CMPU 334 Exam 1
Spring 2020

Name: Master Key

Instructions:

This is a closed book, closed notes exam. No electronic devices, including calculators are allowed. You have 75 minutes. There are 12 problems and 8 pages to this exam.

Good luck!

1 (8)	
2 (8)	
3 (8)	
4 (8)	
5 (8)	
6 (8)	
7 (8)	
8 (10)	
9 (8)	
10 (8)	
11 (8)	
12 (10)	
Total (100)	

1. **The purpose of the operating system.**
   a. What are two roles the operating system plays in order to make application programmers' jobs easier?

      1) a virtual machine

      2) a standard library

   b. What are two (often conflicting) goals/requirements for the OS while running programs?

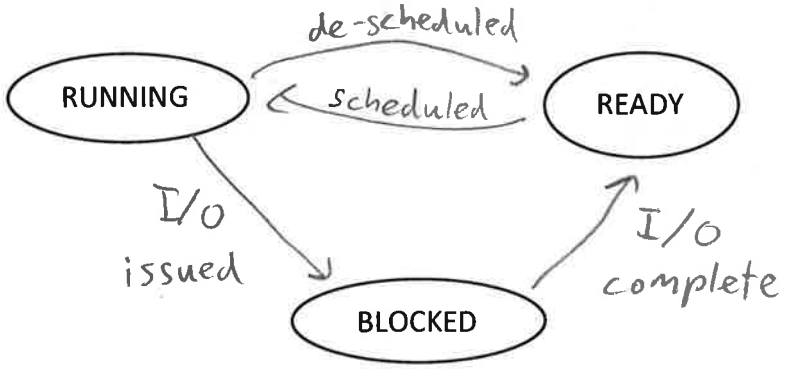
      1) efficiency

      2) safety / control / isolation / security

"how" "which" → policy

2. **Mechanisms vs. Policies.** Circle the items that correspond to mechanisms below:
   a. The OS uses LRU to select a page to evict upon memory pressure
   b. Hardware raises a page fault for a memory access
   c. The OS inserts a new TLB entry upon a (software-managed) TLB miss
   d. The OS schedules processes with Round Robin → policy

3. **The States of a Process.** Draw the arrows to complete the process state machine. Label each arrow with a word or two describing what could cause the transition. Be sure to leave out arrows for transitions that never occur.



4. **Scheduling Metrics.** The following 3 jobs (with their running times listed) arrive at time 0 in the following order:
   Job 0 (length = 9)
   Job 1 (length = 1)
   Job 2 (length = 4)

   Compute the response time and the turnaround time for the following scheduling algorithms. Assume the RR order is the same as the order in which the jobs are listed and a time quanta of 1.

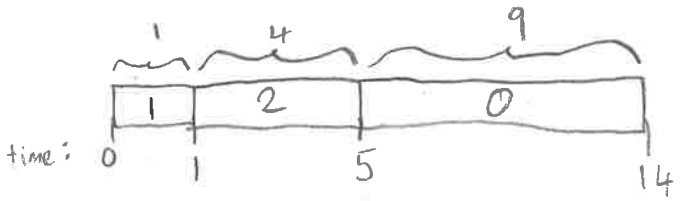
SJF (Shortest Job First)

first run − arrival

complete − arrival

schedule : 1, 2, 0

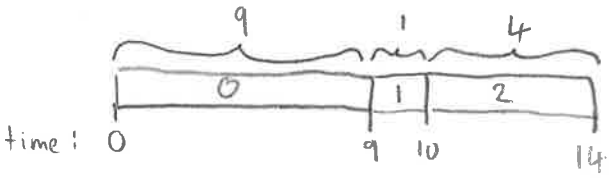
Job	Response Time	Turnaround Time
0	5−0 = 5	14−0 = 14
1	0−0 = 0	1−0 = 1
2	1−0 = 1	5−0 = 5



FIFO (First In First Out)

schedule : 0, 1, 2

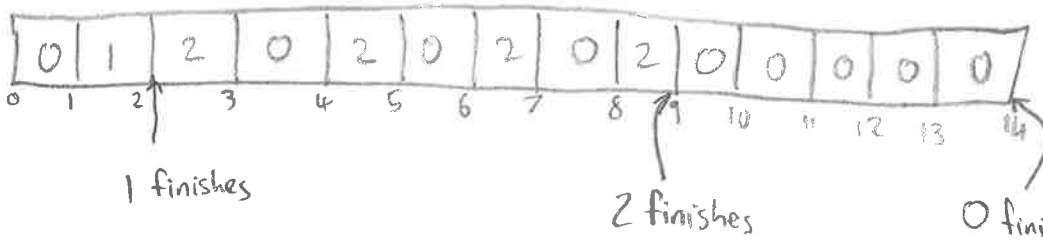
Job	Response Time	Turnaround Time
0	0−0 = 0	9−0 = 9
1	9−0 = 9	10−0 = 10
2	10−0 = 10	14−0 = 14



RR (Round Robin)

Job	Response Time	Turnaround Time
0	0−0 = 0	14−0 = 14
1	1−0 = 1	2−0 = 2
2	2−0 = 2	9−0 = 9

Schedule :



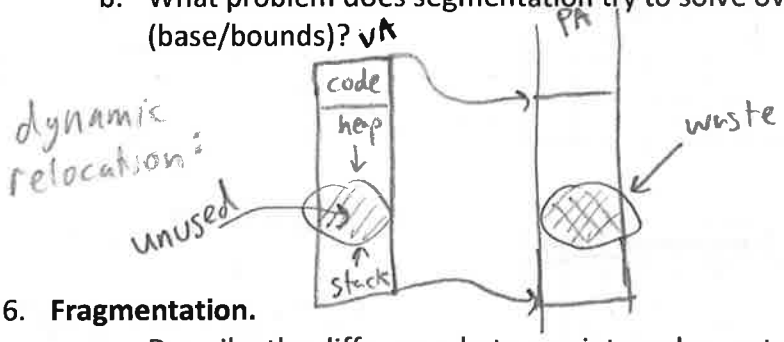
5. Address Translation.

a. How many virtual memory accesses must be translated for the following short program?

```
1. load  0x2000, %r1
2. inc   %r1
3 store %r1, 0x2000
```

2 load/stores
+ 3 instruction fetches
5

b. What problem does segmentation try to solve over simple dynamic relocation (base/bounds)?



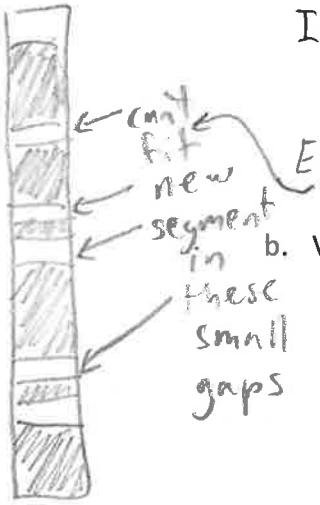
Eliminate wasted space by relocating each segment independently

6. Fragmentation.

a. Describe the difference between internal vs. external fragmentation. Give an example of when the OS may encounter each.

Internal: unusable space within allocated objects →
e.g. large pages

External: unusable space between allocated objects
e.g. variable-sized segments



b. Very large pages

i. What type of fragmentation could occur with large page sizes?

internal

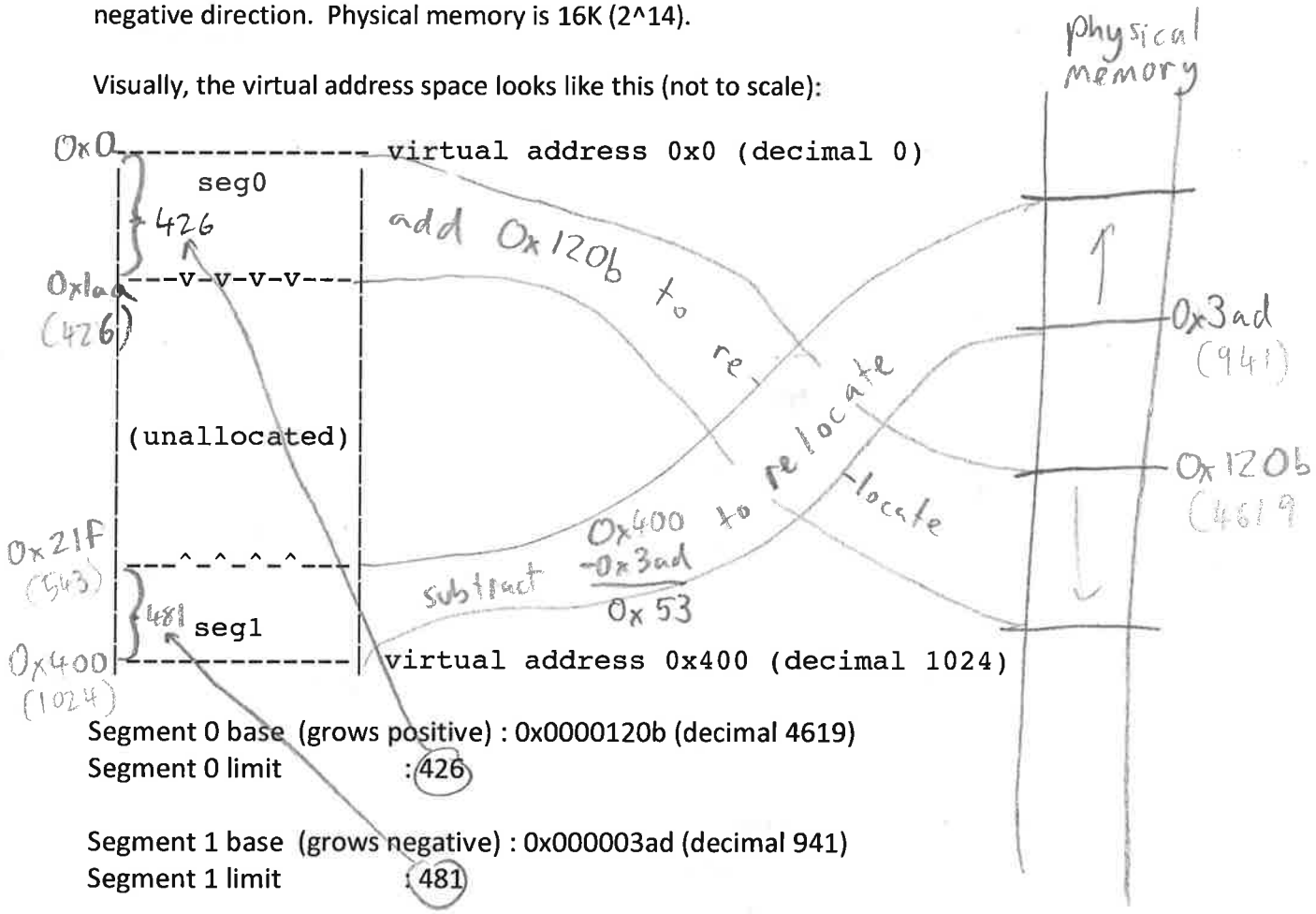
ii. What are two advantages to using a large page size?

1) smaller page table

2) better TLB coverage

7. **Address translations with segmentation.** For this problem, you should assume a simple 1K (2^10) virtual address space with only two segments. The top bit of the 10 bit virtual address is used to check which segment the virtual address is in: 0 for segment 0 (where, say, code and the heap would reside) and 1 for segment 1 (where the stack lives). Segment 0 grows in a positive direction (towards higher addresses), whereas segment 1 grows in the negative direction. Physical memory is 16K (2^14).

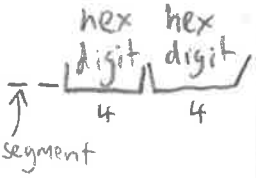
Visually, the virtual address space looks like this (not to scale):



Segment 0 base (grows positive) : 0x0000120b (decimal 4619)
Segment 0 limit : 426

Segment 1 base (grows negative) : 0x000003ad (decimal 941)
Segment 1 limit : 481

Each VA is 10 bits long



For each VA below, write which segment the VA is in (0 or 1) and the physical address (PA) that it translates to. If the VA is out of bounds, write SEGFAULT instead of the PA.

VA 0x0b4 (decimal: 180) --> SEG 0 PA 0x12bf
VA 0x269 (decimal: 617) --> SEG 1 PA 0x216
VA 0x341 (decimal: 833) --> SEG 1 PA 0x2ee
VA 0x1ee (decimal: 494) --> SEG 0 PA Segfault
VA 0x092 (decimal: 146) --> SEG 0 PA 0x129d

0x 0b4
+0x120b
12bf

0x 269
-0x 53
0x 216

0x341
-0x 53
0x2ee

0x092
+0x120b

which segment?

bounds check

8. **Address Translation with Linear Page Tables.** In this problem we will assume a 16K (2^14) address space, 64K (2^16) physical memory size, and 4K (2^12) page size.

→ bits in VA → bits in offset

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

8: 1000...
0: 0000...

→ valid bit

Page Table (from entry 0 down to the max size)

[0] 0x80000003 PFN = 0x3
[1] 0x80000006 PFN = 0x6
[2] 0x00000000 ← invalid
[3] 0x80000000 PFN = 0x0

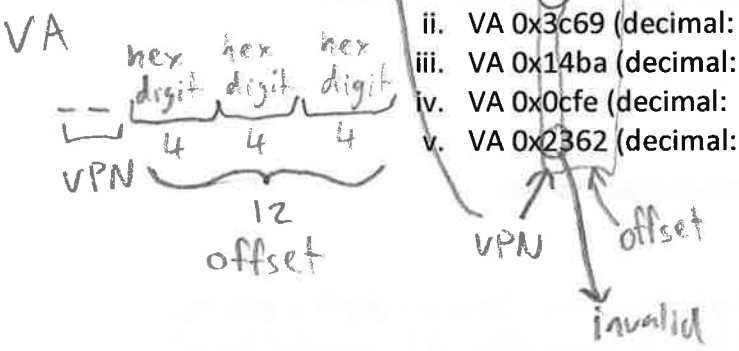
VPN

a) How many bits make up a virtual address (VA) 14
b) How many bits of the VA determine the offset 12
c) How many bits of the VA determine the VPN 2 (14-12)
d) For the following virtual addresses, write the physical address or INVALID if there is not a valid translation

→ invalid

PA = PFN | offset

i. VA 0x2dd1 (decimal: 11729) --> PA invalid
ii. VA 0x3c69 (decimal: 15465) --> PA 0x0c69
iii. VA 0x14ba (decimal: 5306) --> PA 0x64ba
iv. VA 0x0cfe (decimal: 3326) --> PA 0x3cfe
v. VA 0x2362 (decimal: 9058) --> PA invalid



9. **Smaller Page Tables**: Circle all that are possible strategies to reduce the size of page tables in memory

(a.) Use larger pages (e.g., 2MB) → more coverage with fewer PTEs
b. Use smaller pages (e.g., 512 KB)
(c.) Use more levels in the page tables → don't need to allocate lots of invalid PTEs
d. Use fewer levels in the page tables
(e.) Use an inverted page table → 1 PTE per physical page regardless of # processes
(f.) Use a separate page table per segment → hybrid approach avoids needing to allocate lots of invalid PTEs
(g.) Write page tables out to swap → user page tables can be allocated in kernel virtual memory

10. **Translation Lookaside Buffers.**

a. Why is the principle of *locality* in programs important for TLB performance?

The TLB is a cache of address translations. Locality means that a program's memory accesses (for code or data) will likely be near (on the same page) as other recent accesses. Thus the translation is likely to hit in the TLB, resulting in good performance

b. True or false: Circle the true statements below:

i. The primary goal of the TLB is to reduce the size of page tables
ii. The TLB is a cache of address translations → no, speed them up
iii. The TLB resides in main memory (RAM)
iv. Depending on the architecture, TLB misses can be served by hardware or software → no, the point is to avoid costly RAM access
v. TLB entries translate segment bases to bounds
vi. TLB entries translate VPNs to PFNs
vii. The TLB translates addresses generated by instruction fetches
viii. The TLB translates addresses generated by loads
ix. The TLB translates addresses generated by stores
x. The TLB evicts pages to disk

ALL memory accesses

11. Page Table Entries.

a. PTEs typically contain the PFN and some other bits that provide useful information to the OS. Give three examples of other bits one may find in a PTE.

1. valid bit 4. permission bits (r/w/x) 5. present bit
2. dirty bit (whether contents need to be written on eviction)
3. referenced bit (for e.g. clock algorithm)

b. When a page is swapped to disk, what happens to the corresponding PTE?

1. the present bit is cleared.
2. the PFN is replaced with the offset on the swap disk.

12. **Page Replacement Policies.**

evict page used furthest in the future

a. Assuming a replacement policy of OPT (optimal), and a cache of size 3 pages, figure out whether each of the following page references hit or miss in the page cache.

VPN Access	Hit/Miss (Circle one)	State of memory after access (3 pages)
0	Hit/Miss	0 - -
1	Hit/Miss	0 1 -
1	Hit/Miss	0 1 -
5	Hit/Miss	0 1 5
6	Hit/Miss	6 1 5
4	Hit/Miss	4 1 5
5	Hit/Miss	4 1 5
4	Hit/Miss	4 1 5
1	Hit/Miss	4 1 5
1	Hit/Miss	4 1 5

→ used furthest in future

→ used furthest in future

b. Assuming a replacement policy of LRU (least recently used), and a cache of size 3 pages, figure out whether each of the following page references hit or miss in the page cache.

VPN Access	Hit/Miss (Circle one)	State of memory after access (3 pages)
0	Hit/Miss	0 - -
1	Hit/Miss	0 1 -
1	Hit/Miss	0 1 -
5	Hit/Miss	0 1 5
6	Hit/Miss	6 1 5
4	Hit/Miss	6 4 5
5	Hit/Miss	6 4 5
4	Hit/Miss	6 4 5
1	Hit/Miss	1 4 5
1	Hit/Miss	1 4 5

→ least recently used

→ least recently used

→ least recently used

c. In what circumstances is LRU a poor policy choice?

Looping over more pages than fit in memory causes every new page to be a miss!