

537 Midterm Exam,
Old School Style

"Drawing Pretty Pictures"

Name: _____

This exam contains 13 old-school pages.

In Kindergarten, most of you spent a lot of time drawing and coloring. Those were the days! When school was easy, and your biggest concern was whether you forgot to bring your lunch, and how to tie your shoes. Hint: use the Bunny-knot method -- it is easier. How times have changed, with virtual page tables, context switches, and virtual machines cluttering your brains.

In this exam, we'll reconnect to the days of your youth by remembering that in OS class, you too can draw pictures. In fact, almost all answers in this exam are pictorial in nature; all you have to do is know what picture to draw.

Good luck!

Answers are accepted in pencil, pen, and (of course) crayon.

1. _____

2. _____

3. _____

4. _____

5. _____

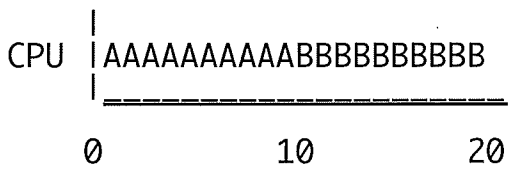
6. _____

7. _____

8. _____

Total: _____

Scheduling policies can be easily depicted with some graphs. For example, let's say we run job A for 10 time units, and then run job B for 10 time units. Our graph of this policy might look like this:



In this question, you'll show your understanding of scheduling by drawing a few of these pictures.

(a) Draw a picture of Shortest Job First (SJF) scheduling with three jobs, A, B, and C, with run times of 5, 10, and 15 time units, respectively.

Make sure to LABEL the x-axis appropriately.



(b) What is the average TURNAROUND TIME for jobs A, B, and C?

(c) Draw a picture of ROUND-ROBIN SCHEDULING for jobs A, B, and C, which each run for 6 time units, assuming a 2-time-unit time slice; also assume that the scheduler (S) takes 1 time unit to make a scheduling decision.

Make sure to LABEL the x-axis appropriately.



(d) What is the average RESPONSE TIME for round robin for jobs A, B, and C?

(e) What is the average TURNAROUND TIME for round robin for jobs A, B, and C?

Assume you have a multi-level feedback queue (MLFQ) scheduler. In this question, we'll draw a picture of how it behaves over time.

Unlike the drawings in the previous problem (for SJF and RR), the y-axis will also be important for these pictures, as it will show the PRIORITY of the jobs over time.

(a) Assume a 3-level MLFQ (high priority is 2, low priority is 0). Assume two jobs (A and B), both BATCH jobs (no I/O), each with a run-time of 10 time units, and both entering the system at $T=0$. Assume the quantum length at the highest priority level is 1, then 2 at the middle, and 3 for the lowest priority.

Draw a picture of how the scheduler behaves for these jobs. Make sure to LABEL the x-axis.



(b) Assume the same scheduling parameters as above. Now the jobs are different; A and B both are BATCH jobs that each run for 10 time units (no I/O again), but this time A enters at $T=0$ whereas B enters the system at $T=6$.

Draw a picture of how the scheduler behaves for these jobs. Make sure to LABEL the x-axis.



(c) Calculate the RESPONSE TIME and TURNAROUND TIME (in part b) for Job A

(d) Calculate the RESPONSE TIME and TURNAROUND TIME (in part b) for Job B

Assume you have a chunk of memory that you need to manage. When someone requests a chunk, you take the first available chunk and return it, starting at the lowest address in the space you are managing (i.e., a LOWEST-ADDRESS-FIRST policy, perhaps). The space is managed with a simple free list; when someone returns a chunk, you COALESCE the list, thus merging smaller allocated chunks back into a bigger free space.

Assuming you have 50 bytes of memory to manage, and that exactly one allocation has taken place (for 10 bytes), here is what memory would look like (with spaces in-between every 10 bytes for readability):

HHHHAAAAAA AAAAFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF

(low) Addresses of Managed Space (high)

In the picture, A means allocated, F means free, and H is a 4-byte header that is REQUIRED before every allocated chunk.

(a) Assume a 50-byte free space. Draw what it would look like after these requests: allocate(10), allocate(10), and allocate(10).

(b) Assume a 50-byte free space. Draw what it would look like after allocation requests of allocate(10), allocate(20), and allocate(20).

(c) Assume a 50-byte free space. Draw what it would look like after the following requests: x=allocate(10), y=allocate(10), z=allocate(10), free(y), w=allocate(24).

(d) Assume now that there is NO COALESCING of free space. Also assume that instead of allocating via the policy of LOWEST-ADDRESS-FIRST, you instead use a BEST-FIT policy. Assume a 50-byte free space. Draw what it would look like after the following requests: x=allocate(10), y=allocate(10), z=allocate(10), free(y), free(z), w=allocate(4).

Assume virtual memory hardware that uses segmentation, and divides the address space in two by using the top bit of the virtual address. Each segment is thus relocated independently.

What we'll be drawing in this question is what physical memory looks given some different parameters. We'll also label where a particular memory reference ends up.

For all questions, assume a virtual address space of size 16 bytes (yes tiny!) and a physical memory of size 64 bytes. Thus, if we had a virtual address space placed in physical memory, it might look like this (with spaces between every 8 physical bytes):

```
0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF1111
```

In this example, the segment 0 base register is 0, segment 1 base is 64 (it grows backwards), and both length registers are 4. 0's are used to record where segment 0 is in memory; 1's are for segment 1; F means free.

(a) What would physical memory look like if we had the following values instead? (draw a picture below)

```
seg0 (base)  : 12      seg0 (limit) : 6
seg1 (base)  : 10      seg1 (limit) : 3
```

(b) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 4 (or DRAW AN X on the physical-memory address if the access is illegal)

(c) What would physical memory look like if we had the following values instead? (draw a picture below)

```
seg0 (base)  : 40      seg0 (limit) : 4
seg1 (base)  : 50      seg1 (limit) : 4
```

(d) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 14 (or DRAW AN X on the physical-memory address if the access is illegal)

(e) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 4 (or DRAW AN X on the physical-memory address if the access is illegal)

Which memory is accessed during the execution of an instruction?
 For this question, assume a linear page table, with a 1-byte page-table entry. Assume an address space of size 128 bytes with 32-byte pages. Assume a physical memory of size 128 bytes. The page-table base register is set to physical address 16. The contents of the page table are:

VPN	PFN
0	1
1	Not valid
2	3
3	Not valid

Now, finally assume we have the following instruction, which loads a SINGLE BYTE from virtual address 70 into register R1:

10: LOAD 70, R1

This instruction resides at virtual address 10 within the address space of the process.

In the diagram of physical memory below:

(a) Put a BOX around each valid virtual page (and label them)

(b) Put a BOX around the page table (and label it)

(c) CIRCLE the memory addresses that get referenced during the execution of the instruction, including both instruction fetch and data access (there is no TLB).

(d) LABEL these addresses with a NUMBER that indicates the ORDER in which various physical addresses get referenced.

Physical Memory

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
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127

In this question, you will examine virtual memory reference traces. An access can be a TLB hit or a TLB miss; if it is a TLB miss, the reference can be a page hit (present) or a page fault (not present).

Assume a TLB with 4 entries, and a memory that can hold 8 pages. Assume the TLB and memory both are empty initially. Finally, assume LRU replacement is used for both the TLB and memory.

(a) What happens on each access in the following reference trace? a TLB hit, TLB miss/page hit, or TLB miss/page fault? (these can be abbreviated H, M, or PF)

0
1
2
3
0
1
2
3

(b) What happens on each access in the following reference trace? (write H, M, or PF)

0
1
2
3
4
0
1
2
3
4

(c) Now assume a memory that can only hold 3 pages. What happens on each access in the following reference trace? (H, M, PF)

0
1
2
0
1
3
0
3
1
2
1

In this question, we'll examine a multi-level page table, like that found in the (optional) homework. The parameters are the same:

- The page size is an unrealistically-small 32 bytes.
- The virtual address space for the process in question (assume there is only one) is 1024 pages, or 32 KB.
- Physical memory consists of 128 pages.

Thus, a virtual address needs 15 bits, 5 of which are the offset. A physical address requires 12 bits, also with 5 as the offset.

The system assumes a multi-level page table. Thus, the upper five bits of a virtual address are used to index into a page directory; the page directory entry (PDE), if valid, points to a page of the page table. Each page table page holds 32 page-table entries (PTEs). Each PTE, if valid, holds the desired translation (physical frame number, or PFN) of the virtual page in question.

The format of a PTE is thus:
VALID | PFN7 PFN6 ... PFN0

and is thus 8 bits or 1 byte.

The format of a PDE is essentially identical:
VALID | PT7 PT6 ... PT0

For this question, assume the PDBR (page-directory base register) is 73.

On the next page is the physical memory dump, where your answers will go.

(a) CIRCLE which bytes are accessed during a load from virtual address 0x3009.

(b) Put SQUARES around bytes accessed during a load from 0x7042.

page 64: 14 0b 15 14 01 18 05 00 17 0a 04 08 14 09 12 06 1d 02 0f 00 13 04 1d 0a 16 11 1e 0c 02 02 09 1c
page 65: 0d 0d 14 12 19 15 0f 10 14 08 0e 03 15 00 17 1e 14 17 1c 06 0c 19 1a 1a 13 1c 1c 05 10 00 0a 10
page 66: 7f 7f 7f 7f 7f 7f 7f 7f 91 df 7f 7f 7f 7f 7f 7f 7f 7f 7f a4 7f 7f 7f 7f 7f 7f 7f 7f ce
page 67: 7f 7f 7f 7f e1 7f 7f e9 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f c0 7f 7f c1 7f 7f 7f 7f
page 68: 00
page 69: 0a 1c 00 07 00 15 05 06 07 02 0b 08 1a 11 03 0d 00 15 01 16 1d 1c 19 13 03 16 11 17 02 0f 19 13
page 70: 17 07 08 07 0b 01 1e 09 14 17 17 0a 19 15 12 0d 04 05 0e 16 01 1e 18 04 1e 08 07 16 0e 0d 02 00
page 71: 00
page 72: 19 0e 1a 1c 16 16 11 02 12 07 0e 1c 17 17 0c 11 0f 00 14 13 1c 16 01 1d 0e 10 1c 17 0d 03 03 11
page 73: a2 d2 97 96 d9 7f 87 b4 b7 f2 f4 82 bf 7f be 93 e8 9d 99 9e f1 7f 7f b0 d8 da eb b1 81 c3 c2 f6
page 74: 17 05 04 1e 03 0b 08 05 04 00 10 08 1b 0f 09 0b 13 13 12 0d 08 02 19 0d 16 05 06 19 13 0a 17 0c
page 75: 01 17 0a 15 19 0d 17 01 04 18 1b 01 0d 0c 07 1b 0c 0e 0a 0e 06 12 01 09 0a 11 0e 06 10 15 0a 19
page 76: 00
page 77: 00
page 78: 00 12 08 1a 07 06 12 15 07 1c 11 0f 00 1a 0e 19 01 0e 0f 0c 0a 11 01 19 14 00 1c 09 05 0e 1c 08
page 79: 1c 08 16 13 0c 15 0e 17 09 09 06 1a 0f 00 1d 0f 08 03 12 19 04 07 06 1b 14 02 09 03 1c 1c 06 00
page 80: 0e 1b 04 13 1d 09 01 1e 1c 14 19 0c 1a 06 0c 11 16 17 1b 0a 1c 0b 0d 00 09 17 1b 10 12 1e 09 13
page 81: 1d 1d 17 02 16 1a 0a 16 10 06 09 0a 19 13 1e 06 09 18 04 11 02 0a 1b 09 09 09 05 10 04 02 15 00
page 82: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f bd 7f 7f ae 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 83: 00
page 84: 0f 13 01 0c 19 1e 08 09 1a 13 17 13 18 1c 17 0b 05 01 00 0a 13 0e 07 10 17 16 1e 03 0d 03 1d 09
page 85: 00
page 86: 00
page 87: 08 0f 13 1e 1c 10 06 0c 02 15 15 1b 07 1c 07 02 16 09 13 09 0a 0e 04 09 14 03 18 01 04 1d 09 06
page 88: 7f
page 89: 7f 7f fe 7f 7f 7f d7 7f
page 90: 7f
page 91: 1c 0e 0e 0f 02 19 1d 0e 07 17 0c 14 05 1c 1a 09 04 1a 04 0b 0a 15 09 12 0f 0b 1d 16 16 10 07 10
page 92: 00
page 93: 00
page 94: 13 10 13 10 16 10 04 0f 04 1b 09 04 0d 03 16 18 09 13 14 0a 03 0b 0e 10 02 14 00 15 1c 0c 11 0e
page 95: 15 14 19 11 08 06 1b 06 1d 16 1d 08 0f 03 11 1a 1c 1d 09 08 0d 06 05 16 1b 14 18 0d 02 1e 1b 1c
page 96: 05 02 1d 09 1e 1a 00 1c 0a 1c 0c 12 1a 1b 10 03 13 02 1e 0e 0e 03 08 12 16 13 18 08 0f 16 1a 1e
page 97: 12 10 1b 02 16 12 10 1b 1a 0b 14 18 1a 13 03 1c 1c 1c 11 19 01 0c 04 0c 11 13 09 16 04 18 10 15
page 98: 00
page 99: 11 1e 16 02 00 15 0c 11 1e 11 0a 01 1e 1b 12 15 17 07 11 06 09 0c 19 0f 0a 14 1d 1b 04 03 0d 12
page 100: 00
page 101: 0f 02 13 0f 07 12 09 1a 11 0d 07 1e 0c 09 07 05 12 12 0d 17 08 1b 07 0c 0f 11 11 15 01 0b 0d 1d
page 102: 01 04 08 1a 04 07 06 0c 07 01 1a 0c 18 17 0e 12 11 0c 10 13 0e 1c 00 00 0a 15 13 10 18 0e 0f 17
page 103: 14 17 09 16 05 09 10 1e 1d 0d 02 06 02 07 0e 16 1b 1c 17 16 0a 14 01 12 00 19 0c 0e 19 0a 0e 14
page 104: 7f 7f 7f 7f 7f 7f 7f 7f cf 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f d4 7f 7f 7f 7f 7f 7f 7f 7f
page 105: 09 09 04 04 0b 07 1e 1c 0d 13 1a 0c 04 01 07 02 04 18 1c 1c 06 1d 03 1e 01 02 0c 04 15 19 1a 0d
page 106: 0d 1e 03 1b 13 07 0e 0a 11 0e 05 1b 00 05 13 0f 03 1e 1d 1b 1c 17 0b 17 06 0c 04 06 08 12 12 1e
page 107: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e3 7f 7f 7f e5 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 108: 13 02 05 13 10 12 1a 12 08 00 0c 13 02 1b 0c 0c 06 17 11 13 15 1a 17 16 1c 11 02 04 10 10 05 19
page 109: 0d 1d 1d 17 13 18 18 18 0c 0c 01 01 0d 18 15 13 1b 06 0f 06 00 05 14 12 1b 12 02 11 1d 03 1a 06
page 110: 01 18 1a 18 16 12 02 0e 00 00 1b 18 08 0e 1c 19 07 19 1b 07 0a 12 11 07 05 0a 1b 14 02 13 13 0a
page 111: 15 00 17 11 13 1b 12 15 13 01 01 12 08 11 0c 10 03 00 18 13 03 1b 19 10 1d 05 16 09 0f 1a 06 04
page 112: 00
page 113: 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e0 7f 7f 7f 7f 7f 7f 7f 7f e6 7f 7f 7f 7f 7f
page 114: 7f 86 7f 7f 7f 7f 7f 7f 7f 7f
page 115: 00
page 116: 7f
page 117: 1a 1a 0b 1e 1d 0a 00 15 16 0d 07 1e 13 09 16 07 02 05 12 0d 05 03 12 02 00 08 07 0f 01 1d 11 14
page 118: 7f 7f 7f 95 7f 7f 7f 7f 7f 7f 7f 7f af 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
page 119: 07 17 05 0e 16 17 14 03 0b 06 00 0d 03 09 07 18 06 11 11 10 02 0f 15 04 10 1c 1e 02 05 1d 11 16
page 120: 18 1d 0f 11 09 14 07 06 0e 18 19 01 09 07 15 04 00 1b 15 05 11 0c 00 09 10 0f 0b 09 18 15 06 1a
page 121: 00
page 122: 07 0a 06 14 14 03 10 0b 09 13 1b 0c 18 13 0a 0d 07 10 0c 07 1e 00 0b 05 0c 12 05 1e 11 10 17 1e
page 123: 15 14 13 00 19 0f 05 1e 1e 03 1a 0b 00 09 11 1b 1c 0a 11 19 10 16 15 09 0c 18 04 1c 1d 1e 1b 07
page 124: 0c 02 1c 05 19 16 03 1a 09 12 15 1e 09 09 00 1e 03 08 10 04 06 05 03 09 1a 0d 1b 11 00 1e 18 03
page 125: 00
page 126: 00 0d 00 16 06 08 0a 0e 11 1d 14 0d 10 1c 00 0f 10 09 1a 1e 0f 0d 11 19 1b 02 0e 18 1c 06 11 02
page 127: 15 0e 0a 1a 03 14 1b 00 01 14 1a 02 05 02 12 0b 08 13 13 05 1c 07 15 02 0b 17 1b 03 0c 18 0f 09

Ah, virtual machine monitors. You use them, and now (hopefully) you understand them (a little bit).

Assume in this question some hardware that has a software-managed TLB.

Assume we are running a virtual machine monitor (VMM), an operating system (OS) on top of the VMM, and a user process running on the OS.

Draw a picture of the control flow during a TLB miss generated by the user process. The picture should reflect a time-line of what happens during this miss, including when the user process, OS, and VMM run, and what they do when they run.