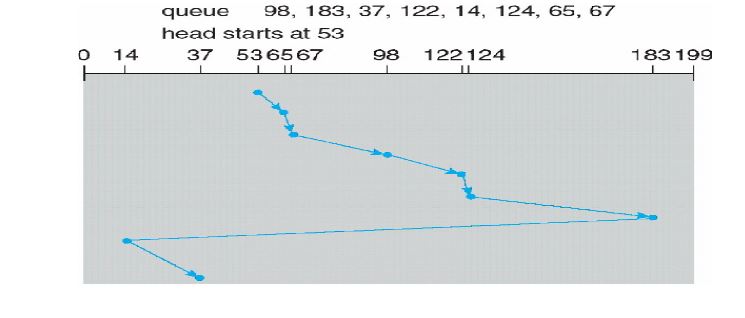
to the other end of the disk.

* The SCAN algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves just like an elevator in a building, firstservicing all the requests going up and then reversing to service requests the other way.



**جدولة البحث (LOOK scheduling):**

في هذا النوع يتحرك الذراع فقط إلى أبعد مهمة سينفذها في كل اتجاه وليس إلى نهاية القرص تماماً، والنسخ من (SCAN) و (C-SCAN) التي تستخدم هذا النموذج تسمى (LOOK) و (C-LOOK) . الشكل التالي يوضح مسار الذراع في الانتقال بالنسبة للمثال السابق:



* Versions of SCAN and C-SCAN that follow this pattern are called **LOOK** and **C-LOOK scheduling**, because they ***look*** for a request before continuing to move in a given direction

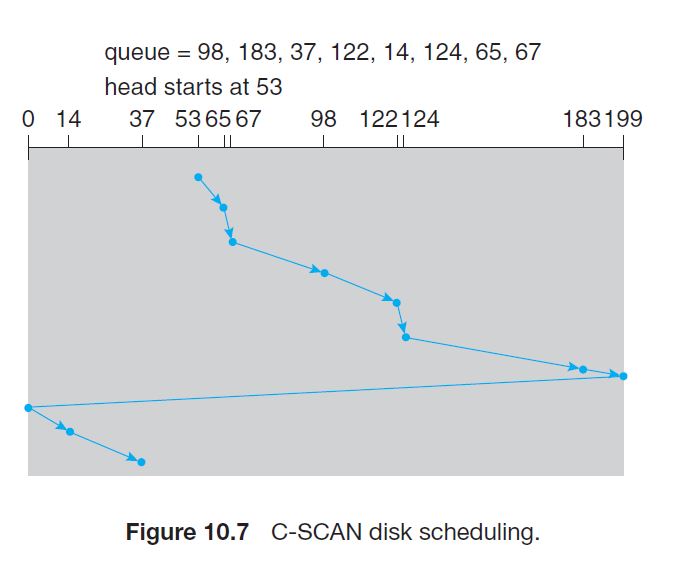
**C-SCAN Scheduling**

**Circular SCAN (C-SCAN) scheduling** is a variant of SCAN designed to provide

a more uniform wait time.

C-SCAN scheduling algorithm essentially treats the cylinders as a circular list

that wraps around from the final cylinder to the first one.



* Given so many disk-scheduling algorithms, how do we choose the best one?

SSTF is common and has a natural appeal because it increases performance over

FCFS. SCAN and C-SCAN perform better for systems that place a heavy load on

the disk, because they are less likely to cause a starvation problem. For any

particular list of requests, we can define an optimal order of retrieval, but the

computation needed to find an optimal schedule may not justify the savings

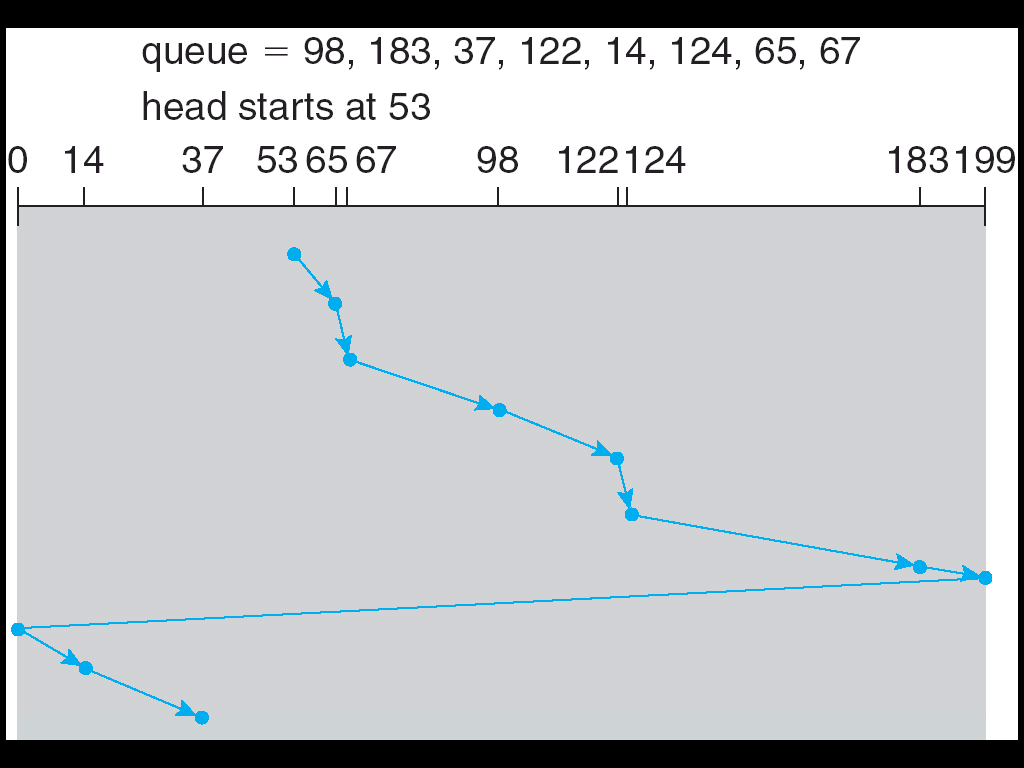
over SSTF or SCAN.

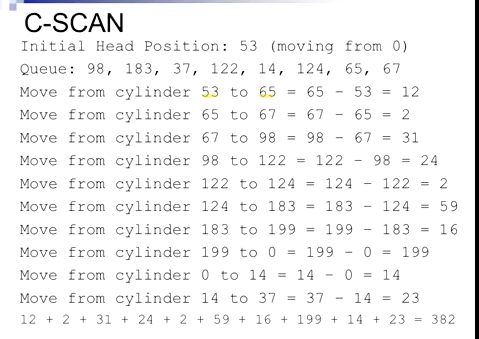
* What is the advantage of LOOK over SCAN disk head scheduling?

Answer: The LOOK algorithm is a type of SCAN algorithm. The difference is that, instead of forcing the disk head to fully traverse the disk, as is done in the SCAN algorithm, the disk head moves only as far as the final request in each direction

**C-**SCAN…

* Provides a more uniform wait time than SCAN
* The head moves from one end of the disk to the other, servicing requests as it goes
* When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
* Treats the cylinders as a circular list that wraps around from the last cylinder to the first one



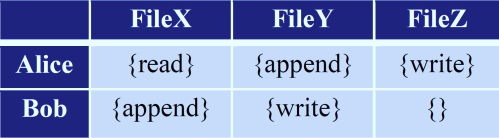


1. Alice can read file X, can append to file Y, and can write to file Z. Bob can append to file X, can write to file Y, and cannot access file Z. Design the access control matrix that specifies the described set of access rights for subjects Alice and Bob to objects file X, file Y and file Z. (1 mark)

Alice have the following permissions on Files FileX,FileY and FileZ  
Alice can read data from FileX  
Alice can append data to FileY  
Alice can write data to FileZ

Bob have the following permissions on Files FileX,FileY and FileZ  
Bob can append data to FileX  
Bob can write data to FileY  
Bob cannot have rights access right on FileZ

The Access Control Matrix for Alice and Bob as follows



1. Write the set of access control lists and the set of capability list for the given question 4. (1 mark)

**The set of access control lists:**

ACL (FileX) = Alice: {read}, Bob: {append}

 ACL (FileY) = Alice: {append}, Bob: {write}

 ACL (FileZ) = Alice: {write}, Bob: {}

**The set of capability list:**

CList (Alice) = FileX: {read}, FileY: {append}, FileZ: {write}

 CList (Bob) = FileX: {append}, FileY: {write}, FileZ: {}