CSE306 - Homework and Solutions

Note: your solutions must contain enough information to enable me to judge whether you understand the material. Simple yes/no answers do not count. A simple core dump of everything you remember will not do it either. Take a look at the solutions to see what level of detail is required.

1. Consider a hypothetical 32-bit microprocessor with 32-bit instructions composed of two fields: 1-byte opcode and the rest used for addressing main memory.

   What is the maximum *directly addressable* memory capacity for this processor?

2. Suppose we have a multiprogrammed computer where each job has identical characteristics: Each job runs N computational periods of length T, where half of that time is spent on I/O and half on computation. Jobs are dispatched in a simple round-robin manner and I/O activity can overlap with CPU operations.

   Compute the average turnaround time (average total time to complete each job), throughput (average number of jobs completed per time period T), and processor utilization (percentage of time that the processor is not idle) for the cases of *one*, *two*, and *four* simultaneous jobs for the following two cases:

   (a) each period T is distributed as follows: CPU – first half; I/O – second half;

   (b) CPU – during the first and the third quarter; I/O – during the other quarters.

3. Suppose that the dispatcher uses an algorithm that favors programs that have used less CPU time in the recent past. Explain why this algorithm favors I/O bound jobs (*i.e.*, those that use more I/O time than CPU time) and, at the same time does not permanently deny CPU to CPU-bound jobs (*i.e.*, those that use more CPU than I/O).

4. If a process exits and there still are threads of that process running, what happens to the thread?

5. Consider an OS that supports only user-level threads and allows to run only one process at a time. Can a scheduler for *user-level* threads implement preemptive multitasking, such as round-robin without the help of the kernel? Explain.

6. Consider a segment table

Segment	Base	Limit
0	200	500
1	2000	50
2	100	90
3	1000	900

   Assuming that the first digit represents the segment, what are the physical addresses for the following logical addresses? Which references are illegal?

(a) 0400
(b) 3100
(c) 1060
(d) 3900
(e) 2089

7. Suppose you have a machine whose hardware supports paging but not segmentation. Is it possible to implement an OS for this machine that would support segmentation efficiently? Assume there is no translation look-aside buffer. Explain.

8. In the above problem, now take into account the effect of having the look-aside buffer.

9. Suppose pages were referenced in the following order: A,B,C,D,A,B,E(w),A(w),B,C,D,E, where (w) means that the corresponding page was written to. Assume that initially the main memory was empty and that it contains just 3 frames. How many page transfers there will be under these page replacement strategies:

   (a) FIFO
   (b) LRU

10. For the previous problem and the FIFO page replacement algorithm, assume that the hardware uses an **inverted** page table. Draw pictures that show the evolution of the inverted page table as pages are swapped in or out. Include dirty bits and whatever is necessary.

11. Consider hardware that is based on inverted page tables. Does the format of page table entries require the valid/invalid bit? Explain.

12. Suppose that we have five jobs to schedule. The jobs have arrived in the order A, B, C, D, E and have CPU bursts as follows: 10, 1, 2, 3, 5.

    Compute the turnaround and average waiting times for each of the jobs using:

    (a) FCFS
    (b) SJF
    (c) Round-robin with time quantum=2

    Assume that all these jobs are already waiting and the CPU has just become available.

13. In the previous case, assume that the arrival times are 0, 2, 3, 4, 5 and that scheduling strategy is preemptive priority scheduling with priority function 2*waitingTime – remainingCPUtime (the higher the value, the higher the priority). Compute the average turnaround and average waiting times.

14. An instruction takes 2 time units to execute. Processing a pagefault takes an *additional* n time units (just the cpu time, I/O is asynchronous). Page faults happen every k instructions. Give a formula for the effective instruction time (*i.e.*, the average number of time units CPU needs to execute an instruction). Explain.

15. Similar to the previous problem, but now we have a fast cache where memory pages are kept (note: memory pages themselves are kept in the cache—not the page table entries).

    The times are as follows:

    > Referencing a word of a page in the cache: c
    > Referencing a word in main memory: m to load page in cache, then restart the reference using cache.
    > Referencing a word on disk: d to bring page from disk to main memory, then copy page to cache, then restart reference.
    > The cache hit ratio is 0.9
    > The main memory hit ration is 0.6

    Ignoring the overhead of accessing page tables, compute the average time required for memory access in such a system?

16. Which type of processes is generally favored by a multilevel feedback queuing scheduler—the processor-bound jobs or the I/O-bound ones? Explain.

Solutions

1. $2^{24} = 16$M. This is because 32 (instruction size) - 8 (opcode) is 24.

2. In case of 1 job, CPU utilization is 50% for both (a) and (b). In case of two jobs it is N/(N+0.5) for (a) and N/(N+0.25) for (b). In case of four jobs it is 2N/(2N+0.5) for (a) and 2N/(2N+0.25) for (b). (Draw the execution diagrams to see that.)

   Assume that all jobs arrived at the same time, so we count the turnaround time of all jobs from the same moment. The turnaround time in case of 1 job is N*T.

   In case (a) for 2 jobs, the individual turnaround times are NT, (N+0.5)T and the average is (N+0.25)T. In case (b), the average is 0.5(NT + (N+0.25)T) = T(N+1/8).

   In case (a) for of 4 jobs, the turnaround times for the individual jobs are 2NT-T, (2N-0.5)T, 2NT, (2N+0.5)T and the average is (2N-0.25)T. In case (b), the individual turnaround times are (2N-0.5)T, (2N-0.25)T, 2NT, (2N+0.25)T and the average is (2N+7/8)T.

   All these figures are obtained by drawing and then examining the execution diagrams. For instance, in case of 4 jobs and the execution pattern (a), such a diagram might looks as follows:

```
              T       2T       3T      4T       5T       6T
CPU | 1   2   3   4 | 1   2   3   4 |
    |---+---|---+---|---+---|---+---|---+---|---+---|
I/O |     1   2   3 | 4   1   2   3 | 4

       R o u n d   1   R o u n d  2    R o u n d  3
```

   We see that each round of execution where all jobs finish their time period T takes 2T+0.5T. The next round will finish at 4T+0.5T, the following at 6T+0.5T, etc. Therefore, executing all jobs will take 2TN+0.5T time units. Job 4 ends at this time, job 3 ends 0.5T earlier, job 2 ends another 0.5T before that, etc.

3. I/O bound jobs are likely to use less CPU time in the recent past, so they'll be favored.

   CPU-bound jobs will not be starved permanently because eventually their recent CPU usage will decline. So their priority will keep increasing until they are scheduled.

4. They are terminated as well.

5. No. Such a scheduler runs as a user process; moreover, it cannot run concurrently with the threads it manages, since there is only one process and no kernel-level threads and so the system can run only one thread at a time. Thus, this user-level scheduler cannot preempt the threads it manages. (This is precisely why you cannot assume that Java threads will preempt each other on every platform, so the programmer must make sure that periodically threads yield control voluntarily.)

6. The absolute addresses are as follows:

   (a) 600

   (b) 1100

   (c) illegal

   (d) illegal (Why?)

   (e) 189

7. Yes. You need to do a little more work in software when segments are loaded, but when the program runs there is no additional loss of cycles.

   Consider a 32 bit machine with 20 bits reserved for page numbers. Suppose we want to implement a segmentation schema were 10 bits are reserved for segment number and 22 for offset.

   We can divide each page table into sets of 2^{10} consecutive page entries. Each such set will represent a segment. We also need a segment table that will keep track of segment sizes. This is necessary because the page table format leaves no room for the segment size parameter.

   When a segment, n, is loaded, the OS must fill the corresponding page entries $n * 2^{10}$ through $(n + 1) * 2^{10} - 1$ with consecutive frame numbers that correspond to the frames where the segment is being loaded. Page entries that are assigned to the segment but lie beyond the segment boundary are marked as invalid. For instance, if our segment n has size 2^5 pages, then all page entries in the range $n * 2^{10} + 2^5$ through $(n + 1) * 2^{10} - 1$ will have their presence bit set to "invalid."

   Once segment is loaded, reference to bytes inside that segment can be done through the paging hardware without any loss of efficiency.

   There also is some extra work at the time segment is swapped out, because it will be necessary to check the dirty bits of *all* pages that belong to the segment. On the other hand, there is much less work in I/O when the image of a segment needs to be saved on disk, since the OS only need to save the specific dirty pages, not the entire segment.

8. With TLB, there will be slight loss of efficiency compared to segmentation hardware. Indeed, in a machine with paging hardware, TLB will be designed to accommodate page table entries, not segment table entries. To better see what happens, suppose that the first page table entry, T_1, for some segment S is in TLB, but the CPU must process a memory reference to the second page. Even though we can get the address of the segment from T_1 (since T_1 has the address of the first frame in the segment), our paging hardware does not know it. Instead, it will try to look up the second entry in the page table for S and save it in TLB (possibly replacing T_1!).

9. We consider memory snapshots under each page replacement algorithm separately. Numbers in parentheses show the number of page transfers.

(a) FIFO: A,B,C (3). D,B,C (1). D,A,C (1). D,A,B (1). E,A,B (1). E,C,B (2). E,C,D (1). Total: 10. Note that when A is replaced by C (the E,C,B snapshot), A is dirty, so page replacement takes two I/Os.

(b) LRU: A,B,C (3). D,B,C (1). D,A,C (1). D,A,B (1). E,A,B (1). C,A,B (2). C,D,B (2). C,D,E (1). Total: 12. Note that in the snapshot E,A,B, page C replaces E and not A (unlike FIFO), because A was referenced more recently than E. The cost of replacing E with C is 2, since page E is dirty.

10. We only show the frame number, the dirty bit (D), the reference bit (R), and the process id (PID) in each page table entry. We use some arbitrary process ids to illustrate the fact that page entries in the inverted page table do not need to belong to the same process. We also assume that the hardware (or some software daemon) clears up reference bit after 3 ticks. In the picture, below, the content of the inverted page table changes left-to-right and top-to-bottom.

	Frame#	PID	D	R
A	1	11	0	1
B	2	11	0	1
C	3	12	0	1

	Frame#	PID	D	R
D	1	12	0	1
B	2	11	0	1
C	3	12	0	1

	Frame#	PID	D	R
D	1	12	0	1
A	2	11	0	1
C	3	12	0	1

At this point, the reference bit (R) is cleared.

	Frame#	PID	D	R
D	1	12	0	0
A	2	11	0	0
B	3	11	0	1

	Frame#	PID	D	R
E	1	13	1	1
A	2	11	0	0
B	3	11	0	1

	Frame#	PID	D	R
E	1	13	1	1
A	2	11	1	1
B	3	11	0	1

	Frame#	PID	D	R
E	1	13	1	0
A	2	11	1	0
B	3	11	0	1

	Frame#	PID	D	R
E	1	13	1	0
C	2	12	0	1
B	3	11	0	1

	Frame#	PID	D	R
E	1	13	1	0
C	2	12	0	1
D	3	12	0	1

	Frame#	PID	D	R
E	1	13	1	1
C	2	12	0	0
D	3	12	0	0

11. No. By definition, inverted page tables contain page entries only for valid pages.

12. We consider the execution sequences in each case:

(a) FCFS: AAAAAAAAAABCCDDDEEEEE
Turnaround: (10+11+13+16+21)/5 = 14.2
Wait time: (0+10+11+13+16)/5

(b) SJF: BCCDDDEEEEEAAAAAAAAAA
Turnaround: (21+1+3+6+11)/5 = 8.4
Wait time: (11+0+1+3+6)/5 = 4.2

(c) RR with time slice=2: AABCCDDEEAADEEAAEAAAA
Turnaround: (21+3+5+12+17)/5 = 11.6
Wait time: (11+2+3+9+12)/5 = 7.4

13. The CPU bursts start as follows: AABCC.
At this point A has priority 2*3 – 8 = -2 and D has priority 2*1 – 3 = -1. E has priority -5 at this point, so D gets to execute. However, after 2 ticks, D's priority becomes 2*1 – 1 = 1 and A's priority becomes 2*5 – 8 = 2. The priority of E is 2*2 – 5 = -1. So, A preempts: AABCCDDAA
At this point the priorities are: A: 2*5–6=4, D: 2*3–1=5, E: 2*4–5=3. Thus, D preempts and finishes. A is next:
AABCCDDAADAA
Now the priorities are: A: 2*6–4=8, E: 2*7–5=9, so E preempts:
AABCCDDAADAAEE
New priorities: A: 2*8–4=12, E: 2*7–3=11, so A preempts:
AABCCDDAADAAEEAA
Next E preempts for two ticks and then A. At this point A terminates and then E terminates: AABCCDDAADAAEEAAEEAAE.

Turnaround: (20+1+2+6+16)/5 = 9.0
Wait time: (10+0+0+3+11)/5 = 4.8

14. Normal instruction takes 2 units. Pagefaulting instruction takes $n + 2$ units. Every k-th instruction pagefaults. Thus, on the average, it takes this long to execute k instructions:

$$2 * (k - 1) + (n + 2) \;=\; 2k + n$$

Therefore, the average number of time units per instruction is: $(2k + n)/k \;=\; 2 + n/k$.

15. Reference time for cache is c; reference time for main memory is $c + m$, and for disk it is $d + m + c$. The probability of missing the cache is 0.1. Out of this, the probability of fetching from main memory is 0.6. So, the probability of finding the referenced word in main memory (but not in cache) is $0.1 * 0.6 = 0.06$. The probability of having to go to disk is $0.1 * (1 - 0.6) = 0.04$. Thus, the average access time is:

$$0.9c + 0.06(c + m) + 0.04(c + m + d) \;=\; c + 0.1m + 0.04d$$

16. As processes use more and more CPU, they are moved down to the lower-priority queues. I/O-bound processes, which do not use much CPU, are moved up. So, this strategy favors I/O-bound jobs.