CS 537: Introduction to Operating Systems
Fall 2019: Midterm Exam #1
Solution Guide

This exam is closed book, closed notes.

All cell phones must be turned off and put away.

No calculators may be used.

You have two hours to complete this exam.

Write all of your answers on the accu-scan form with a #2 pencil.

These exam questions must be returned at the end of the exam, but we will not grade anything in this booklet.

You may separate the pages of this exam if it helps you.

Unless stated (or implied) otherwise, you should make the following assumptions:

- The OS manages a single uniprocessor (single core)
- All memory is byte addressable
- The terminology lg means $\log_2$
- 2^{10} bytes = 1KB
- 2^{20} bytes = 1MB
- Page table entries require 4 bytes
- Data is allocated with optimal alignment, starting at the beginning of a page
- Assume leading zeros can be removed from numbers (e.g., 0x06 == 0x6).
- Hex numbers are represented with a preceding "0x"

This exam has 92 questions. Each question is worth the same number of points.

Good luck!

Part 1: Virtualizing the CPU

Designate if the statement is True (a) or False (b).

1) One of the roles of the OS is to directly expose all details of a hardware resource to user-level applications.

False; the OS needs to provide an abstraction of the hardware to applications and may hide details to make different hardware appear identical to other hardware

2) A program can be composed of multiple processes.

True; Can call fork() multiple times within a program

3) Each process has its own virtualized stack pointer.

True; each process (and each thread even) sees its own value for the stack pointer

4) Different processes accessing the same virtual memory address may see different contents.

True; each process has its own virtual address space that is distinct from others

5) Whenever a process is descheduled by the scheduler, the process is moved to the BLOCKED state.

False; when a process is descheduled, it is moved to the READY state

6) System calls are handled by the OS in kernel mode.

True;

7) A context switch is the same as a time slice.

False; a context switch changes the process that is running; the time slice is the amount of time a process has to run in RR or a MLFQ

8) A running job may relinquish the CPU and become BLOCKED when it is waiting for the user to type at the terminal.

True; when a process is waiting for something to happen and there is no reason to schedule it, it is in the blocked state

9) With cooperative multi-tasking it is possible for a malicious user-level process to hold on to the CPU until the machine is rebooted.

True; with cooperative multi-tasking there is no preemption, so the only time the OS gets control is when a process blocks, makes a system call, or exits; if the process is spinning forever in a while() loop, the scheduler may never get control

10) On a timer interrupt, the OS is responsible for transitioning from user to kernel mode.

False; on an interrupt, the hardware performs the transition from user to kernel mode

11) Multiple processes may be in the READY state at the same time.

True; can have many processes that want to use the CPU

12) A reasonable performance goal for a scheduler is to minimize job throughput (i.e., the number of jobs completed per second).

False; would want to MAXIMIZE job throughput

13) If all jobs have the same length (i.e., the same cpu burst length), an FCFS and a SJF scheduler give the same average response time for the workload.

True; if all jobs have the same length, then SJF (which looks at job length) would default to FCFS to break ties.

14) SJF is provably optimal for preventing starvation.

False; SJF is provably optimal (for a non-preemptive scheduler) for minimizing turnaround time, but does suffer from starvation (long jobs may never be scheduled if short jobs keep arriving)

15) A non-preemptive scheduler will only schedule a new job when the RUNNING job exits or becomes BLOCKED.

True; non-preemptive schedulers never deschedule a running process

16) Increasing the time-slice of an RR scheduler is likely to decrease the overhead imposed by scheduling.

True; there will be fewer context switches for the same workload

17) Increasing the time-slice of an RR scheduler makes the scheduler behave more like FCFS.

True; once the time-slice is as long as the job length, it is equivalent to FCFS.

18) Increasing the time-slice of an RR scheduler makes the scheduler behave more like STCF.

False, see above.

19) An MLFQ scheduler assumes the presence of an oracle that knows the length of a job's CPU burst in advance.

False; MLFQ changes the priority of a process after it has begun to run as a way of dynamically adapting to the the length of the CPU bursts that it observes

20) An MLFQ scheduler can preempt running jobs.

True; for example, RR is performed at the same priority.

Part 2: Virtualizing Memory

Designate if the statement is True (a) or False (b).

21) The fork() system call creates a child process whose execution begins at the entry point main().

False; fork() starts the child process at the return point of this call to fork()

22) A process can close() the stdout file descriptor.

True; closing the stdout file descriptor and then opening a new file is exactly how stdout redirection is implemented.

23) The address space of a process contains all of physical memory.

False; address spaces are associated with virtual memory

24) The heap is more likely to suffer from fragmentation than the stack.

True; No fragmentation in a stack since a simple pointer separates allocated from free space

25) Local variables within a procedure are allocated on the heap.

False; allocated on the stack.

26) With dynamic relocation, the OS determines where the address space of a process is allocated in virtual memory.

False; the OS allocates the address space in physical memory; the compiler/linker chose the actual virtual addresses to lay out code and data.

27) With dynamic relocation, the OS can move an address space after it has been placed in physical memory.

True; this is the idea of dynamic relocation (e.g., base+bounds)

28) A Gantt chart illustrates how virtual pages are mapped to physical pages.

False; shows timeline of job scheduling

29) An MMU translates logical addresses generated by the OS running in kernel mode to physical addresses.

False; OS can access all of physical memory directly with no translation.

30) Threads running within the same process are likely to share the same value for the base register, but not the bounds register, in the MMU.

False; threads share exact same address space so share both base and bounds.

31) With pure segmentation (with no paging or TLBs), each segment of each process must be allocated contiguously in physical memory.

True; that is how segments work

32) With pure segmentation (with no paging or TLBs), fetching and executing an instruction that performs a store from a register to memory will involve exactly two memory references.

True; segmentation does not add any extra memory references for looking up address translations; so 1 memory access to fetch the instruction + 1 access for the store operation.

33) The number of bits used to specify a virtual page in a virtual address can be different than the number of bits used to specify a physical page in a physical address.

True; there can be a different number of virtual pages than physical pages.

34) If 11 bits are used in a virtual address to designate an offset within a page, each page must be **2 KB**.

True; 2^11 = 2048 = 2K (byte addressable, so 2KB)

35) If a physical address is 16 bits and each page is 1KB, then the top 10 bits exactly designate the physical page number.

False; 1KB page requires 10 bits for offset; 16-10 = 6 bits for the ppn.

36) If a virtual address is 12 bits and each page is 512 bytes, then each address space can contain up to 16 virtual pages.

False; 512 bytes = 2^9 → 9 bits. 12 – 9 = 3 bits for vpn. 2^3 = 8 (not 16).

37) Given a constant number of bits in a virtual address, the size of a linear page table increases with more (virtual) pages.

True; a PTE is needed for each virtual page, so the size of the table increases with more pages.

38) Given a fixed number of (virtual) pages, the size of a linear page table decreases with a smaller address space.

False; fixed number of virtual pages → same number of entries → size for page table.

39) Given a 22-bit virtual address and 1KB pages, each linear page table will consume 8KB.

False; 22 – 10 = 12 vpns. 2^12 ptes * 4 bytes/pte = 2^14 bytes = 16KB.

40) Compared to pure base+bounds, a linear page table doubles the number of memory references (assuming no TLB).

True; base+bounds needs no extra memory references; linear page tables need to do one page table lookup for every memory reference.

41) A linear page table must be stored contiguously in physical memory.

True; that is the problem with linear page tables

42) For faster translations, the OS looks up page mappings from VPN to PPN in the TLB.

False; hardware does this…

43) A workload that sequentially accesses each of the elements of a large array once will often require the same VPN to PPN translation.

True; the successive elements of the array will reside on the same page which need the same VPN to PPN translation (leading to good TLB hit rates).

44) If the TLB does not support ASIDs, all page table entries (PTEs) are marked invalid on a context switch.

False; the entries of the TLB would be marked invalid, but not the page table entries (if the PTEs were marked invalid, that would mean that vpn is not part of the process's address space and there is no mapping between that vpn and a ppn).

45) With ASIDs, the performance penalty as seen by process A for a context switch from process A, to process B, and back to process A, could decrease when B performs fewer memory operations.

True; if B performs fewer memory accesses, B will replace fewer TLB entries, and thus A will retain more of its entries across the context switch.

46) A TLB miss must be handled by the OS.

False; can be hardware or software

47) With a linear page table, all virtual pages within an address space must be allocated.

False; while the PTEs have to be allocated, the actual pages do not; they can be invalid (not mapped).

48) A multi-level page table typically reduces the amount of memory needed to store page tables, compared to a linear page table.

True; portions of the address space that are not allocated do not need corresponding (inner) page tables.

49) Given a 2-level page table (and no TLB), exactly 3 memory accesses are needed to fetch and execute each instruction.

False; Fetching the instruction will require 3 memory access (2 in the page table, 1 for the actual fetch), but EXECUTING the instruction if it is a load or store will require more…

50) In the memory hierarchy, a backing store is usually larger and faster than the memory layer above that uses that backing store.

False; usually larger and slower as one moves down the memory/storage hierarchy.

51) Given a fixed-size address space, increasing the size of a page generally increases the number of levels that are needed in a multi-level page table.

False; to start, with a larger page, there are fewer vpns and thus fewer PTEs are needed. Further, with a larger page, more PTEs will fit within a page.

52) The size of a PTE is equal to the number of bits needed to select an offset within a page.

False; the size of PTE isn't really related to the size of a page

53) When the dirty bit is set in a PTE, accessing the associated page will cause a segmentation fault.

False; dirty bit just means copy of page n memory does not match copy on backing store and those contents must be written out to the backing store if/when that page is replaced.

54) When the present bit is cleared in a PTE, accessing the associated page will cause a segmentation fault.

False; if not present, this is a page fault.

55) On a TLB miss, the process is moved to the BLOCKED state and another process may be scheduled.

False; TLB misses are handled relatively quickly (maybe even in hardware), so no reason to switch to a different process.

56) OPT always performs better than LRU.

False; better or the same…

57) LRU with N+1 pages of memory always performs as well or better than LRU with N pages.

True; guaranteed that N+1 pages has the same contents as N pages, plus one more

58) FIFO with N+1 physical pages will contain (cache) the same virtual pages as FIFO with N pages, plus the contents of one more virtual page.

False; this is Belady's algorithm and why performance can be worse with N+1 pages

59) Demand paging loads the address space of a process into physical memory when that process is started.

False; the address space isn't loaded when the process is started; demand paging waits until each page is actually accessed

60) When the hand used by the clock algorithm is moving quickly, the same pages in physical memory are being repeatedly accessed.

False; if moving quickly, it means the clock hand keeps finding that the use bit is set on different pages, so many different pages are being accessed

61) The clock algorithm must be able to determine if a page has been written to in some particular time interval

False; it is all about accesses, not writes.

62) The goal of the clock algorithm is to approximate the behavior of a FIFO replacement algorithm.

False; approximate LRU.

63) Increasing the number of jobs simultaneously submitted to a system may decrease the throughput of that system.

True; if the jobs use more physical memory than available, system will thrash and throughput decreases.

Part 3. CPU Job Scheduling

If needed, assume a time-slice of 1 sec. If needed to break ties, give preference to jobs in the order listed (A, B, C).
Assume a workload with the following characteristics:

Job Name	Arrival Time (seconds)	CPU Burst Time (seconds)
A	0	4
B	0	2
C	0	3

64) Given an RR scheduler, what is the average response time for the 3 jobs?
- a. 1 second
- b. 2 seconds
- c. 3.33 seconds
- d. 10 seconds
- e. None of the above

A starts at 0, B at 1, C at 2; Average to get them started is 1.

65) Given an RR scheduler, what is the average wait time for the 3 jobs?
- a. 2 seconds
- b. 3.33 seconds
- c. 4.33 seconds
- d. 6.67 seconds
- e. None of the above

Schedule: ABCABCACA
B wait time: 3 (time not scheduled between arrival and completion)
A wait time: 5
C: wait time: 5
Ave: 13/3 = 4.33

66) Given an RR scheduler, what is the turnaround time for job B?
- a. 1 second
- b. 2 seconds
- c. 4 seconds
- d. 5 seconds
- e. None of the above

B finishes at time 5

67) Given a FIFO scheduler, what is the average turnaround time for the 3 jobs?
- a. 6.33 seconds
- b. 6.67 seconds
- c. 8.67 seconds
- d. 9 seconds
- e. None of the above

AAAABBCCC
4 + 6 + 9 = 19/3

68) Given a SJF scheduler, what is the average turnaround time for the 3 jobs?
- a. 3.33 seconds
- b. 5.33 seconds
- c. 6.33 seconds
- d. 6.67 seconds
- e. None of the above

BBCCCAAAA
2 + 5 + 9 = 16/3

Assume a new workload with the following characteristics:

Job Name	Arrival Time (seconds)	CPU Burst Time (seconds)
A	0	7
B	4	4
C	6	2

69) Given a SJF scheduler, what is the average turnaround time for the 3 jobs?

a. 6.33s
b. 6.67s
c. 7.33s
d. 9.67s
e. None of the above

No preemption
AAAAAAACCBBBB
7+(9-6)+(13-4)=7+3+9 → 19/3
Remember not to count the time before a job arrived in its turnaround time…

70) Given a STCF scheduler, what is the average turnaround time for the 3 jobs?

a. 6.33s
b. 6.67s
c. 7.33s
d. 9.67s
e. None of the above

At time 4 when B arrives, A has only 3 units left, so A stays scheduled
At time 6 when C arrives, A has only 1 unit left, so A stays
So, with no preemption, ends up being the same as SJF…

Part 4. Reverse Engineering for Dynamic Relocation

Assume you have an architecture with a 64KB address spaces and 128KB of physical memory. Assume you are performing dynamic relocation with a base-and-bounds register. You collect a trace of virtual address to physical address translations and see the following results:

```
VA  0: 0x6baa (decimal: 27562) --> VALID: 0x1efbc (decimal: 126908)
VA  1: 0x4248 (decimal: 16968) --> VALID: 0x1c65a (decimal: 116314)
VA  2: 0x82e2 (decimal: 33506) --> SEGMENTATION VIOLATION
VA  3: 0x67a9 (decimal: 26537) --> VALID: 0x1ebbb (decimal: 125883)
VA  4: 0xc8a7 (decimal: 51367) --> SEGMENTATION VIOLATION
VA  5: 0x2568 (decimal: 9576 ) --> VALID: 0x2568 (decimal: 9576 )
```

What can you infer about state of the memory system? Assume only a single user process or the OS is running throughout the trace.

71) The base register contains

- a. 0x2434 (decimal 9268)
- b. 0x18412 (decimal 99346)
- c. 0x25b66 (decimal 154470)
- d. Contents cannot be determined
- e. None of the above

Virtual address + base = physical
So, base = physical - virtual
Decimal is easier for this, so: 126908 – 27562 = 99346

72) The OS was running when which virtual address was traced?

- a. VA 0
- b. VA 1
- c. VA 3
- d. VA 5
- e. None of the above or multiple of the above

VA = PA for item VA 5

73) Assuming the user process is running, virtual address 0x4826 (decimal: 18470) translates to what physical address?

- a. 0x84a6 (decimal: 33958)
- b. 0x1cc38 (decimal: 117816)
- c. Answer cannot be determined from information provided
- d. Segmentation Fault
- e. None of the above

18470 < other valid translation (e.g., 27562), so must be < bounds and in valid portion of address space
Thus 18470 + 99346 = 117816

74) Assuming the user process is running, virtual address 0x9558 (decimal: 38232) translates to what physical address?

- a. 0x2196a (decimal: 137578)
- b. 0x1f220 (decimal: 127520)
- c. Answer cannot be determined from information provided
- d. Segmentation Fault
- e. None of the above

38232 > another VA that caused a seg fault (33506), so must be > bounds and not valid

75) Assuming the user process is running, virtual address 0x739a (decimal: 29594) translates to what physical address?

   a. 0x1f7ac (decimal: 128940)
   b. 0x1fce2 (decimal: 130274)
   c. Answer cannot be determined from information provided
   d. Segmentation Fault
   e. None of the above

29594 < address that caused seg fault (e.g., 33506) but > all observed addresses that were valid (e.g., 27562), so can’t tell if this VA is valid or not

Part 5. Linear Page Tables

What is the size of a linear page table for one process given the following assumptions? Remember, unless otherwise specified, assume PTEs are 4 bytes each.

76) Bits in virtual address: 32, Page Size: 4KB
   a. 1MB
   b. 4MB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

Page size of 4KB → Offset = 12 bits; 32 – 12 = 20 bits. 2^20 PTEs * 2^2 bytes/PTE= 4 MB

77) Bits in virtual address: 32, Page Size: 4KB, PTE size: 8 bytes
   a. 2MB
   b. 8MB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

Same as above except 2^20 PTEs * 2^3 bytes/PTE= 8 MB

78) Bits in virtual address: 20, Page Size: 512 bytes
   a. 8KB
   b. 16KB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

512 bytes → Offset = 9 bits; 20 – 9 bits = 11; 2^11*2^2 = 2^13 = 8KB

79) Bits in virtual address: 10, Page Size: 4KB
   a. 4KB
   b. 4MB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

4KB → 12 bits for offset, can't have only 10 bits for entire virtual address…

80) Bits in virtual address: 12, Number of Virtual Pages: 128, Page Size: 1KB
   a. 64KB
   b. 512KB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

1KB → 10 bits for offset; 128 pages → 6 bits for vpn; 16 bits > 12 bits for va → not possible

81) Number of Virtual Pages: 4096, PTE size: 16 bytes
   a. 64KB
   b. 128KB
   c. Situation not possible
   d. Not enough information to determine page table size
   e. None of the above

Page size = 4096 * 16 bytes = 2^12 * 2^4 = 2^16 = 64KB

82) Bits in virtual address: 24, Number of Physical Pages: 1024
   a. 4KB
   b. 16KB
   c. Situation not possible

d. Not enough information to determine page table size
e. None of the above

Since the number of physical pages does not need to equal the number of virtual pages, we don't have enough info to know the number of vpns

Part 6. Multi-level Page Tables

Same setup as homework. Assume dynamic relocation is performed with a **two-level page table** with no TLB. Assume the page size is 32 bytes, the virtual address space for the current process is 1024 pages, or 32 KB, and physical memory consists of 128 pages. A virtual address needs 15 bits (5 for the offset, 10 for the VPN) and a physical address 12 bits (5 offset, 7 for the PFN). The upper five bits of a virtual address index into a page directory; the page directory entry (PDE), if valid, points to a page of the page table. Each page table holds 32 page-table entries (PTEs). Each PTE, if valid, holds the desired translation (physical frame number, or PFN) for the virtual page. The format of an 8-bit PTE is VALID | PFN6 ... PFN0. Contents of memory are as follows:

```
hex offset   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
dec offset   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

0x 0 (  0): 11 0d 16 1b 07 13 04 0d 0a 19 1c 1a 10 16 14 10 04 08 1d 12 03 0b 04 03 0b 09 00 0f 1c 0e 0c 0b
0x 1 (  1): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x 2 (  2): 1b 0c 1e 19 12 0d 11 1a 0a 04 1b 13 0b 0c 03 11 18 0f 15 16 0a 13 1e 16 05 14 13 18 15 1c 1e 06
0x 3 (  3): 05 03 14 0a 1b 07 1d 09 12 1c 15 1c 15 01 1b 12 09 05 1a 12 1d 17 1d 11 15 12 08 1d 09 18 1d 14
0x 4 (  4): 0e 0b 0b 17 00 08 1d 0e 00 13 14 1c 1b 1d 08 09 15 0a 0b 01 06 04 0b 15 06 08 02 0c 12 12 14 10
0x 5 (  5): 03 0d 13 0d 05 0f 06 0f 16 09 0d 13 10 18 1b 1d 05 0e 03 15 0c 1a 0e 18 00 13 0c 19 03 02 15 0a
0x 6 (  6): 06 06 10 01 02 02 16 1e 17 0f 04 18 04 02 0f 1a 09 1a 0a 1d 16 1a 1e 19 06 0f 10 16 07 11 08 05
0x 7 (  7): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d2 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x 8 (  8): 7f 7f d8 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x 9 (  9): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x a ( 10): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x b ( 11): 0e 0f 15 1d 18 10 15 10 1e 12 12 0c 0c 17 0b 1c 16 1d 15 11 0b 00 14 17 16 17 01 15 08 0c 1a 0c
0x c ( 12): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x d ( 13): 14 06 16 0c 19 11 1b 0d 0c 1c 15 0d 1b 16 13 00 1b 03 00 0a 06 07 0f 0f 15 1a 15 1a 0d 04 16 1e
0x e ( 14): 7f 7f 7f 7f 7f 7f 7f 8d 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e8 7f d9 7f 7f 7f 7f 7f 7f 7f 7f ca 7f 7f
0x f ( 15): 14 05 1e 0e 07 13 01 14 1a 03 07 08 06 13 19 0e 15 1c 19 0b 1b 05 19 02 09 03 11 03 12 05 14 0f
0x10 ( 16): 14 08 00 0d 19 04 1d 0f 04 03 0b 0e 05 02 0f 07 0d 0d 15 11 07 05 1a 01 03 02 19 10 1c 13 0a 0d
0x11 ( 17): 00 01 06 18 18 18 15 1e 07 15 0d 1d 0d 16 18 1d 17 0a 14 18 1d 07 16 16 04 03 11 12 0a 13 04 11
0x12 ( 18): 17 0b 10 08 01 1e 14 15 13 01 19 1a 07 1a 1d 05 1c 01 17 1e 04 0d 00 10 01 0f 05 05 14 07 04 08
0x13 ( 19): 07 1b 0e 0d 17 0a 07 03 09 07 16 0b 15 08 02 09 0c 1c 06 06 04 07 1d 1c 07 01 0f 12 03 0c 17 00
0x14 ( 20): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x15 ( 21): 19 06 16 0e 0f 02 1b 00 1a 00 0b 0d 12 08 1e 0a 06 16 14 12 1d 1b 09 1b 1b 01 0f 1a 0c 0a 0f 1a
0x16 ( 22): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x17 ( 23): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c6 7f 7f 7f 7f 7f 7f bb 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 86 7f
0x18 ( 24): 7f 7f 7f 7f fc 7f 7f 7f 7f 7f f4 7f 7f 7f 7f 7f 7f fd 7f 7f ba 7f 7f 7f f9 7f 7f 7f 7f 7f 7f 7f
0x19 ( 25): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1a ( 26): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1b ( 27): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1c ( 28): 09 0d 09 0b 1a 00 1c 1c 13 17 16 0d 05 03 12 15 16 1e 09 12 08 1a 02 12 1a 07 1e 0c 1e 09 1c 0b
0x1d ( 29): 0c 01 0a 11 04 1c 1c 0f 1c 15 10 04 1d 1c 0e 0b 02 13 1a 0c 08 13 18 05 08 13 01 19 09 19 0e 05
0x1e ( 30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1f ( 31): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b1 7f 7f
0x20 ( 32): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x21 ( 33): 18 13 09 14 16 0b 02 0e 06 05 0c 02 1b 08 1b 16 00 1b 19 10 1b 0e 00 04 09 1a 19 03 0a 03 07 0a
0x22 ( 34): 7f 7f b4 7f b7 7f 7f 7f eb 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d7
0x23 ( 35): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x24 ( 36): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x25 ( 37): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f cf 7f
0x26 ( 38): 7f 7f 7f 7f b9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 91 7f 7f 7f 7f 7f 7f
0x27 ( 39): 7f 7f 92 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ae 7f 7f 7f 7f 7f ef 7f 7f 7f 7f 7f 7f 7f 82 7f
0x28 ( 40): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x29 ( 41): 1c 0b 1b 0f 13 0e 0c 1b 12 05 09 0c 11 0b 08 09 1e 16 1a 1b 04 10 16 0f 10 13 13 05 01 08 01 09
0x2a ( 42): 7f 7f 7f ea 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ac 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x2b ( 43): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 9c 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x2c ( 44): 0b 05 01 10 18 14 06 15 01 0c 19 1c 0e 1a 0f 10 12 1e 17 0e 16 13 0a 18 19 12 1d 00 0f 13 0a 05
0x2d ( 45): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e5 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 8b 7f 7f
0x2e ( 46): 1c 13 10 0d 0e 00 11 1d 03 01 13 19 0d 02 13 14 07 19 10 18 07 0b 14 14 1a 1d 16 00 0e 14 18 09
0x2f ( 47): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 90 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x30 ( 48): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x31 ( 49): 17 02 09 12 1a 1a 0a 08 0b 03 08 17 1e 08 02 16 1d 18 0c 17 17 17 0f 1e 18 07 0d 02 0b 19 0f 10
0x32 ( 50): 11 05 0c 1c 11 19 08 1a 18 06 10 15 11 12 07 11 0a 18 0b 1e 11 07 0c 03 16 01 12 09 18 03 15 16
0x33 ( 51): 1e 13 19 1a 1a 0d 08 0f 1b 1b 0e 1c 17 01 13 18 01 0b 14 0f 10 02 08 1a 0a 0e 12 08 0c 0f 1a 1a
0x34 ( 52): 11 15 01 0c 0c 19 18 0d 0c 18 16 01 19 1a 08 14 0e 08 01 10 16 0c 13 05 19 16 0f 03 00 0c 07 01
0x35 ( 53): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x36 ( 54): 7f 7f 7f 7f 7f 7f d3 7f 7f 7f 7f 7f 7f 7f 7f 7f a9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x37 ( 55): 01 1d 0a 03 06 07 1a 05 08 03 0e 06 0a 16 1c 19 01 0b 1a 01 06 0f 0b 17 16 0e 07 07 08 02 14 04
0x38 ( 56): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x39 ( 57): 0a 04 1b 03 02 09 04 07 15 0a 1a 0b 1a 19 08 11 14 05 01 1b 00 05 04 1a 14 0f 0f 1d 06 19 09 08
0x3a ( 58): 05 05 03 0f 1a 1c 17 02 13 07 03 03 17 02 1a 0a 0b 1c 05 0d 1a 13 07 03 19 13 1d 05 08 15 09 1e
0x3b ( 59): 06 0d 1a 01 1d 06 0f 09 01 13 05 0c 19 0d 0a 04 00 19 0f 09 17 10 07 1d 0c 0a 1e 0c 0e 19 14 07
```

```
hex offset    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
dec offset    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

0x3c ( 60): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3d ( 61): 13 0c 02 0d 00 05 0e 0d 13 12 17 11 18 01 0e 1d 17 19 1e 04 0f 1b 09 10 05 02 1c 16 18 1b 11 1a
0x3e ( 62): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x3f ( 63): 7f 7f 7f 7f 7f 8f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ee 7f 7f 7f 7f e4 7f 7f 7f 7f 7f
0x40 ( 64): 10 13 14 14 1b 1e 1e 03 16 0c 08 03 1c 1b 10 0f 06 1c 17 05 11 1e 08 0b 0b 0b 03 13 07 15 0e 1a
0x41 ( 65): 7f 7f 7f 7f 7f 7f 7f 7f c4 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x42 ( 66): 1a 14 0c 01 0a 0f 0b 0d 07 06 1c 18 0b 19 02 13 08 08 0f 11 09 07 0c 19 1c 1b 16 1d 01 1b 0b 01
0x43 ( 67): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x44 ( 68): 0f 0b 0f 15 0f 17 0b 13 0c 11 17 03 01 06 07 04 11 0e 0e 17 03 1c 18 07 17 1a 00 0a 16 0a 05 08
0x45 ( 69): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x46 ( 70): 0a 07 0b 0c 08 09 07 09 14 13 13 1c 14 04 0c 10 17 10 02 0e 14 18 0d 06 01 0f 17 03 10 12 11 07
0x47 ( 71): 0d 1d 17 02 02 08 19 18 19 1a 16 11 1a 19 19 1b 10 02 13 15 1d 17 1b 1c 0a 1c 11 0d 0d 0f 0c 16
0x48 ( 72): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x49 ( 73): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4a ( 74): 05 10 07 1b 00 11 1b 13 04 10 19 13 1c 19 0b 03 15 0e 19 0d 16 04 17 1a 19 18 14 05 0a 08 09 12
0x4b ( 75): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4c ( 76): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f f1 7f 7f 7f 7f 7f
0x4d ( 77): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4e ( 78): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 80 7f 7f 7f 7f 7f 95 7f 7f 7f 7f 7f 7f
0x4f ( 79): 00 0d 1e 1e 15 06 06 04 0d 1d 06 19 11 08 0e 01 13 1c 0a 1e 08 0a 15 06 01 0f 0f 05 05 03 05 04
0x50 ( 80): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x51 ( 81): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x52 ( 82): 0f 12 01 0f 00 17 00 13 1e 1a 07 10 04 0a 1a 19 00 0b 13 05 18 1b 11 0b 1a 0f 05 08 04 1a 0f 15
0x53 ( 83): 1e 16 11 0b 14 1c 00 0b 08 18 14 00 1b 0c 04 03 02 07 02 15 04 0d 14 0d 09 0e 0e 11 12 09 15 1a
0x54 ( 84): 7f 7f 7f 7f 7f 7f 7f 7f 93 7f 7f 7f df 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f b3 7f 7f
0x55 ( 85): 19 01 1d 1b 0c 06 16 11 03 12 0f 12 00 10 0d 1a 16 16 11 07 19 08 1d 10 0c 0c 0e 0f 04 0c 01 11
0x56 ( 86): 7f 7f 7f 7f bd 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x57 ( 87): 18 0f 0e 10 04 0f 07 09 01 04 01 10 07 11 0f 07 03 12 14 15 0f 11 00 09 18 0b 03 0d 19 03 03 0f
0x58 ( 88): 02 11 14 08 1e 03 19 10 12 0d 01 0d 15 17 19 09 0f 03 0d 0d 18 02 01 13 18 02 15 04 1a 13 06 0b
0x59 ( 89): 09 1a 0e 16 01 18 06 0d 0f 0a 07 0d 01 04 04 1a 09 1e 1b 05 1a 1e 0d 0e 14 0f 08 16 06 0b 00 0a
0x5a ( 90): 1d 14 08 14 16 04 18 12 12 14 13 04 04 00 09 06 12 03 04 10 18 0f 01 1c 11 1c 17 1c 19 17 1a 14
0x5b ( 91): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x5c ( 92): 0f 0a 1e 0a 10 1b 1e 0d 0b 13 10 10 11 05 05 11 07 0e 01 0b 05 0f 1c 05 06 0a 08 02 02 15 1a 00
0x5d ( 93): 7f 7f 7f f7 7f 7f 7f 7f 7f 7f 7f c2 7f 7f 7f 7f 7f 7f 7f 7f f8 7f 7f 7f 7f b2 7f 7f 7f 7f 7f 7f
0x5e ( 94): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x5f ( 95): 08 16 0d 1c 06 07 09 02 03 1a 10 18 17 13 0c 1b 09 1b 02 12 1d 10 0b 0e 0d 05 0d 06 0f 10 11 15
0x60 ( 96): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f da 7f 7f 7f 7f 7f 7f a1 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x61 ( 97): 12 17 03 06 1d 16 0d 1e 01 12 04 09 06 0c 01 05 15 05 0f 1e 18 03 16 12 10 18 1d 0b 19 06 11 1b
0x62 ( 98): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x63 ( 99): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x64 (100): 19 18 15 1c 10 1a 08 04 07 0c 18 17 09 17 1b 0c 15 09 0c 06 09 02 0c 18 05 06 1c 0a 07 05 01 1b
0x65 (101): 17 17 02 09 1a 10 18 07 1a 1b 05 16 1e 01 07 1c 15 0b 05 1b 01 1e 01 0a 0f 12 19 0c 1d 0a 0b 1e
0x66 (102): 7f 7f 7f 7f 7f 7f 7f fb 7f 7f 7f 7f 7f 9d 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f ed 7f 7f 7f
0x67 (103): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x68 (104): 0e 18 0f 02 0a 0b 1c 12 0e 13 01 15 19 14 0b 08 00 02 04 0e 18 06 0e 13 13 00 1c 12 09 07 15 1c
0x69 (105): 14 05 01 10 01 1c 0f 10 0c 14 1c 09 0b 12 09 12 10 0d 14 10 02 1e 0b 0d 0c 1c 1d 0f 0c 02 1b 05
0x6a (106): 16 02 11 03 10 15 15 00 10 05 0f 0f 0f 17 09 0e 07 06 13 04 18 0b 03 18 00 09 14 00 15 11 1e 09
0x6b (107): 00 1c 0e 1e 17 0e 01 03 18 00 01 15 12 14 02 1d 17 00 11 0a 07 13 1e 00 18 1c 02 1b 00 09 00 05
0x6c (108): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6d (109): 04 18 18 09 00 03 17 1e 02 07 0c 02 13 1a 1a 05 1c 18 04 1a 1c 1b 17 19 10 00 15 0f 04 04 1b 1c
0x6e (110): 14 02 09 1c 1e 1d 1b 1c 1b 0e 07 10 01 11 08 06 0d 14 0e 03 1e 06 11 19 15 0d 1b 02 10 03 0f 09
0x6f (111): 0d 0b 0a 1c 15 0a 0c 0b 03 1d 1a 1d 10 01 02 06 0e 09 1d 1b 0d 09 0b 0d 0d 03 19 16 1e 09 0d 18
0x70 (112): 7f 7f 7f 7f 7f 7f 7f dc 7f e1 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d5 7f 7f 7f 7f 7f 7f 7f 7f f5
0x71 (113): 07 11 1d 0b 16 00 17 1a 10 1b 12 18 0f 0b 08 11 17 01 14 01 13 16 1c 14 14 15 14 15 15 1a 0b 1d
0x72 (114): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 84 7f c7 7f 7f 7f 7f 7f c0 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x73 (115): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x74 (116): 11 17 17 1c 19 13 16 12 07 00 1c 03 0d 05 07 05 10 03 1a 12 04 06 16 1d 0c 00 07 11 0b 16 01 1e
0x75 (117): 16 13 1a 19 19 17 0c 17 0c 11 00 04 14 02 03 1b 1b 1e 18 09 12 10 02 04 13 0d 11 1c 02 04 1e 03
0x76 (118): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x77 (119): 10 15 0d 09 05 11 15 10 0e 12 06 1c 1a 05 1a 15 01 17 19 09 06 0e 01 04 15 0b 0c 1c 18 09 01 01
0x78 (120): 1b 18 14 09 08 07 01 1c 09 00 09 03 0d 0a 05 0e 0c 0a 19 09 10 1b 1d 0b 01 1d 0f 06 03 07 09 04
0x79 (121): 01 0d 03 1d 1d 0d 1a 07 1a 05 09 10 17 0e 03 01 04 08 14 05 1c 14 09 11 1a 07 14 13 19 14 01 1a
0x7a (122): e0 8e a7 ab ff f2 bf af b6 f0 dd ce 88 d4 97 fe 7f aa 7f 7f cc c1 a2 e6 ad a6 98 9f 87 7f a5 d6
0x7b (123): 1b 13 1d 07 16 19 04 1c 1e 06 05 03 05 0a 18 06 07 15 19 00 18 15 09 0e 01 0d 00 17 10 05 06 12
0x7c (124): 0c 1b 17 0b 14 06 05 01 0b 07 10 1e 04 11 1a 12 10 17 00 1d 15 19 19 03 02 05 0f 1d 1c 0c 13 16
0x7d (125): 0f 02 0a 19 00 13 10 16 0b 1d 03 0c 09 11 06 0b 0d 1c 00 08 04 16 17 00 18 1b 0e 12 18 1c 06 15
0x7e (126): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 85 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
0x7f (127): 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 83 7f e9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
hex offset    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
dec offset    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

PDBR: 0x7a (122)

83) When accessing **virtual address 0x5891**, what will be the first page accessed (hexadecimal)?
   a. 0x32
   b. 0x58
   c. 0x63
   d. 0x7a
   e. Error or None of the above

Access page of page directory first (page directory base register = pdbr) = 0x7a

84) What **index** of the page directory will be accessed first (hexadecimal)?
   a. 0x05
   b. 0x16
   c. 0x17
   d. 0x1f
   e. Error or None of the above

Write out address in binary: 0101 1000 1001 0001 → regroup 10110 00100 10001
Top five 5 bits used to select entry in page directory → 10110 → 16 + 4 + 2 = 22decimal → 0x16
Get 0xa2 from offset 0x16 of page 0x71

85) What will be the **second page** accessed? (hexadecimal)
   a. 0x08
   b. 0x22
   c. 0x28
   d. 0x36
   e. Error or None of the above

Write out 0xa2 → 1010 0010; top bit indicates valid; 32 + 2 = 34 → 0x22

86) What are the **contents** of the corresponding PTE that will be read?
   a. 0x7f
   b. 0xb7
   c. 0xe7
   d. 0xf7
   e. Error or None of the above

From address, 00100 -> 4 is the offset; 0xb7

87) What is **the final physical address** for this virtual address?
   a. 0x091
   b. 0x6f1
   c. 0x991
   d. 0xdf1
   e. Error or None of the above

Write out 0xb7 → 1011 0111 → valid. Write out bits to get address 0110111 10001; to get hex, regroup as 110 1111 0001 → 0x6f1

88) What are the **contents** (i.e., the value) at that final physical address?
   a. 0x14
   b. 0x01
   c. 0x0b
   d. 0x7f
   e. Error or None of the above

ppn (from is 32 + 16 + 7 = 55 is the page number; offset is 10001 or 17

89) How many memory accesses were required in this example to obtain the contents at the final physical address?

a. 1
b. 2
c. 3
d. 4
e. Error or None of the above

Two to get the page table entries, one to obtain the contents at the final address

Part 7. Page Replacement Policies

Assume you have the following stream of accesses to virtual page numbers for some workload:

```
5, 0, 5, 4, 1, 3, 0, 2, 4, 5, 3, 2, 1
```

90) If physical memory contains 4 pages, how many misses will be incurred with the OPT replacement algorithm?

- a. 7
- b. 8
- c. 9
- d. 10
- e. None of the above

5 0 4 1 → 4 misses
For 3, replace access furthest in future → page 1 → 5043 → 5 misses
0 hit
For 2, replace 0 → 5243 → 6 misses
4 hit
5 hit
3 hit
2 hit
1 miss (doesn't matter what you replace) → 7 misses

91) If physical memory contains 4 pages, how many misses will be incurred with the LRU replacement algorithm?

- a. 7
- b. 8
- c. 9
- d. 10
- e. None of the above

5 0 4 1 → 4 misses
For 3, replace furthest in past, which is 0; 5341 → 5 misses
For 0, replace 5, 0341 → 6 misses
For 2, replace 4, 0321 → 7 misses
For 4, replace 1. -0324 → 8 misses
For 5, replace 3, 0524 → 9 misses
For 3, replace 0, 3524 → 10 misses
2 hit
For 1, replace 4, 3521 → 11 misses

```
5, 0, 5, 4, 1, 3, 0, 2, 4, 5, 3, 2, 1
```

92) If physical memory contains 4 pages, how many misses will be incurred with the FIFO replacement algorithm?

a. 7
b. 8
c. 9
d. 10
e. None of the above

5 0 4 1 → 4 misses
For 3: 3 0 4 1 → 5 misses
0 hit
For 2: 3 2 4 1 → 6 misses
4 hit
For 5: 3 2 5 1 → 7 misses
3 hit
2 hit
1 hit

Congratulations on finishing your first CS 537 Exam!