CS 219 Spring 2024

Mid-semester Examination: SOLUTIONS

Name	
Roll Number	

Question no.	Marks obtained	Question no.	Marks obtained
1–8		9–13	
14–18		19–22	
23–25		26–27	
28–30		**Total**	

Please read the following instructions carefully before you start.

- The paper has **30 questions**, for a total of **50 marks**, accounting for **30%** of your grade.
- You must write your answers in this question booklet itself and turn it in. You need not turn in any rough sheets you may use.
- Please write your answers **legibly**, with all relevant calculations and explanations, in the space provided.
- Questions 1–20 carry carry 1 mark each. Questions 21–30 carry 3 marks each.
- For questions involving numerical answers, you must provide some calculation or explanation, and not just the final answer.
- Please avoid asking for clarifications, unless you think there is a mistake in the question itself.

1. Consider a page in the virtual address space of a newly created process, corresponding to the compiled code from the executable. The process has not yet begun execution and is ready to be scheduled. The OS is using demand paging, and has not yet allocated physical memory for the code page. What are the values of the valid and present bits in the page table entry of this page in the page table of the process?

   Ans: Valid = 1, Present = 0

2. Consider a system with 32-bit addresses and 4KB pages. A process has a 4KB page allocated for its stack, but its current stack frames occupy only the bottom 2KB of the stack. The stack has recently been accessed and has not been swapped out. Consider a virtual address in the top 1KB of the unused portion of the stack. What are the values of the valid and present bits in the page table entry corresponding to this virtual address in the page table of the process?

   Ans: Valid = 1, Present = 1

3. There are many ways of classifying the pages of a process, e.g., file-backed or anonymous, new empty page or a previously used page with content. Using the terminology in this classification, give an example of a type of page for which the OS does NOT have to perform any disk I/O when a page fault occurs, i.e., when the MMU traps to the OS because the accessed page is not in DRAM. You may assume that the OS has enough free frames and does not have to evict a victim page.

   Ans: Newly created empty anonymous page, or copy-on-write pages

4. Repeat the previous question, but now, give an example of a type of page for which the OS has to perform some disk I/O to service the page fault.

   Ans: File-backed pages, or previously used pages that store content

5. Give an example of a page access where the MMU traps to the OS on the page access even when the page is both valid and present in DRAM.

   Ans: Write access to a read only page, or access to a kernel page from user mode, or any other type of illegal access

6. A user finds that an application running on the system has slowed down significantly due to a bug in the program. The user finds that the CPU utilization is very low. The utilization of main memory is very high, at over 95%. The disk I/O activity is also very high. Give an example of a programming error committed by the developer that can cause this situation.

   Ans: Many correct answers are possible including not freeing up malloc memory, not clearing up zombie processes, having a very large working set size that does not fit in RAM. Note that we have NOT given marks if you simply say that a large number of page faults are occurring. The questions asks you to give possible reasons for the page faults.

7. Which one of the following actions does NOT help in improving system performance when the system is thrashing? Select the correct answer.

   (a) Increase size of DRAM

   (b) Increase size of swap space

   (c) Terminate some processes

   (d) Evict some pages from active processes to swap space

Ans: (b)

8. Consider a two-level page table in a 32-bit system with 4KB pages. The entry at index i (assume index starts at 0) in the outer page directory array contains the physical address of the inner page table page containing page table mappings for virtual addresses in the range $[X, Y)$. What are the values of virtual addresses X and Y?

   Ans:

   $X = i \times 2^n$ and $Y = (i+1) \times 2^n$, where n is sum of the number of bits used to index into the inner page table and number of bits used as offset. For 32-bit architectures with 4KB pages and 4-byte page table entries, n is 22.

9. Which of the following statements is/are true about page table mappings for OS code and data in the page table of every process? Select all that apply.

   (a) The OS code is usually mapped at low virtual addresses.

   (b) The OS code is usually mapped at high virtual addresses.

   (c) There is only one copy of OS code in memory, that is mapped into all address spaces.

   (d) The OS code is mapped into the page tables of only kernel processes, and not user processes.

   Ans: (b), (c)

10. Consider a shared memory page segment that is mapped into the virtual address space of two processes P_1 and P_2. Let v_i denote the virtual page number of this page in the address space of P_i, and let f_i denote the physical frame number that the page maps to in the page table of P_i. Which of the following statements best describes the relationship between the two v_i and f_i?

    (a) v_1 is always equal to v_2, f_1 is always equal to f_2.

    (b) v_1 is always equal to v_2, f_1 can be different from f_2.

    (c) v_1 can be different from v_2, f_1 is always equal to f_2.

    (d) v_1 can be different from v_2, f_1 can be different from f_2.

    Ans: (c)

11. Which of the following is/are disadvantages of using virtual addressing over physical addressing in modern computer systems? Select all that apply.

    (a) Exposing user to complexities of non-contiguous memory allocation

    (b) Memory overhead for tracking address translation information

    (c) More internal fragmentation

    (d) More external fragmentation

    Ans: (b)

12. Assuming all else remains the same, which of the following changes are expected to happen when we increase the page size in a computer system? Select all that apply.

(a) Size of page table increases

(b) Hit rate of TLB increases

(c) External fragmentation increases

(d) Internal fragmentation increases

Ans: (b), (d)

13. For which of the following events does the OS update the address translation information in the MMU, e.g., update page table pointer in the MMU, flush invalid page table entries in the TLB? You may assume that the TLB caches page table entries of the currently running process only. Select all that apply.

(a) Context switch from one process to another

(b) A running process moves from user mode to kernel mode to handle a trap

(c) Some page table entries in the page table of the currently running process have changed

(d) Some page table entries in the page table of a blocked/sleeping process have changed

Ans: (a), (c)

14. Consider an alternate model of maintaining address translation information in a system using paging, called the inverted page table. In this method, there is only one page table maintained across all processes. The page table has one entry for every physical frame in the system. The i-th entry in the inverted page table contains information about which page of which process (i.e., PID and virtual page number) is stored in the i-th physical frame in memory, with a special null entry indicating that the frame is empty. Which of the following is/are advantages of using an inverted page table as compared to a regular page table? Select all that apply.

(a) Lesser memory required to store page tables in the system

(b) Easier to look up the physical frame number corresponding to a virtual page number

(c) Easier to ensure isolation and protection between processes

(d) Lesser external fragmentation

Ans: (a)

15. Which of the following actions is/are performed by the MMU? Select all that apply.

(a) Allocate memory for inner and outer page tables of a process upon process creation

(b) Update dirty and accessed bits suitably on memory accesses

(c) Translate a virtual address to a physical address on every memory access

(d) Raise a trap when address translation fails

Ans: (b), (c), (d)

16. Which of the following is/are expected advantages of segmentation over paging for memory management? Select all that apply.

(a) Lesser external fragmentation

(b) Lesser internal fragmentation

(c) Lesser overhead of metadata needed for address translation

(d) Better protection and isolation of processes from one another

Ans: (b), (c)

17. Which of the following IPC actions are expected to block the invoking process (by default)? Select all that apply.

    (a) Reading data from a shared memory segment, when no other process has written to it

    (b) Reading data from pipe, when no other process has written into the write end of the pipe

    (c) Calling accept on a connected socket, when no other process has invoked connect on it

    (d) Writing into a shared memory segment, when no other process has attached the segment yet

    Ans: (b), (c)

18. Consider a CPU scheduler that schedules processes only based on priority. Every process is assigned a priority, either by the user, or by the OS using some heuristics. The scheduler maintains the list of ready processes in a sorted order, sorted by priority. Whenever the CPU is free to run a process, the scheduler picks the highest priority process in the list of ready processes, and runs it till it voluntarily gives up the CPU. Does this scheduling policy ensure that every process in the system completes its execution within a finite amount of time? Answer yes/no, with suitable justification.

    Ans: Many correct answers are possible. A higher priority process may run in an infinite loop and not yield the CPU to a lower priority process. More interestingly, there can be a stream of higher priority processes coming in, and a lower priority process may never get a chance to execute. However, simply saying a higher priority process runs for a long (but finite) time is not enough, as it does not automatically imply that a lower priority process won't get to run in a finite time.

19. Consider the `struct context` and trap frame data structures that are pushed on to the kernel stack of processes in xv6. Let EIP_C and EIP_T denote the EIP value saved in these structures. Briefly describe the EIP values stored in these structures for a process that has been context switched out because of a blocking system call. If the structure is not present on the kernel stack, you can put a value of 0 for the corresponding EIP.

    Ans:

    EIP_C = EIP value where process stopped in kernel mode, EIP_T = EIP value where process made the system call in user mode

20. Repeat the previous question for a process that has just completed execution of the `getpid()` system call in kernel mode, and is about to return from the trap to user mode.

    Ans:

    EIP_C = 0 (no context structure is present), EIP_T = EIP value where process made the system call in user mode

21. Consider the following pseudocode where a parent process P creates two child processes C1 and C2. All three processes invoke functions to perform some tasks. We would like to ensure that the tasks run in the order of `task0()`, `task1()`, `task2()`, no matter what order the processes are scheduled in by the scheduler. You cannot make any assumptions on the execution times of these tasks, so you cannot put sleep statements to control the order of execution. What changes will you make to the code below to ensure this order of execution of the tasks? You can use the space next to the question to write your solution *briefly* in 2–3 sentences.

```
int rc1 = fork();
if(rc1 == 0) {
  task1(); //child C1
}
else {
  int rc2 = fork();
  if(rc2 == 0) {
    task2(); //child C2
  }
  else {
  task0(); //parent P
  }
}
```

Ans: Many solutions are possible. The processes can use some kind of IPC (pipes, sockets, signals) to indicate to each other when a task has finished. For example, P can write into a pipe after finishing its task, and C1 can block on the pipe read before starting its task. Similarly, C1 can write into a pipe after it finishes its task, and C2 can read from the pipe before running its task.

NOTE: Some common wrong answers which were not given any credit are also listed here. You cannot move around tasks, as that defeats the purpose of the question, which was to control the order of execution of the processes. A child process also cannot call wait on the parent. The wait system call is only to reap child processes. It is also not possible to use boolean variables and such to indicate completion of tasks, because every process has its own copy of the variable, and changes to the variable in one process will not be seen by the others. Finally, some of you have written multiple solutions, some of which were correct and some incorrect. We have not awarded marks in such cases as well, as we cannot do the job of choosing the right answers.

22. Suppose we would like to compute the average duration between when a process is created, and when it performs the first write to some variable in the user data. To do this, we modify the OS to maintain two variables in the PCB of a process: a timestamp t_0 when the process is first created, and timestamp t_w when the process writes to a variable in the user data for the first time. The timestamp t_0 is set when the process is created during fork. Describe one way by which the OS can detect the time of the first write so that it can set timestamp t_w suitably. You cannot make any assumptions on hardware providing any new features, and your solution must work on existing hardware. You can assume that you can change the page tables and page fault handler of the process. Your solution should be realistic enough to be implemented in real systems, without imposing excessive overheads. Describe your answer *briefly* in 2–3 sentences.

Ans: The OS marks the pages of the process as read-only. The MMU traps to the OS on the first write, the OS can record the timestamp, and reset this bit for the future. Another less efficient solution (which was awarded partial marks) is to mark the pages as not present. In this case also, the MMU traps to the OS, but such traps happen on both reads and writes, whereas we only want to record the time of the first write. A few other solutions got only partial marks, especially if the solution involved hardware changes (change in MMU behavior etc.) or the solution was not clearly explained.

23. Consider a process running on a system with 52-bit virtual addresses, 16KB pages, and 4-byte page table entries. The OS uses hierarchical paging. Assume 1K = 2^{10}.

    (a) Calculate the number of levels in the page table of the process.
    Ans: 4

    (b) Calculate the maximum number of pages required for the page table in the innermost level.
    Ans: Total entries in innermost level is 2^{38} which fits in 2^{26} pages in the innermost level.

    (c) Calculate the number of entries stored in the outermost page directory.
    Ans: 4

24. For the same question above, you are told that the OS allocates all pages, including page table pages, on demand. The process has accessed 8K unique pages during its execution, for which valid page table entries have to be maintained. You may assume that none of these pages has been swapped out yet.

    (a) What is the minimum possible number of pages in the page table of this process (across all levels) after all accesses have completed?
    Ans: 5 pages = 2 pages in the inner most level, plus one page each in the other levels.

    (b) Repeat the above question for the maximum possible number of pages in the page table.
    Ans: 2^{13} + 2^{13} + 4 + 1

25. Consider a system with 32-bit addresses, 4KB pages, and 4 byte page table entries. The OS uses a two-level page table, with the inner page table pages allocated on demand. In the outermost page directory of the page table of a process, you are told that the first entry (at index 0) is valid, and the second entry in invalid (null, or not pointing to any inner page table). Similarly, in the first page in the set of inner page table pages, you are told that the first and third page table entries are valid, and the second entry (at index 1) is invalid. For each of the virtual addresses given below, indicate whether an access to that address will result in a trap (due to an invalid entry) or not. You must answer yes/no for the trap, and show suitable calculations to justify your answer.

    (a) 3KB
    Ans: No trap, since the first page in both inner and outer page tables, which holds this address, are valid.

    (b) 9KB
    Ans: No trap. This address falls in the third entry of inner page table, which is valid.

    (c) 5MB
    Ans: Trap. This address falls in the second entry of outer page directory, which is invalid.

Note: Only 1 mark out of 3 has been awarded if correct answers are given without any explanation.

26. Consider an OS that implements approximate LRU in the following manner. Let us call the duration between timer interrupts as a "round'. When the timer interrupt goes off, the OS saves a copy of the accessed bits of all active pages for the past round, and resets all the accessed bits to 0 for the next round. In this manner, the OS knows which pages have been accessed in every round. When the OS has to evict a page, it computes the set of pages not accessed over the past K rounds, and picks one of these pages to evict. Shown below in the table are the values of the accessed bits of 8 pages (numbered 1–8) as recorded by the OS over the past 3 rounds. For each value of K below, list all the pages that will be considered for eviction by the OS using the history of the accessed bits from the last K rounds.

Page number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
At last timer (latest round)	0	0	1	0	1	0	1	0
At second last timer (second last round)	1	0	1	0	0	1	0	1
At third last timer (third last round)	1	1	0	0	0	0	1	1

(a) $K = 1$
Ans: 1,2,4,6,8.

(b) $K = 2$
Ans: 2,4

(c) $K = 3$
Ans: 4

Note: no partial marks for the above question.

27. The table below shows the arrival times and CPU bursts of 4 processes in a simple system. For the various scheduling policies given below, calculate the completion times for all 4 processes, assuming the scheduler runs processes on a single core. You must show which process ran for what time periods, so that we can understand how you calculated the completion times.

Process	**Arrival time**	**CPU burst**
P1	0	5
P2	1	4
P3	2	2
P4	3	1

(a) Non-preemptive Shortest Job First (SJF)
Ans: P1 runs, followed by P4, P3, P2. Completion times are P1: 5, P2: 12, P3: 8, P4: 6

(b) Preemptive Shortest Remaining Time First (SRTF) where ties between processes with equal remaining time are broken by giving higher preference to the currently running process
Ans: P1 runs for 2 units, then P3 for 2 units, then P4, P1, P2. Completion times are P1: 8, P2: 12, P3: 4, P4: 5.

(c) Preemptive SRTF where ties are broken by giving preference to the newly arrived process
Ans: P1 runs for 1 unit, then P2, P3, P4, followed by P3, P2, P1, with new arrivals preempting the running processes. Completion times are P1: 12, P2: 8, P3: 5, P4: 4.

Note: 1 mark has been deducted if only completion times are shown without any justification.

28. Consider the program shown below that prints the value of variable `a`, and a pointer address.

```
int a = 10;
if (fork() == 0) {
  a = a + 1;
  printf("Child1 a = %d, %p\n", a, &a);
}
else {
  a = a - 1;
  if(fork() == 0) {
    printf ("Child2 a = %d, %p\n", a, &a);
  }
  else {
    a = a + 2;
    printf ("Parent a = %d, %p\n", a, &a);
  }
}
```

(a) What are the values of variable `a` printed by the various processes?
Ans:
`Parent a = 11`, `Child1 a = 11`, `Child2 a = 9`

(b) Now, suppose we comment out the line `a = a - 1` in the code above. What are the values of the variable `a` printed by the program now?
Ans:
`Parent a = 12`, `Child1 a = 11`, `Child2 a = 10`

(c) Are the values of the pointer variable printed by all processes the same or different? Explain your answer.
Ans: Yes, all pointer addresses are virtual addresses, which are the same in the parent and child memory images.

29. Consider a process with N pages in its current working set, which it accesses in order, for a long time. That is, if the N pages are numbered 1, 2, ..., N, then the process accesses pages as follows: 1, 2, ... N, 1, 2, ..., N, 1, 2, ..., N, and so on. The OS allocates only $N-1$ physical frames for the process. For each of the page replacement policies below, calculate the fraction of page accesses that result in a page fault in the steady state (i.e., ignoring the compulsory misses at the beginning of execution).

(a) FIFO (First In First Out) policy
Ans: 1 (all accesses will result in page fault)

(b) LRU policy

Ans: 1 (all accesses will result in page fault)

(c) Optimal policy that looks into the future to minimize the number of page faults

Ans: $\frac{1}{N-1}$. Note that the answer $\frac{1}{N}$ is incorrect and has not been awarded marks.

30. Draw a complete picture of the kernel stack of a newly created process in xv6, after the process has been fully created and is waiting to be context switched in by the xv6 scheduler. Show the 2 different structures on the stack, and mention the values of the 3 different EIPs stored on the stack inside/outside these structures.

Ans: The kernel stack has context structure on the top (EIP is forkret), a return address (EIP) of trapret, and the trapframe structure on the bottom (EIP is pointing to user code). Partial marks have been awarded if the structures are correctly shown but EIP values are incorrect.