Course Work Exercises Answers – Yousef Amar (1095307) EE2K1 2011

Exercise 1

A double sided disk has eight platters. Each platter has two heads and contains 1000 tracks per surface. Each track has 100 sectors, and each sector holds 512 Bytes where eight bits were used for control purpose. The disk rotates at 7200 rotations per minute. The track-to-track seek time is 2 ms and the settling time is 20ms.

i) Determine the total useful capacity of the disk.
2 * 8 * 1000 * 100 * (512-1) = 817600000 bytes or 779.724121 megabytes

ii) Determine the average time to locate a sector assuming the disk head goes back to the middle of the tracks after each seek operation.
One revolution = 1/(7200/60/1000) = 8.33 ms
(1000/2/2)*2 + 8.33/2 + 20 = 524.166ms for a head to locate a sector on average.

iii) Determine the transfer time of sector 10.
In one revolution (8.33ms), 100 sectors pass under the head. Therefore the time it takes to read one sector is 8.33/100 = 0.0833ms or 8.33us disregarding seek time.
If seek time were to be included, the time it would take to read sector 10 would be, on average, (1000/2-10)*2 + 8.33/2 + 20 + 0.0833 = 1004.2483.

iv) Explain why clusters were used instead of sector by operating system.
A list of sectors is needed for every file. For large files, this list can be very long. The list itself is stored on the disk, taking up space. Clusters, or blocks of sectors, reduce the size of this list. Instead of listing every sector, a group of sectors is recorded. In the disk in the example were to have a cluster size of 4, for instance, a cluster would be about 2kB. It would then be listed as a single cluster, instead of 4 sectors, implying the final list a quarter of the size in this case. The disadvantage here is that all files are rounded up to the nearest cluster in size, meaning a 4.1kB file would take up 6kB of memory.

v) Consider a file of 20KB that one wants to store in the above disk formatted with FAT32 format where a cluster contains 8 sectors. Determine the amount of internal fragmentation when storing the above file. Discuss the limitations of the file allocation table strategy with respect to linear directory structure.

A cluster contains 8*512B = 4kB meaning a file of 20kB could quite snugly take up 5 clusters. This mean however that 5 head seeks are required in order to read this file which can be very expensive time-wise (average of 5*8.33ms).Although the FAT algorithm tries to allocate the memory in a way that a file takes up sectors that are near each other to reduce the seek time for a single file, the disk will eventually need to defragment and bring all these "fragments" of files together in one area instead of scattered all over the disk.
This is similar to "compacting" that is done in a linear directory structured disk (although the problem is concerning space and not seek time); deleting files leaves gaps that are potentially too small for new files to be written into effectively leaving less space. If all the gaps are closed, the resulting combined free memory can be occupied by said large files. Furthermore the linear directory system, in its simplicity, only allows for a file to be stored in once continuous block of memory that likewise reiterates the problems of space being lost to these gaps. Although the head does not have to seek multiple times for a single file, a file cannot be stored in any order, as it can in the FAT format.
The FAT format allows a file to be stored in any order, sector-wise, meaning that a large file can still fill up a gap left by a deleted file. This means however that files are fragmented, increasing seek time, and a list has to be kept that records all the sectors taken up by a file which can potentially take up a lot of memory. Similarly, a list of all free sectors has to be kept.

vi) Assume that all clusters of the above file are stored in track 55, and given that the head is initially located at cylinder 0, compare the performance of the SSTF, SCAN and FCFS strategies to read the above file.

With SSTF, the seek time is significantly reduced in general, but it still might take longer for our track 55 file to be read. Theoretically though, it should do pretty well as track 55 is not too far away from cylinder 0. The only time starvation could occur, is if many sectors that need to be read were located significantly before track 55 and close together, as reading those would take advantage over track 55 and the head could go back in the other direction several times.
SCAN, on the other hand, starts at the top and sequentially moves through all requests (and then back from the inside to the outside after which the scan is repeated) meaning it will eventually reach track 55, reducing the variance as in SSTF.
FCFS, as the name implies, preforms operations in the order requested. Although this guarantees no starvation, it is not very efficient as the average seek time is much greater. Even though track 55 is relatively close to cylinder 0 in this case, there could be an earlier pending request in track 1000, meaning the head would have to move there first and then all the way back to track 55.

vii) Compare the performance of the read and write operation in the above file.

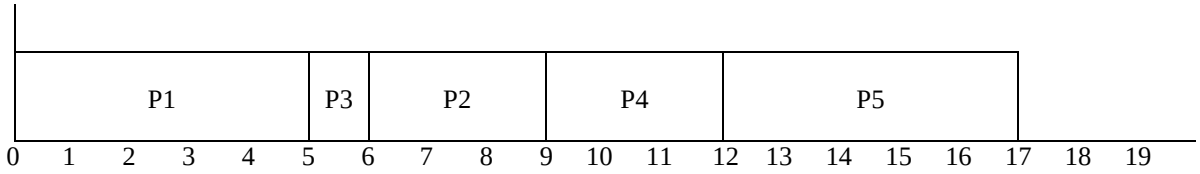
Since the entire file is in a single track, the time it takes to read and write the file is only limited by the rotation speed and latency of the disk. Generally, reading is faster than writing but their performances are still proportional to each other. Even if the clusters with in the track that the file occupies are in different orders, it wouldn't make a difference.
A sector is read at (512*8)/8.33us = 49.17 Mbits/second

Exercise 2
Consider the processes P1, P2, P3, P4 and P5 whose arrival time, burst time and priority are given below

	Arrival time	Burst time	Priority
P1	0	5	Normal
P2	2	3	Normal
P3	4	1	Normal
P4	5	3	Normal
P5	5	5	Normal

i) Draw execution time diagram and calculate the average total turnaround time, waiting time and response time when using pre-emptive shortest job first scheduling algorithm.

If on arrival, a process has the same burst time as the time remaining on the concurrently running process, pre-emption is random (or so I was told). In this case, I decided that the already running task completes first to remain consistent with context switch delays and such if they were to ever occur and lengthen the process:



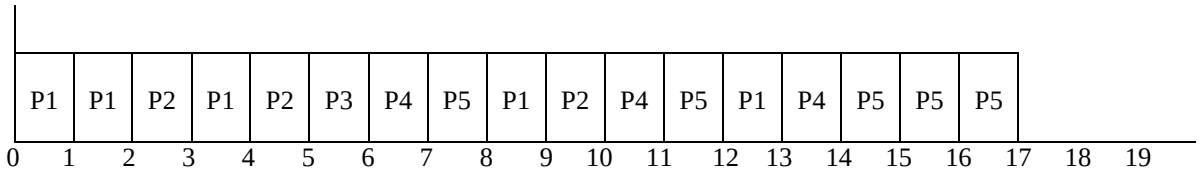
Alternatively, P2 could have pre-empted P1 on arrival at t=2 and it would have been equally as valid. The rest of P1 would have finished at in place of P2 (6 to 9). Had that been the case, the average waiting time would be less, yet the average turnaround time would have been greater. Similarly, P2 and P4 could have exchanged places, but it would make sense practically to let P2 go first as it had an earlier arrival time and although it would increase the average turnaround time overall, P2 was prone to starvation and the priority could have incremented due to aging.

In the above case:
Average turnaround time: (5 + 7 + 2 + 7 + 12)/5 = 6.6
Average waiting time: (0 + 4 + 1 + 4 + 7)/5 = 3.2 (no pre-emption)
Average response time: (0 + 4 + 1 + 4 + 7)/5 = 3.2

Yet, as mentioned, the outcome could have been very different.

ii) Repeat question i) when using Round Robin algorithm with quantum time q=1 unit.

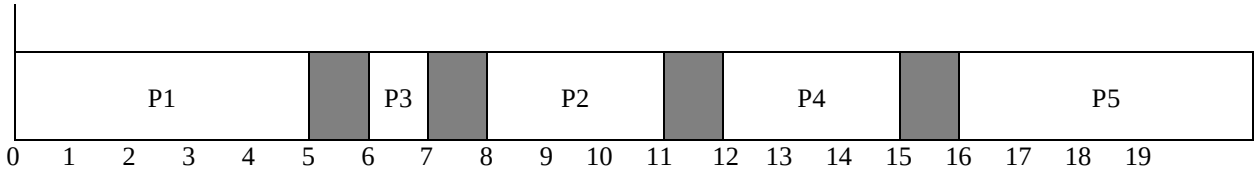
As P4 and P5 arrive at the same time, it is impossible to know which one enters the queue first. Assuming P4 does:



Average turnaround time: (13 + 8 + 2 + 9 + 12)/5 = 8.8
Average waiting time: (8 + 5 + 1 + 6 + 7)/5 = 5.4
Average response time: (0 + 0 + 1 + 1 + 2)/5 = 0.8

iii) Assume that context switch is not negligible and it takes 1 unit of time, determine the new average waiting time, average turnaround time and CPU efficiency of Shortest Job First algorithm. Discuss how the context switch influence the CPU efficiency of the chosen scheduling algorithm.

Since the SJF question was done with context switch time in mind, it should have better waiting and turnaround times. When P2 arrives with a burst time of 3 at the exact moment that P1 only has 3 units of time left, *not* switching would be better since executing P2 would in fact take *4* time units including the context switch. Whether or not this is taken into consideration in the algorithm is irrelevant.



In the above case:
Average turnaround time: (5 + 9 + 3 + 10 + 16)/5 = 8.6
Average waiting time: (0 + 6 + 2 + 7 + 11)/5 = 5.2 (no pre-emption)
CPU utilisation: (5 + 3 + 1 + 3 + 5)/21 = 17/21 = 80.95%

Yet, as mentioned, the outcome could have been different. The number of context switches is inversely proportional to the CPU efficiency; the higher the number of context switches, the less the efficiency. Had there been any pre-emption, the CPU efficiency would have been less as there would be more context switches and thus less time in which the CPU is busy on a process. If the algorithm in question is pre-emptive, it is likely to decrease the CPU efficiency (although not as much as, say, round robin). Any non-pre-emptive algorithm utilises the CPU to its maximum as the context switches that occur are unavoidable.

iv) Determine the average waiting time if multi-level queue algorithm was used where the first queue uses FCFS with maximum quantum time of 2units, the second queue uses Round Robin with quantum time 3 units.

Assuming negligible context switch and time to change queues as well as that the FCFS queue has absolute priority over the RR one (i.e. Fixed priority queue scheduling from foreground to background),

Average waiting time: $(7 + 8 + 0 + 6 + (2+5))/5 = 5.6$

v) Assumes processes P1, P2, P3, P4 and P5 are real time processes with deadlines 8, 10, 13, 16 and 15, respectively. Describe the execution of the earliest deadline first scheduling and latest deadline first scheduling algorithms.

For pre-emptive and non-pre-emptive earliest deadline first in this example the processes would naturally execute in the order P1, P2, P3, P4 and P5 to their full burst time. Latest deadline first in the opposite order although there is no reason why anyone would ever really want to use that algorithm. It makes sense to use earliest deadline first scheduling for processes where the deadline has to be kept for any reason. This carries over into real like, such as in a garage that has to fix cars for customers by a certain time. They would of course attempt to complete the ones with the earliest deadlines first. There is not much else one can say about this.

Exercise 3

i) Prove the following

Although you cannot really "prove" notation as such, the following is explaing the logic behind it:

1. $f(n) = \Theta(g(n))$ & $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

Knowing that f(n) is within the bounds of a constant number multiplied by g(n) on both sides (asymptotic tight bound), and that g(n) is within the bounds of a constant number multiplied by h(n) logically can only mean that f(n) is within the bounds of a constant multiplied by h(n) on both sides (with different constants of course) and thus f(n) = Θ(h(n)).

$f(n) \in \Theta(g(n)) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

2. $f(n) = O(g(n))$ & $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

As g is an asymptotic upper bound for f and will always be greater than or equal to it and at the same time the same relationship holds for g(n) and h(n) (that h is an upper bound to g), f(n) can only be lower than the upper bound set by h(n).

$f(n) \in O(g(n)) \in O(h(n)) \Rightarrow f(n) \in O(h(n)) \Rightarrow f(n) = O(h(n))$

3. $f(n) = \Omega(g(n))$ & $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

In contrast to the previous questions, g is an asymptotic lower bound for f and will always be less than or equal to it and at the same time the same relationship holds for g(n) and h(n) (that h is an lower bound to g), f(n) can only be greater than the lower bound set by h(n).

$f(n) \in \Omega(g(n)) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

ii) Determine the theta notation of the expressions:

S1(n) $= 1 + 2^k + 3^k + \ldots + n^k$ for any k>1

$S1(n) = \Theta(n^k)$

$S2(n) = 2 + 4 + 8 + 16 + \ldots + 2^n$

$S2(n) = \Theta(2^n)$

iii) Determine the big theta of the following pieces of code

```
j=n;
    While (j>=2)
    { for i=1 to n
      x = x+1;
     x= x*x-3;
      j=j/3;
      }
     }
```

⇨ Θ(log n) disregarding constants and lower order terms

iv) Consider a modified Quicksort algorithm in which the recursion operation of quicksort is maintained until the number of elements of the underlying subarray is less than 10 elements where bubble sort algorithm is used instead of quick sort.

- Write a pseudo code explaining the functioning of this new Quicksort algorithm.

  The following code is my actual code in simplified pseudocode form. For the full detailed pseudocode (explaining quicksort and bubble sort) and the code listing, see lab report.

```
If rightmost index is less than 10
        Bubblesort
Else if rightmost index is greater than to the leftmost index
        Recursively quicksort using this same function
```

- Write a recursion expression indicating the complexity of this algorithm

  $T(n) = n*n*k + c$ for $n<10$ and $n>1$
  $T(n) = 2*T(n/2) + c$ for $n\geq10$ and $n>1$

- Solve this recurrent relation and determine the complexity of this algorithm

  $O(n^2)$ for $n<10$
  $O(n \log n)$ for $n\geq10$

  All while $n>1$