EECS 678 – Fall 2019

Answer Key to Selected Final Exam Review Questions

1 Chapter 6

1. What is a critical section? What are the three conditions to be ensured by any solution to the critical section problem?

   Answer: Slide 7.

2. The following code only ensures two of the three conditions for solving the critical section problem. What condition does it not support? Modify the code to provide support for all three conditions to address the critical section problem.

```
int mutex = 0;

do {
    while(TestAndSet(&mutex));

    // Critical section

    mutex = 0;

    // Remainder section
}while (TRUE);
```

   Answer: The code does not support the condition for bounded waiting. See slides 19–31 for solution.

3. Provide the pseudo code definitions of the following hardware atomic instructions:
   (a) TestAndSet and (b) Swap.

4. A general synchronization solution using locks looks as follows:

```
int mutex;
init_lock(&mutex);

do {
   lock(&mutex);

   // critical section

   unlock(&mutex);

   // remainder section
}while(TRUE);
```

Provide the definitions of *init_lock*, *lock*, and *unlock* when using (a) TestAndSet, (b) Swap and (c) Semaphores.

5. Define semaphores (simple semaphores and semaphores with no busy waiting).

   Answer: Slides 32, 33, 36.

6. What is *priority inversion*? How does the *priority inheritance protocol* address this issue?

   Answer: slide 39.

7. Explain why spinlocks are not appropriate for uniprocessor systems, yet may be suitable for multiprocessor systems.

   Answer: On uniprocessor systems, a process that spins while waiting for the lock stalls the process holding the lock (and executing in the critical region). Thus no progress is made.

   (I will let you extrapolate on what happens on a multiprocessor to make this suitable. Also, think about when spinning will be ideal to do on multiprocessors.)

8. Show that if the wait and signal semaphore operations are not executed atomically, then mutual exclusion may be violated.

9. List the drawbacks of Semaphores. How can Monitors overcome the drawbacks?

10. How do monitors ensure mutual exclusion?

11. The code on Slide 53 shows a solution to the bounded buffer problem using monitors and condition variables. Using illustrative figures (as demonstrated in class) show how the following sequence of process events will be handled with (a) Hoare and (b) Mesa condition variable semantics.

    Events: C1 P1 C2

    Here, P*n* indicates 'producer' and C*n* indicates a 'consumer' process.

12. How do the semantics of the *wait()* and *signal* operations differ in semaphores and monitors?

2 Chapter 5

1. (a) ____ is the amount of time to execute a particular process (submission time to completion time).
   (b) ____ is the amount of time a process is waiting in the ready queue.
   (c) ____ modeling uses a pre-determined workload and defines the performance of each algorithm for that workload.

2. Describe the difference between preemptive and non-preemptive scheduling. State why strict non-preemptive scheduling is unlikely to be used on machines supporting programs that provide interactive user services.

3. The processes listed below(P1, P2, P3, P4, and P5) are assume to all arrive at time 0. Perform the following analysis addressing how different scheduling algorithms would execute these processes, and how each would perform as measured by different metrics.

   - (a) Draw 4 Gantt charts illustrating the execution of these processes using FCFS, SJF, non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum=1) scheduling

Name	Burst Time	Priority
P1	3	1
P2	7	2
P3	2	2
P4	4	4
P5	1	3

- (b) What is the turnaround time of each process for each of the scheduling algorithms specified in part (a)?
- (c) What is the waiting time of each process for each of the scheduling algorithms given in part (a)?
- (d) Which of the algorithms in part (a) results in the minimum average waiting time (over all processes)?

4. Most round-robin schedulers use a fixed size CPU time quantum for allocating CPU time. Large quantum sizes provide certain advantages to the system, while small quantum sizes provide other advantages. Assume that you are designing a system where throughput is more important than response time, while use of round-robin scheduling is required. Explain whether you would use a relatively large or relatively small quantum value for such a system, and why.

   Answer: Larger time quantums reduce the ratio of time spent in the context-switch overhead, and can enable higher overall system performance. But, this higher performance comes at the cost of responsiveness, which can be better provided with a shorter time quantum. In this problem, since throughput is more important, we will use a larger time quantum.

5. Five batch jobs, P1 through P5, arrive at a computer center at essentially the same time. Their burst times and priorities are defined by the table below. 0 is the best priority. For

Name	Burst Time	Priority
P1	15	2
P2	5	8
P3	11	5
P4	9	1
P5	8	7

   each of the following scheduling algorithms, determine the turnaround time for each process, and the average turnaround for all jobs. Ignore process switching overhead. Show how you arrived at your answers.

   Except for Round Robin, assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

   - First Come First Serve (P1 first, P6 last)
   - Shortest Job First
   - Non-Preemptive Priority
   - Round Robin with a quantum of 1

6. Devise an approximation formula to estimate the length of the next CPU burst for the following criteria: (a) length of last CPU burst and past prediction histories are equally weighted. (b) past prediction history does not count.
What would each formula *guess* for each of its next CPU bursts for a sequence of real CPU burst times that appear as following:
CPU burst (t_i): 4 10 10 10 2 6 10 10 6 4 4
Assume that the *initial* guess time is 6 time units.

Answer: We can start from the same approximation formula as used in the class (Slide 22) to estimate the length of the next CPU burst.

$\tau_{n+1} = \alpha * t_n + (1\text{-}\alpha)\ \tau_n$

(a) To equally weigh the length of last CPU burst and past prediction histories: we use $\alpha = 0.5$
Then, with $\alpha = 0.5$ and $\tau_0 = 6$, and **CPU burst (t_i)**: 4 10 10 10 2 6 10 10 6 4 4

$\tau_0 = 6$,
$\tau_1 = \alpha * t_0 + (1-\alpha)\tau_0 = 1/2*4 + 1/2*6 = 5$
$\tau_2 = \alpha * t_1 + (1-\alpha)\tau_1 = 1/2*10 + 1/2*5 = 7.5$
$\tau_3 = \alpha * t_2 + (1-\alpha)\tau_2 = 1/2*10 + 1/2*7.5 = 8.75$
$\tau_4 = \alpha * t_3 + (1-\alpha)\tau_3 = 1/2*10 + 1/2*8.75 = 9.375$
$\tau_5 = \alpha * t_4 + (1-\alpha)\tau_4 = 1/2*2 + 1/2*9.375 = 5.6875$
...

(b) To ignore the past prediction history: we use $\alpha = 1.0$
Then, $\tau_{n+1} = t_n$,
For **CPU burst (t_i)**: 4 10 10 10 2 6 10 10 6 4 4

$\tau_0 = 6$,
$\tau_1 = t_0 = 4$
$\tau_2 = t_1 = 10$
$\tau_3 = t_2 = 10$
$\tau_4 = t_3 = 10$
$\tau_5 = t_4 = 2$
...

7. Why is *aging* used during priority scheduling algorithms?

Answer: Priority scheduling algorithms can cause *starvation* for low-priority jobs (in cases when there is a steady stream of higher priority jobs entering the system). Aging will gradually raise the priority of *older* jobs over time, so they will eventually get a chance to run.

8. What is multi-level queue scheduling? Explain its main features.

Answer: Multi-level queue scheduling algorithms maintain multiple job queues. They then perform scheduling at two levels: (a) inter-queue scheduling – to select the queue from which a job will be run on the CPU, and (b) intra-queue scheduling – to find the job from the selected queue to run.

An important feature of multi-level queue scheduling algorithms is that the jobs do not automatically move between the various queues.

See slides 32-35 for more information.

9. During lottery scheduling each short job is assigned 5 tickets and each long job is given 2 tickets. How much CPU will be allocated to each short and long job is the system contains: (a) 5 short and 5 long jobs, (b) 1 short and 10 long jobs.

   Answer: (a) Total tickets = 5*5 + 5*2 = 25 + 10 = 35. Fraction of time allocated to each short job: 5/35 Fraction of time allocated to each long job: 2/35

   (b) Total tickets = 1*5 + 10*2 = 5 + 20 = 25. Fraction of time allocated to each short job: 5/25 Fraction of time allocated to each long job: 2/25

10. In ____ contention scope, all the threads in a single process are mapped to a single kernel thread.

    Answer: process contention scope

11. What is load balancing on multi-threaded systems? How does the OS perform load balancing?

    Answer: See slide 55. Load balancing is an attempt by the OS to keep the workload evenly distributed across all CPUs in a multi-core CPU system. The OS uses two algorithms to perform load balancing:
    (a) Push migration: an OS task checks periodically if a processor is lightly loaded, and if so then *pushes* a task (from a heavily loaded CPU) to this processor's ready queue.
    (b) Pull migration: Idle processor itself pulls task from others when its queue is empty during scheduling.

3 Chapter 8

1. Illustrate how to protect processes from one-another using the *base* and *limit* registers.
   Answer: Use the figure from slide 13. Compare each address generated with the base and limit registers.

2. Explain compiler time, load time, and execution time address binding. What type of binding is generally used in current OS?
   Answer: Slide 4. Execution time binding is most common.

3. Discuss the differences between logical and physical addresses and address spaces as they relate to an executing program.
   Answer: Slide 6. The MMU translates the logical address to physical address using the techniques of *segmentation* and/or *paging*.

4. Explain static and dynamic loading.
   Answer: Static loading – Entire program needs to be in memory and assigned addresses before program starts.
   Dynamic loading – routine is only loaded when called, as done in Java.

5. What is dynamic linking ?
   Answer: Mainly used for libraries.

   Static linking – all libraries included in static program image.
   Dynamic linking – library code not included in program image, only linked at runtime, if needed.

6. What is the need for swapping ?

7. What are the drawbacks of contiguous memory allocation ?
   Answer: External fragmentation. Any of the best-fit, first-fit, and worst-fit allocation policies causes this problem.

8. Explain, compare, and contrast the following resource allocation policies: (a) First Fit, (b) Best Fit, and (c) Worst Fit.

9. Compare and contrast internal and external fragmentation.

10. What is paging? How does it overcome the drawbacks of contiguous memory allocation ?

11. Assuming a page size of 2K and a 32-bit machine, how many bits are used for the page number, and how many are used for the offset within the page? Give the main reason why page sizes are usually powers of two.
    Answer: 21 bits for the page number, 11 bits for the offset within the page.

    Three reasons for page sizes being powers of 2 (you needed two):
    1. It simplifies the design of hardware because translation of a logical address into a logical and then physical page number and page offset requires only masking off sections of an address represented as a binary number. So the hardware can mask and shift rather than having to add, subtract and divide.
    2. It makes the hardware both cheaper and faster.
    3. Page sizes should be multiples (usually 1) of disk block size, and they are usually powers of 2 as well.

12. Consider a system with memory mapping done on a page basis, and using a single level page table. Assume that the necessary page table is always in memory.
    (a) If a memory reference takes 100 nanoseconds, how long does a memory mapped reference take if there is no TLB within the MMU to cache recently used page addresses?
    (b)Now we add an MMU which imposes an overhead of 15 nanoseconds on a hit or a miss. If we assume that 90 percent of all memory references hit in the MMU TLB, what is the Effective Memory Access time?
    Answer: (a) 100ns for accessing the page table + 100ns for accessing the data = 200ns.

    (b) There are two cases. In the first case, the TLB contains the required entry. Then we pay the 15 ns overhead on top of the 100 ns memory access time. In the second case, the TLB does not contain the item. Then we pay an additional 100 ns to get the required entry into the TLB, making 215 ns altogether. Hence, the EMAT is: (115 * 0.9) + (215 * 0.1) = 125 ns

13. When is a simple page table structure ineffective ? Explain the properties of (a) hierarchical page tables, (b) hashed page table, and (c) inverted page tables.

14. How is segmentation different from the paging memory model ? Illustrate the segmentation architecture.

15. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?
    Answer:

    a. First-fit: 212K is put in 500K partition, 417K is put in 600K partition, 112K is put in 288K partition (new partition 288K = 500K 212K), 426K must wait.

b. Best-fit: 212K is put in 300K partition, 417K is put in 500K partition, 112K is put in 200K partition, 426K is put in 600K partition.

c. Worst-fit: 212K is put in 600K partition, 417K is put in 500K partition, 112K is put in 388K partition, 426K must wait.

In this example, best-fit turns out to be the best.

16. Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers): a. 2375, b. 19366, c. 30000, d. 256, e. 16385

Answer: Since page size is 1 KB, we have 10 bits of offset. The remaining bits are the page numbers. So, a. 2375 = 0x947 = (1001 0100 0111)binary.
Then, offset = lower 10 bits = (01 0100 0111) = 0x147 = 327 decimal.
Page number = remaining high-order bits = (10) = 2 decimal/hex.
So, results are:
a. page = 2; offset = 327
b. page = 18; offset = 934
c. page = 29; offset = 304
d. page = 0; offset = 256
e. page = 16; offset = 1

17. Consider a computer system with a 32-bit logical address and 4 KB page size. The system supports up to 512 MB of physical memory. How many entries are there in:
a. A conventional single-level page table?
b. An inverted page table?

Answer: a. 4kb page size – 12 bits. Therefore, 20 bits for page table. Page table contains 2^{20} entries.

b. 512 MB of physical memory requires 29 bits. Frame size = page size = 4kb. So, page offset is still 12 bits. So, number of physical frames = $2^{29-12} = 2^{17}$.

18. Consider the following segment table: What are the physical addresses for the following

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

logical addresses? a. 0,430, b. 1,10, c. 2,500, d. 3,400, e. 4,112

Answer: a. 219 + 430 = 649
b. 2300 + 10 = 2310
c. illegal reference, trap to operating system
d. 1327 + 400 = 1727
e. illegal reference, trap to operating system

4 Chapter 9

1. Explain the need for Virtual Memory. Do you know any modern OS that use virtual memory concepts?
   Answer: slide 2. Most modern, including Linux and Windows, use virtual memory.

2. Explain demand paging.

3. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.
   Answer: Page faults occur if a legal process address resides on a page not currently present in any physical frame in memory. The page then needs to be brought into memory (from disk). Slides 9-10 for actions taken by OS on page fault.

4. When would the OS invoke a page replacement algorithm ? Explain basic page replacement.
   Answer: On a page fault, if all physical memory frames are occupied, see slides 15-16.

5. Assume that you have a page-reference string for a process with M frames (initially all empty). The page-reference string has length P; N distinct page numbers occur in it. Answer these questions for any page replacement algorithm:
   (a) What is a lower bound on the number of page faults?
   (b) What is an upper bound on the number of page faults?

   Answer: Lower Bound: N. The reference string indicates that the program actually references N distinct pages, and the best that any page replacement algorithm can do is make the first page fault for a page the last.

   Upper Bound: P. The worst possible situation is one where the working set is so small, and the page replacement algorithm so inept, that every reference to a new page in the reference string is a page fault.

6. What is the Belady's anomaly? Which page replacements suffer from it, and which are immune?

   Answer: Slide 22. FIFO, and may be counting-based algorithms suffer from it.

7. Explain two LRU implementation algorithms (counter and stack implementation).

8. How can *victim frames* optimize page fault handling ?

   Answer: Slide 30.

9. Should there be a minimum and/or maximum limit on the number of frames allocated to each process ? Explain.

10. Distinguish between: (a) Equal and proportional allocation, (b) Global and local page replacement algorithms.

11. What is thrashing? How may it be caused?

12. Consider a demand-paged computer system where the degree of multi-programming is currently fixed at four. The system was recently measured to determine utilization of CPU and the paging disk. The results are one of the following alternatives. For each case, what is happening? Can you increase the degree of multiprogramming to increase the CPU utilization?
    (a) CPU utilization, 13 percent; disk utilization, 97 percent

(b) CPU utilization, 87 percent; disk utilization, 3 percent
(c) CPU utilization, 13 percent; disk utilization, 3 percent.

Answer: 1. CPU utilization, 13 percent; disk utilization, 97 percent. The CPU utilization is low, yet the disk utilization is very high. This indicates that either all the processes are primarily I/O bound jobs, or the system is thrashing. Increasing the degree of multiprogramming will only serve to make thrashing even worse, since the paging is not helping in improving performance. However, increasing multiprogramming with some CPU-bound jobs might help if the high disk utilization is because of I/O-bound jobs. Of course, it is hard to know if a process is CPU-bound or not until its behavior demonstrates the fact. This illustrates the important point that in this situation we would have to know why the disk utilization was high; application-level I/O or page fault I/O.

2. CPU utilization, 87 percent; disk utilization, 3 percent. The CPU utilization is high, and the disk utilization is low. This indicates that the system is operating normally. The degree of multiprogramming is about right. It might be possible to increase it slightly, but that also runs the risk of increasing it too much, thereby engendering a thrashing situation. Paging is helping to improve performance in this case.

3. CPU utilization, 13 percent; disk utilization, 3 percent. Both CPU and disk utilization are low. The few processes running on the system are receiving all the service they want. However, the low CPU utilization indicates that the degree of multiprogramming could be increased to increase the CPU utilization. Paging is not helping system performance, but it is not hurting it either, for the simple reason that there is almost no paging going on. Simply stated, the system does not have enough to do!

13. What are the benefits of the *slab allocation* policy over the buddy allocator.

    answer: Slide 45.

14. Explain some issues in deciding the page *size* to use.

    answer: Slide 47 OR slide 23 in Chapter 8.

15. List costs and benefits of using virtual memory.

    Answer: Costs: If the system is already using discontiguous address spaces via page tables, then VM only provides an additional ability to start process execution without the entire process in memory all at one time. The cost is page faults happening during the execution of a process.

    VM gives OS an ability to bring multiple partial programs into memory at the same time. However, a high degree of multiprogramming can lead to thrashing, or to a large amount of time spent handling page faults. In this case, VM costs can exceed benefits. The only way to resolve this is to increase physical memory, or/and to reduce the degree of multiprogramming.

16. A certain computer provides its users with a virtual-memory space of 2^{32} bytes. The computer has 2^{18} bytes of physical memory. The virtual memory is implemented by paging, and the page size is 4096 bytes. A user process generates the virtual address 11123456. Explain how the system establishes the corresponding physical location. Distinguish between software and hardware operations.

    Answer: The virtual address in binary form is:

    0001 0001 0001 0010 0011 0100 0101 0110

    Since the page size is 2^{12}, the page table size is 2^{20}. Therefore, the low order 12 bits "0100 0101 0110" are used as the displacement into the page (offset), while the remaining 20 bits

"0001 0001 0001 0010 0011" are used as the index in the page table. These binary bits represent the decimal number "69923" (on my calculator). The page-table entry at index 69923 will give us the 6-bit physical frame number. We combine this physical frame number with the offset to get the physical address. (Ignore that part about "Distinguishing between software and hardware operations".)

17. Table 9.1 is a page table for a system with 12-bit virtual and physical addresses with 256-byte pages. The list of free page frames is D, E, F (that is, D is at the head of the list, E is second, and F is last.)

Page	Page Frame
0	-
1	2
2	C
3	A
4	
5	4
6	3
7	
8	B
9	0

Given the following virtual addresses, convert them to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. (A dash for a page frame indicates the page is not in memory.) (a) 9EF, (b) 111, (c) 700, (d) 0FF

Answer:

(a) 9EF - 0EF, (b) 111 - 211, (c) 700 - D00, (d) 0FF - EFF

18. What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

Answer: Thrashing is caused by underallocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

19. Assume there is an initial 1024 KB segment where memory is allocated using the Buddy system. Using Figure 9.27 as a guide, draw the tree illustrating how the following memory requests are allocated: (a) request 240 bytes (b) request 120 bytes (c) request 60 bytes (d) request 130 bytes

Next, modify the tree for the following releases of memory. Perform coalescing whenever possible: (a) release 240 bytes (b) release 60 bytes (c) release 120 bytes

Answer: The following allocation is made by the Buddy system: The 240-byte request is assigned a 256-byte segment. The 120-byte request is assigned a 128-byte segment, the 60-byte request is assigned a 64-byte segment and the 130-byte request is assigned a 256-byte segment. After the allocation, the following segment sizes are available: 64-bytes, 256-bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

After the releases of memory, the only segment in use would be a 256-byte segment containing 130 bytes of data. The following segments will be free: 256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, and 512K.

5 Chapter 10

1. The OS file system interface is said to provide an abstraction over reality. Explain.

   Answer: Slide 2.

2. What is a *raw* partition? When might it be useful?

   Answer: Raw partition is one without a file system. Useful when storing very uniform and application-specific data structures, like database entries, of swapped pages in a swap drive.

3. What are the advantages and disadvantages of : (a) Single-level directory, (b) Tree structured directory.

   Answer: Slides 6-11.

4. How do acyclic graph directory structure overcome the limitation of tree structured directory structure ?

   Answer: Acyclic graph directory allows sharing of files, via hard and soft links.

5. Explain soft and hard links, as used in Unix.

6. Explain in brief the file protection system maintained in Linux.

   Answer: Linux employs the generic read/write/execute permissions for (user,group,universe). Linux also uses Access Control Lists (ACL) to enforce finer-grained protections on files.

7. Could you simulate a multilevel directory structure with a single-level directory structure in which arbitrary long names can be used? How? Would your answer change if the file names were limited to seven characters?

   Answer: Yes, we can simulate given arbitrary large file name. Each directory name can essentially be appended to the primary file name. However, this scheme will fail if the length of the filenames is restricted.

8. Consider a file system where a file can be deleted and its disk space reclaimed while links to that file still exist. What problems may occur if a new file is created in the same storage area or with the same absolute path name? How can these problems be avoided?

   Answer: Let F1 be the old file and F2 be the new file. A user wishing to access F1 through an existing link will actually access F2. Note that the access protection for file F1 is used rather than the one associated with F2. This problem can be avoided by insuring that all links to a deleted file are deleted also. This can be accomplished in several ways:

   (a) maintain a list of all links to a file, removing each of them when the file is deleted,
   (b) retain the links, removing them when an attempt is made to access a deleted file,
   (c) maintain a file reference list (or counter), deleting the file only after all links or references to that file have been deleted.