

CSE 120, Answers to HW2

Chapter 2

Q 30 (See Tanenbaum 2.3.1)

No, as long as mailbox access is serialized.

Q 35

If a process occurred twice, it would just be scheduled twice as often as other processes. This could be used as a way to allow higher priority processes to get more CPU time.

Q 38

CPU efficiency here is the fraction of time the CPU is being used for purposeful work.

(a) $Q = \infty$

A process is most often switched out before its quantum runs out. Out of $T+S$ total CPU time, actual processing only takes time T . The efficiency is therefore $T/(T+S)$.

(b) $Q > T$

Same as before, $T/(T+S)$.

(c) $S < Q < T$

A simple answer relies on the fact that processes would mainly be swapped out due to their quanta running out, so we would have $Q/(Q+S)$ for efficiency.

The above argument does not take into account the fact that T may not be an exact multiple of Q and that every once in a while a process gets switched out before it runs out of its quantum. For an actual processing time of T , $\lfloor T/Q \rfloor + 1$ context switches are required, so we have an efficiency of $T/(T + (\lfloor T/Q \rfloor + 1)S) \approx T/(T + (T/Q)S)$.

(d) $Q = S$

It is reasonable to assume that $T \geq S$. So replacing S by Q in the above formula we have an efficiency of about $1/2$.

(e) Q nearly 0.

In other words T is big enough that it has almost no effect on when context switches happen. We therefore simply have an efficiency $Q/(Q+S)$. If S is considerably bigger than Q then the efficiency is close to 0.

Q 40

In all the answers below T_x represents the turnaround time for the job that needs x minutes of CPU time.

We assume that the processing time for process switches is close to 0.

(a) Round Robin

By the time the shortest job is finished running any other job would have ran just as long. Therefore since there are 5 total jobs they would have all ran for a total of 5×2 minutes when the shortest job finishes. Every job by then would have used 2 minutes of its running time so the next shortest job would run for an additional $(4-2) \times 4$ minutes. Applying this reasoning sequentially for every job (shortest jobs first) we get the turnaround times

$$T2 = 2*5 = 10$$

$$T4 = 2*4 + T2 = 18$$

$$T6 = 2*3 + T4 = 24$$

$$T8 = 2*2 + T6 = 28$$

$$T10 = 2 + T8 = 30$$

and we have an average turnaround of 22.

(b) Priority scheduling

$$T6 = 6$$

$$T8 = 8 + T6 = 14$$

$$T10 = 10 + T8 = 24$$

$$T2 = 2 + T10 = 26$$

$$T4 = 4 + T2 = 30$$

and we have an average turnaround time of 20.

(c) First-come, first-served.

$$T10 = 10$$

$$T6 = 6 + T10 = 16$$

$$T2 = 2 + T6 = 18$$

$$T4 = 4 + T2 = 22$$

$$T8 = 8 + T4 = 30$$

and we have an average turnaround time of 19.2

(d) Shortest job first

$$T2 = 2$$

$$T4 = 4 + T2 = 6$$

$$T6 = 6 + T4 = 12$$

$$T8 = 8 + T6 = 20$$

$$T10 = 10 + T8 = 30$$

and we have an average turn around time of 14.

Q 41 (See Tanenbaum 2.5.3)

The process would be swapped in 5 times, using 1 quantum the first time, then 2, 4, 8, and last 15 quanta out of 16 allotted to it.

Q 43

After the first two runs the prediction is $\frac{1}{2}(40 + 20)$. After the third the prediction is $\frac{1}{2} \frac{1}{2}(40 + 20) + \frac{1}{2}(40)$ and the prediction after the last run is then $\frac{1}{8}(40 + 20) + \frac{1}{4}(40) + \frac{1}{2}(15) = 25$.

Chapter 3

Q 15

Deadlock is not possible here because one process would necessarily acquire the two resources it needs and we therefore would never get into the circular dependency that creates a deadlock.

Q 16

To be deadlock free we need enough resources so at least one process is fully satisfied, and when this process releases its resources, at least one other process should be able to get all the resources it needs, and so forth till all the processes complete. Moreover, this should be true independently of the order in which resources are acquired. To satisfy these requirements we need to have all m resources needed by at least one process, and every other process should be at most one resource short of their need, i.e. they must be able to obtain at least $m - 1$ resources.

In other words we need the condition $r \geq m + (p - 1)(m - 1)$ or simply $r > p(m - 1)$.

Q 20 (See Tanenbaum 3.5.2)

The state shown is a safe state if there exists an allocation ordering that would prevent deadlocks. To check this let's first compute the current needs of every process:

	Allocated	Maximum	Current need	
A	10211	11213	01002	and the current availability is of 00x11
B	20110	22210	02100	
C	11010	21310	10300	
D	11110	11221	00111	

We see right away that x must be at least 1 for this state not to be deadlocked. We can then run process D to completion at which point the availability is $11(x+1)21$.

Only C can then be ran provided $(x+1)$ is at least 3, i.e. x is at least 2. When C completes the availability is of $22(x+1)31$ and B can then be ran. When B completes the availability is of $42(x+2)41$ and process A can still not be satisfied because it requires 2 more instances of the fifth resource and there is only 1 available.

According to the definition of *safe state* there is no value of x for which this state is safe since process A requires more resources than the total existing. Yet, as we have shown, for $x \geq 2$ there exists a scheduling order that avoids deadlocks.

Q 24

In this system a deadlock could only occur when n processes, $P_1, P_2, \dots, P_n$, are interested in accounts $x_1, x_2, \dots, x_n$, where P_1 holds x_1 and wants x_2 , P_2 holds x_2 and wants x_3 , ..., and finally P_n holds x_n and wants x_1 .

Let $n = 2$ and we quickly notice that a deadlock would not be possible if P_n tried to acquire x_1 first before x_n . It turns out that the system would be deadlock free if we simply impose the same order of acquisition on all processes:

Devise an ordering scheme on the account numbers (trivial if these are actual numbers). If under this scheme $x < y$, then any processes interested in both x and y must acquire x first, then y .

In the scenario given above, let $x_1 < x_2 < \dots < x_n$, then P_n would not hold x_n since it'll be waiting on x_1 , so P_{n-1} is able to complete, allowing every other process to eventually complete.

Chapter 4

Q 1 (See Tanenbaum 4.1.3)

The CPU is wasted only when all four processes are waiting for I/O at the same time. Since each is idle $1/2$ of the time, the fraction of the CPU wasted is $1/2^4 = 1/16$ (assuming independence between processes).

Q5

(a) 12 KB

First Fit

The 20 KB hole is taken and an 8 Kb hole is left on the first request. Second, the 18 KB hole is taken which leaves a 6KB hole, then the last two holes taken will be the 12 KB and the 15 KB holes.

Request #	1	2	3	4
Hole Taken	20	18	12	15
New Hole	8	6	0	3

Best Fit

Request #	1	2	3	4
Hole Taken	12	15	18	20
New Hole	0	3	6	8

Worst Fit

Request #	1	2	3	4
Hole Taken	20	18	15	12
New Hole	8	6	3	0

Next Fit

Request #	1	2	3	4
Hole Taken	20	18	12	15
New Hole	8	6	0	3

(b) 10 KB

First Fit

Request #	1	2	3	4	5	6
Hole Taken	10	20	10	18	12	15
New Hole	0	10	0	8	2	5

Best Fit

Request #	1	2	3	4	5	6
Hole Taken	10	12	15	18	20	10
New Hole	0	2	5	8	10	0

Worst Fit

Request #	1	2	3	4	5	6
Hole Taken	20	18	15	12	10	10
New Hole	10	8	5	2	0	0

Next Fit

Request #	1	2	3	4	5	6
Hole Taken	10	20	10	18	12	15
New Hole	0	10	0	8	2	5

(c) 9 KB

First Fit

Request #	1	2	3	4	5	6	7	8
Hole Taken	10	20	11	18	9	9	12	15
New Hole	1	11	2	9	0	0	3	6

Best Fit

Request #	1	2	3	4	5	6	7	8
Hole Taken	9	10	12	15	18	9	20	11
New Hole	0	1	3	6	9	0	11	2

Worst Fit

Request #	1	2	3	4	5	6	7	8
Hole Taken	20	18	15	12	11	10	9	9
New Hole	11	9	6	3	2	1	0	0

Next Fit

Request #	1	2	3	4	5	6	7	8
Hole Taken	10	20	11	18	9	9	12	15
New Hole	1	11	2	9	0	0	3	6

Note:

Next fit and first fit are the same in this scenario, assuming the hole left after picking a large hole is ahead of the last fit index.

Q11

We have one page fault for k instructions. This means that k instructions take a total of $10k + n$ nsec, in other words the effective instruction time is of $10 + n/k$ nsec.

Q12

8KB is 2^{13} bytes, i.e. we require 13 bits for the offset into the page. That leaves 19 bits for the pages, or 2^{19} table entries. It takes $2^{19} * 100$ nsec to load the entire table, so the fraction of the CPU time devoted to loading page tables is $(2^{19} * 100 \text{ nsec}) / (100 \text{ msec}) = 2^{19} * 10^{-6} = 0.52$

Q14

The number of pages can be determined by d alone. The size of this field determines the size of a page ($2^{|d|}$), so just divide the total size of the address space (2^{32}) by the page size to obtain the number of pages: this is $2^{32-|d|}$.

Q17

Let the miss rate be $1/k$ (a hit rate of $k - 1/k$), i.e. there is a TLB miss for every k instructions. For each TLB miss we need to read the table in memory (5 nsec), and update the TLB. Here we assume the time for the update is negligible compared to the memory read overhead. So the overhead in k instructions is of $k + 5$ nsec for an average overhead of $1 + 5/k$ nsec. For the mean overhead to be of 2 nsec we therefore require $k = 5$ for a hit rate of $4/5$.

Q21

There would be as many elements in the hash table as page frames in main memory. For an average hash chain length of less than 1, we need more hash slots than hash elements. There are a total of $256/8 = 32$ page frames, so the hash table must have more than 32 slots. If we want the hash table size to be a power of 2, then we can use 64 slots (assuming each slot size is a power of 2, which it normally is since it needs only hold a pointer).

Q37

Fragmentation is when the physical memory contains unusable holes after a serie of allocation and de-allocation of chunks of memory.

With internal fragmentation these holes are due to allocation/de-allocation of memory within a single address space and occurs with paging systems. These are due to processes not filling up pages given to them entirely.

With external fragmentation on the other hand the holes are in between memory allocated to segments, each segment corresponding to a logical address space. These therefore occur in systems using pure segmentation.

Q Banker's Algorithm

If P4's request is allowed we will be in the following state:

Processes	Allocation	Maximum	Current need	
P1	45	60	15	There are $150 - 125 = 25$ units now available.
P2	40	60	20	
P3	15	60	45	
P4	25	60	35	

notice that P1 alone holds more resources than any single other process still needs, so as long as we can run P1 to completion the state is safe. With 25 units still available, P1 canbe ran to completion so we can grant the request. If instead P4 asked for 35 units to start we would have 15 units left and again P1 can be ran to completion, so we can also grant this request.