CS2106 Operating Systems
2025/26 Semester I
Final Assessment
SUGGESTED SOLUTIONS

Part 1 – Multiple Choice Questions (20 MARKS)

There are TEN (10) questions in this part, answer ALL questions. Each question has exactly one correct option. In case of ambiguity choose the option that you think is the most correct.

Question 1. Which of the following statement(s) is/are true about the **pure segmentation disjoint memory scheme**?

i. In the pure segmentation scheme the address space of a process must be contiguous.
ii. In the pure segmentation scheme the address space of a segment must be contiguous.
iii. In the pure segmentation scheme the start of the next segment must come immedately after the end of the previous segment.
iv. In the pure segmentation scheme a segment of one process might be adjacent to a segment of another process.
v. Segments of a process can overlap.

a. i., ii. and iv. ONLY are TRUE.
b. ii., iii., and iv. ONLY are TRUE.
c. ii. and v. ONLY are TRUE.
d. +ii., iv. ONLY are TRUE.
e. NONE of the options a. to d. are correct.

Commented [KT1]: There were some questions about whether this is physical or logical memory. In pure segmentation (i.e. without paging), it doesn't matter.

Process memory space does not need to be contiguous in physical or logical memory since segments do not need to be adjacent.

Segments must always be contiguous in logical space because that is how the CPU sees memory, and without paging must be contiguous in physical space since it is defined only by a start address and a length.

Question 2. Which of the following statement(s) is/are true about managing **contiguous memory**?

i. Using bitmaps to manage patitions is more memory efficient but slower than using linked-lists.
ii. To minimize overheads, memory should be managed at the individual byte level.
iii. In fixed-size partition schemes it is possible to allocate two non-contiguous partitions to a single process.
iv. It is not possible to have internal fragmentation with variable-sized partitions.

v. It is not possible to have external fragmentation with buddy allocation.

a. ii. and v. ONLY are TRUE.
b. i, iii and iv. ONLY are TRUE.
c. ii., iii. and iv. ONLY are TRUE
d. +i., iv. ONLY are TRUE.
e. None of the options a. to d. are correct.

Question 3. We have a system with 1MiB of memory in total with 1KiB allocation units (1 KiB = 1024 bytes, 1 MiB = 1024 KiB). We are using **buddy-allocation**, and our algorithm returns the first free partition that is greater than or equal to the requested size.

We are given the following allocation requests:

A: Allocate 50 KiB
B: Allocate 135 KiB
C: Allocate 128 KiB
D: Allocate 200 KiB

Allocation (Note: X/Y means block starting at allocaton unit X of size Y)

A: Allocate 50 KiB (IF: 14KiB)

10	~~0/1024~~
9	~~0/512~~, 512/512
8	~~0/256~~, 256/256
7	~~0/128~~, 128/128
6	**0/64 (A)**, 64/64

B: Allocate 135 KiB (IF: 121 KiB)

10	~~0/1024~~
9	~~0/512~~, 512/512
8	~~0/256~~, **256/256 (B)**
7	~~0/128~~, 128/128
6	0/64 (A), 64/64

C: Allocate 128 KiB (IF: 0 KiB)

10	~~0/1024~~
9	~~0/512~~, 512/512
8	~~0/256~~, 256/256 (B)
7	~~0/128~~, **128/128 (C)**
6	0/64 (A), 64/64

D: Allocate 200 KiB (IF 56 KiB)

10	~~0/1024~~
9	~~0/512, 512/512~~
8	~~0/256~~, 256/256 (B) **512/256 (D)**, 768/256
7	~~0/128~~, 128/128 (C)
6	0/64 (A), 64/64

Average IF: $\frac{14+121+0+56}{4} \times 1024 = 65195\ bytes$

Which of the following shows the correct starting byte addresses for each request?

a. A: 0, B: 256, C: 128, D: 512
b. A:0, B:524288, C: 786432, D: 262144
c. +A:0, B: 262144, C: 131072, D: 524288
d. A:0, B:512, C: 768, D: 256
e. None of the above options a. to d. are correct.

Question 4. Continuing with question 3, what is the AVERAGE internal fragmentation in bytes?

Commented [KT2]: Several students got this wrong as they gave the IF in allocation units rather than bytes, resulting in fractional answers.

a. 47750 bytes
b. +48896 bytes
c. 48 bytes
d. 47 bytes
e. None of the above options a. to d. are correct.

Question 5. Which of the following statement(s) about **segmentation with paging** (as shown in the lecture) is/are TRUE?

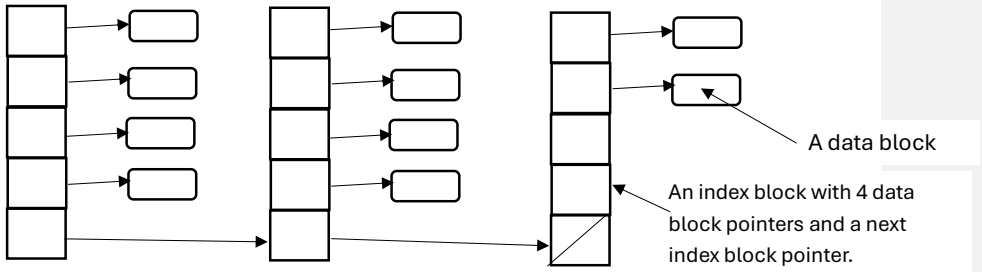
i. Segments must be contiguous in physical memory space.
ii. Segments must be contiguous in logical memory space.
iii. Segments are described by their starting address and length of the segment in bytes in logical memory space.
iv. Segments are described by the starting address and length of the segment in bytes in physical memory space.
v. Segments are described by pointers to their respective page tables and the number of valid pages used by the segment.

a. +ii. and v. ONLY are TRUE
b. ii. ONLY is TRUE.
c. iii. and iv. ONLY are TRUE.
d. i. and v. ONLY are TRUE.
e. None of the above options a. to d. are correct.

Question 6. Which ONE of the following statements about virtual memory is TRUE, given a particular total virtual memory space.

a. Having larger page sizes reduces the number of page faults due to more efficient transfer from secondary storage.
b. Having smaller page sizes reduces the number of page faults due to more efficient transfer from secondary storage.
c. +Having smaller page sizes increases the page table overhead.
d. Having larger page sizes increases the page table overhead.
e. None of the above statements a. to d. are TRUE.

For questions 7 to 10, we have a 2 MiB filesystem with blocks of size 4 KiB (1 KiB = 1024 bytes). We have a hybrid indexed/linked-list structure to maintain information about the location of each data block (blocks that contain actual file data) of a file that looks like this:



(Example structure: Actual structure may have more entries in the index blocks).

The rectangles with the rounded corners represent data blocks.

So entries in the index blocks points to a data block in the file, except the last entry which points to the next index block. In the diagram above we see an example of a file with 10 data blocks.

Index blocks are themselves stored in filesystem blocks but are not data blocks (since they don't store user file data), and are hence considered to be overhead.

Question 7. How many blocks are there in total in our filesystem?

Commented [KT3]: There's a phrasing issue in the options; this question is meant to ask for total blocks in the FS, which was clarified via an announcement.

2 MiB $= 2^{21}$, 4 KiB = 2^{12} Number of blocks = $\frac{2^{21}}{2^{12}} = 2^9$ = 512 data blocks.

a. 64 data blocks
b. 128 data blocks
c. 256 data blocks
d. +512 data blocks
e. None of the above options a. to d. are correct.

Question 8. Suppse that your answer to Question 7 is 256 data blocks (NOTE: This is not necessarily the answer to Question 7). How many data blocks can each index block point to, assuming that we use the mininum number of bits to hold a single block number?

256 blocks means that each block pointer is 8 bits, i.e. 1 byte. Since the block size is 4096 bytes, with 1 byte serving as a pointer to the next index block, we can point to 4095 blocks.

a. 2047 data blocks.
b. 2048 data blocks.
c. +4095 data blocks.
d. 4096 data blocks.
e. None of the above options a. to d. are correct.

Question 9. Again assuming that your answer to Question 7 is 256 blocks, and that your answer to Question 8 is 2048 data blocks, what is the maximum size of a file in bytes, taking into account overheads from the index blocks? Additionally two filesystem blocks are taken up by the filesystem directory.

A single index block is enough to point to all the data blocks, and with two directory blocks, total overhead is 3 blocks. Available blocks for file data is 256 – 3 = 253 blocks. Each block is 4096 bytes, total is $253 \times 4096 = 1{,}036{,}288\ bytes$

a. +1,036,288 bytes
b. 1,044,480 bytes
c. 1,012 bytes
d. 1,024 bytes
e. None of the above options a. to d. are correct.

Question 10. Again assuming that your answer to Question 7 is 256 blocks and your answer to Question 8 is 2048 data blocks, how many accesses in total does it take to read a file of size 512 KB? Assume that only the entire directory is in memory.

512 KiB = 128 data blocks. We need to read the index block + the data blocks = 129 disk accesses.

a. 128 disk accesses
b. +129 disk accesses
c. 255 disk accesses
d. 256 disk accesses
e. None of the above options a. to d. are correct.

Part 2 – Long Questions

There are TWO (2) questions in this section, and each question may have multiple parts. The total marks of each question and each part is indicated. Answer ALL questions.

Question 11. Virtual Memory (10 MARKS)

We look at a virtual memory system where the total virtual address space is 1 MiB (1 MiB = 1024 KiB, 1 KiB = 1024 bytes), and each page/frame is 16 bytes long. The physical address space is 256 KiB long. (Note that these sizes are unrealistic in a real system, but the values are kept small to keep calculations simple). Each page-table entry (PTE) consists of a frame number, a valid bit, a resident bit, a dirty bit and 9 permission bits.

a. (1 marks) How big is each PTE? Express your answer in bytes (Note: It is not possible to have a PTE size with fractions of bytes, e.g. 3.25 bytes)

# of frames = $\frac{2^{18}}{2^4} = 2^{14}$
Frame length =14 bits
Valid bit = 1 bit, resident bit = 1 bit, dirty bit = 1 bit, permissions = 9 bits
Housekeeping bits = 1 + 1 + 1 + 9 = 12 bits
Total = 14 + 12 = 26 bits = 3.25 bytes = 4 bytes

b. (2 marks) Suppose your answer to part a. is 16 bytes (note: This may not be the correct answer). How much memory IN TOTAL is used by the page tables if there are at the moment 5 processes running? Express your answer in MiB (1 MiB = 1024 KiB, 1 KiB = 1024 bytes)

# of pages = $\frac{2^{20}}{2^4} = 2^{16}$
Total size of 1 PT = $2^{16} \times 2^4 = 2^{20}$ = 1 MiB
Total size for 5 processes = $5 \times 1 = 5\ MiB$

c. (3 marks) Suppose your page table has 8192 entries and your TLB has 8 entries. If it takes 10ns ($1\ ns = 10^{-9}\ seconds$) to perform a virtual address translation in the event of a TLB hit and 100 ns in the event of a TLB miss, what is the AVERAGE address translation time if memory is accessed in a uniformly random manner? Express your answer in nanoseconds to two decimal places.

Note that we are only asking for the average time to translate an address, and your answer should not include the time taken to read the instruction or data that the CPU requests for.

Average TLB hit rate = $\frac{8}{8192}$

Average translation time = $\frac{8}{8192} \times 10 + \left(1 - \frac{8}{8192}\right) \times 100$

= 99.91 ns

d. (4 marks) Continuing on with part c, we have the following program. We assume that $5 holds the starting address of an array.

```
        load $1, 0          // Set $1 to 0
loop:   bge $1, 4, end      // Branch to end if $1 >= 4
        lw $2, ($5)         // Load from address in $5
        add $2, $2, 5       // $2 = $2 + 5
        sw $2, ($5)         // Write back to address in $5
        add $5, $5, 4       // Increment $5 by 4
        add $1, $1, 1       // Increment $1 by 1
        j loop
end:    .. Irrelevant code ..
```

Commented [KT4]: This comment was originally $1>=100 which was corrected to $1>=4 via announcement at the start of the paper.

The discrepancy should also be apparent when looking at the bge statement.

Assume that both the code above starts at the first byte of a page. Similarly the elements of the array start at the first byte of a page. If each instruction and each array element is 4 bytes long, what is the AVERAGE address translation time if we run the program above? Ignore the "irrelevant code" at the end.

As before we are only looking for the average time it takes to translate an address. The TLB is initially empty.

Each instruction is 4 bytes, there are 16 bytes per page so we have 4 instructions per page. We label the instructions:

```
        load $1, 0          // i1   P0
```

```
loop:  bge $1, 4, end        // i2   P0
       lw $2, ($5)           // i3   P0
       add $2, $2, 5         // i4   P0
       sw $2, ($5)           // i5   P1
       add $5, $5, 4         // i6   P1
       add $1, $1, 1         // i7   P1
       j loop                // i8   P1
end:   .. Irrelevant code .. // i9   P2
```

For code, there will be a total of $7 \times 4 + 1 + 1 = 30$ instructions executed, i.e. 30 memory accesses. Of this the first accesses to P0 and P1 will trigger TLB misses. The TLB miss rate is then $\frac{2}{30}$ and the average translation time is $\left(1 - \frac{2}{30}\right) \times 10 + \frac{2}{30} \times 100 = 16\ ns$

For data: The code only reads/writes 4 words, all of which fit into a single page. There are 8 accesses (4 reads and 4 writes), of which only 1 triggers a TLB miss.

ATT = $\left(1 - \frac{1}{8}\right) \times 10 + \frac{1}{8} \times 100$

= 21.25 ns

There are 8 memory accesses and 30 code acceses giving 38 accesses, so we take a weighted mean of the two averages to find our final average:

$$ATT = \frac{30 \times 16 + 8 \times 21.25}{30 + 8} = 17.11\ ns$$

Variations considered:

- **Students counting loading the irrelevant code i9 (no penalty):**
  - **Code = $7 \times 4 + 1 + 1 + 1 = 31$ memory accesses**
  - **PFs = 3**
  - **TIme = $\frac{3}{31} \times 100 + \left(1 - \frac{3}{31}\right) \times 10 = 18.71\ ns$**
  - **Average = $\frac{31}{31+8} \times 18.71 + \frac{8}{31+8} \times 21.25 = 19.23\ ns$**
  - **Incorrect averaging (-3 marks = 1 mark) $\frac{18.71 + 21.25}{2} = 19.98\ ns$**
- **Students using 100 iterations rather than 4 (-1 mark = 3 marks):**
  - **Code: $7 \times 100 + 1 + 1 = 702$**
    - **2 PF giving us = $\left(1 - \frac{2}{702}\right) \times 10 + \frac{2}{702} \times 100 = 10.26\ ns$**
  - **Data: 100 read/writes will be spread across $\frac{100}{4} = 25\ pages$. So there will be 25 page faults for 200 reads.**

- 25 PFs give us $\left(1-\frac{25}{200}\right)\times 10+\left(\frac{25}{200}\right)\times 100=21.25\ ns$
- Total memory accesses = 902
  - Average = $\frac{702}{902}\times 10.26+\frac{200}{902}\times 21.25=12.7\ approx$
- Incorrect averaging: (-3 marks = 1 mark) $\frac{10.26+21.25}{2}=15.76\ ns$

Incorrect solutions: Average out the two timings using $\frac{16+21.25}{2}=\ 18.63\ ns$.
Penalty: -3 marks = 1 mark

Question 12. File Systems (10 MARKS)

We have a filesystem on a solid-state drive (SSD) with 1TiB (1 TiB = 1024 GiB, 1 GiB = 1024 MiB, 1 MiB = 1024 KiB, 1 KiB = 1024 bytes) capacity, with blocks of 32 KiB. This filesystem uses a variation of FAT with block pointers large enough to index all blocks in the drive.

a. (1 marks) How many disk blocks in total are there on our drive? Express your answer in powers of 2 (e.g. 2^{16})

   1 TiB = 2^{40} bytes
   Number of blocks = $\frac{2^{40}}{2^{15}}=2^{25}$ blocks

b. (2 marks) What is the minimum size of a block pointer for this filesystem in bytes? Again note that it doesn't make sense to have fractions of a byte, e.g. 3.23 bytes.

   We need minimally a 25 bit pointer = 3.125 = 4 bytes

c. (1 marks) Given your answer in b., how large would your FAT be? Express your answer in MiB (1 MiB = 1024 KiB, 1 KiB = 1024 bytes).

   There are 2^{25} blocks each pointer is 4 bytes, so total size = $2^{25}\times 2^{2}=2^{27}=128\ MB$

Memory is precious, and loading the entire FAT is wasteful. The FAT, in any case, is stored in disk blocks, and we will keep only one disk block's worth of FAT entries in memory. FAT blocks are loaded on demand. Assume that the first few bytes of each block contain the starting and ending indexes for the block. This idea is shown below for the FAT entries in one disk block, in this case for blocks 3752, 3753, 3754, … , 4248, 4249 and 4250.

First index	Last index	FAT entry	FAT entry	...	FAT entry
3752	4250	Entry for 3752	Entry for 3753	...	Entry for 4250

d. (2 marks) How many FAT block pointers can be accomodated into a single disk block?

A single disk block = 32 KiB, of which the first 8 bytes are used by the first and last index.

Remaining bytes = 32760
# of pointers per block = 8190 pointers.

If students somehow used the example above instead of calculating (-1 mark):
# of blocks = 4250 – 3752 + 1 = 499 blocks.

e. (4 marks) Suppose we have a file occupying (in sequence) the following 11 blocks 1, 2, 2341, 2342, 9875, 10305, 25501, 25502, 75, 76 and 2000. How many disk accesses are needed in total (i.e. reading the FAT blocks and the actual data blocks) for this file?

FAT Block	Start Index	End Index
0	0	8189
1	8190	16379
2	16380	24569
3	24570	32760

1, 2, 2341 and 2342 are from the same FAT block. 1 FAT read
9875, 10305 are from the same FAT block. 1 FAT read
25501, 25502 are from the same FAT block, 1 FAT read
75, 76 and 2000 are from the same FAT block, 1 FAT read
Total = 4 FAT reads
# of block reads = 11 + 4 = 15 block reads.

Solution using 499 blocks (no penalty):

Block Access	FAT Block $\lfloor \frac{block\_num}{499} \rfloor$	FAT Load?
1	0	Y (1)
2	0	N
2341	4	Y (2)
2342	4	N
9875	19	Y (3)
10305	20	Y (4)
25501	51	Y (5)
25502	51	N
75	0	Y (6)
76	0	N
2000	4	Y (7)
Total Loads		7

Total block reads = 11 + 7 =1 8