NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

FINAL ASSESSMENT PAPER - ANSWERS

AY2019/20 Semester 1

CS2106 – INTRODUCTION TO OPERATING SYSTEMS

December 2019 Time Allowed: 2 **hours**

INSTRUCTIONS

1. This question paper contains **THIRTYTHREE (33)** questions and comprises **FOURTEEN (14)** printed pages.
2. Maximum score is **100 marks** and counts towards 50% of CS2106 grade.
3. Write legibly with a pen or pencil. **MCQ form should be filled out using pencil.**
4. This is a **CLOSED BOOK** test. However, a single-sheet double-sided A4 reference sheet is allowed.
5. Write your **STUDENT NUMBER** below with a pen.

A								

Questions in Part B	Marks
26	**/12**
27-31	**/16**
32	**/12**
33	**/10**
Total	**/50**

Part A: MCQ questions.

Questions 1-23 from this section should be answered using the MCQ bubble form provided. Answers given in the paper will not be considered for grading. Questions have one correct answer.

1. [2 marks] Using many user-level threads within a process may cause one or more of the following problems:
   i. The chances of stack overflow increase because the threads share the stack.
   ii. Even with multiple cores, the program will not run faster because only one thread at a time will get to run when the process is scheduled.
   iii. Unix signals cannot be used in programs with multiple threads.

   Choose the problems that you might encounter:
   A. i.
   B. i. and ii.
   C. i. and iii.
   D. ii. and iii.
   E. All of the observe.

2. [2 marks] How many times will message "All good!" be printed by the following code fragment:

Line#	Code Snippets
1	`pid = fork();`
2	`if (pid == 0)`
3	`    execve("/bin/true", ["/bin/true"], NULL);`
4	`else`
5	`    execve("/bin/false", ["/bin/false"], NULL);`
6	`printf("All good\n");`

   A. 1
   B. 2
   C. 4
   D. "All good" is never printed.
   E. Code will not compile.

3. [2 marks] Assume you are trying to create a process using fork. Fork call is failing because you have too many zombie processes in the system. What can you do?
   A. Run a command in the shell interpreter to kill a zombie process.
   B. Run a command in the shell interpreter to kill an active process.
   C. Run a command in the shell interpreter to list the zombie processes.
   D. Reboot the system.
   E. Reinstall your operating system.

4. [2 marks] A program is running for the first time in an operating system and it is stopped. Later, the same program (process) is started again. We observe that the startup time the second time is much shorter than the first time. The **most important source of overhead** in starting for the first time is:

A. Starting for the first time, the executable file is first brought from disk to RAM. Second time, the executable sections are already in RAM.
B. The first time when the program is running, PCB needs to be initialized and that takes a long time. Second time, the PCB is already in RAM.
C. When starting to run the program will encounter many page faults for the data that is used in the program. Second time, there will not be new page faults.
D. The first time the TLB needs to be flushed and takes longer than the second time when the program gets to run.
E. Impossible to say.

5. [2 marks] The virtual address space supported is 2^{64} **bits** (not bytes). The page size is 1KiB (2^{10} bytes), the size of the physical memory (RAM) is 32KiB, the size of a page table entry is two bytes. Assuming the addressing is at the **byte** level, calculate the size of the page table required for both standard and inverted page tables (the size of the entry in the inverted page table is the same with the entry in the direct page table).
   A. 2^{54} + 32 bytes
   B. 2^{51} + 32 bytes
   C. 2^{54} + 64 bytes
   D. 2^{55} + 64 bytes
   E. 2^{52} + 64 bytes

   Answer:

Standard page table:
address space is 61 bits in byte addressing
number of pages = 2^61 / 1K = 2^51;
Page Table Size = 2^51 * (PTE size) = 2^52 bytes
Inverted page table:
Total frames = 32k / 1K = 32;
Page Table Size = 32* (PTE size) = 64 bytes

6. [2 marks] Consider a virtual memory system with the page table stored in memory, and no SWAP is used. If a memory access takes 200 nanoseconds, how long does a **paged memory reference** take?
   A. 200 ns
   B. 400 ns
   C. 600 ns
   D. More than 400 ns because the page table might not be in memory.
   E. 200 ns + time to service a page fault.
7. [2 marks] TLB is added to the system from Question 6. 75 percent of all page table references are found in TLB. Assume that finding a page-table entry in the TLB is instantaneous, and a memory access takes 200 ns. What is the **memory reference time** for this memory system?
   A. 200 ns
   B. 300 ns
   C. 400 ns
   D. 250 ns
   E. Impossible to say because we do not know the time taken to service a page fault.

   Answer: 75% * TLB hit-time + 25% * TLB miss-time = 75% * 200ns + 25% * 400ns = 250ns

8. [2 marks] The `mmap()` system call can be used to map a file to memory so that a program can use pointer operations to access its content. On a 32-bit Linux machine with 16GiB ($16 * 2^{30}$ bytes) of RAM and a 1 TiB (2^{40} bytes) disk, can you `mmap()` a 6GiB file stored using a FAT16 file system? (choose the point with the best justification)
   A. True because RAM is enough to store the file.
   B. True because disk space is enough to store the file.
   C. False because the file is not stored on contiguous blocks on the disk.
   D. False because the FAT16 file system cannot handle 6GiB files.
   E. False because the virtual address space is less than 6GiB.

   Answer: No you can't. On a 32-bit Linux machine, the user address space is less than 4GB, but mmap() requires the availability of a contiguous chunk of virtual address space.

9. [2 marks] Considering the same system from Question 8. Assume that now you are running on a 64-bit Linux machine with only 4 GiB of RAM. Could you `mmap()` the same file of 6GiB? (choose the point with the best justification)
   A. False because RAM is not enough to store the file.
   B. True because disk space is enough to store the file.
   C. False because the file is not stored on contiguous blocks on the disk.
   D. False because the FAT16 file system cannot handle 6GiB files.
   E. True because the virtual address space is more than 6GiB.

10. [2 marks] A program is processing large amount of data in memory. After all data has been read within the memory space of the process, the program appears to make much slower progress than expected, although there are no out-of-memory errors. You are running on a system with one core, but the CPU load is near zero. A suggestion is to increase the SWAP space size to better accommodate the program's virtual memory requirements. Will this suggestion help to improve the observed phenomenon? (choose the point with the best justification)
   A. Yes, because now there is enough space on disk to swap out pages.
   B. Yes, because the process does not need to swap anymore.
   C. No, because the program was able to use swap properly even before the increase.
   D. No, because the process cannot use the swap space at all.
   E. No, because the program suffers of stack overflow.

Answer:

The phenomenon described (large memory consumption, low CPU utilization, slow progress) is typical of thrashing – a state where processes spend most of their time paging data in and out from/to disk. The root cause is a shortage of physical memory. Therefore, adding swap space will not solve the problem. In fact, if a shortage of swap space had been an issue, the program would have seen out of memory errors (malloc() failing, or in Linux the OOM killer killing the program.)

11. [2 marks] Consider a buddy system used to implement a memory allocator (such as `malloc`).
   i. Allocation and free operations are performed in constant time (independent of the size of the memory available for allocation)

ii. Internal fragmentation is possible, but limited because the allocator only needs to round up to powers of 2.

Answer:

Internal fragmentation includes the difference between allocated block size and payload. The buddy allocator always rounds up to a multiple of 4KB, so the internal fragmentation is between 0 bytes and 4KB-1; for uniformly distributed payload sizes it would be, on average, ~2KB.

iii. External fragmentation is likely to arise due to limited coalescing ability of this allocator (for instance, it is unable to coalesce blocks of different sizes, if they are not buddy blocks).

The following statements are true:

A. i.
B. i. and ii.
C. ii. and iii.
D. ii. and iii.
E. All of the above.

12. [2 marks] Given 4 physical frames for a process, and LRU (last recently used) page replacement algorithm. After the following sequence of page accesses, what are the pages in RAM sorted from first frame to last frame?
Sequence: 3, 2, 4, 5, 5, 1, 7, 4, 7, 6, 5
A. 5, 7, 4, 6
B. 1, 7, 6, 5
C. 6, 7, 4, 5
D. 4, 7, 6, 5
E. 1, 7, 5, 6

13. [2 marks] Given 4 physical frames for a process, and FIFO (first in first out) page replacement algorithm. After the following sequence of page accesses, what are the pages in RAM sorted from first frame to last frame?
Sequence: 3, 2, 4, 5, 5, 1, 7, 4, 7, 6, 5
A. 5, 7, 4, 6
B. 1, 7, 6, 5
C. 6, 7, 4, 5
D. 4, 7, 6, 5
E. 5, 4, 7, 6

14. [2 marks] Given 4 physical frames, which of the page replacement algorithm(s) give the same number of page fault as the optimal page replacement algorithm (OPT) for the following memory reference string?
Sequence: 3, 2, 4, 5, 5, 1, 7, 4, 7, 6, 5
i. LRU
ii. FIFO

iii. CLOCK (assume that a page is not marked as referenced when the page is first brought from disk to memory).

A. iii.
B. i. and ii.
C. ii. and iii.
D. i. and iii.
E. All of the above.

15. [2 marks] A program dynamically allocates an array. Consider the following memory regions:
    i. OS memory region
    ii. Virtual memory space
    iii. Frames in RAM
    iv. SWAP space on disk (for non-resident memory)

    Choose where the array might be located:

    A. i., iii., iv.
    B. iii, iv.
    C. ii, iii., iv.
    D. iii, iv.
    E. i., ii.

16. [2 marks] The code handling a page fault is stored in the following memory region:
    A. OS memory region
    B. Virtual memory space of the program that caused the page fault.

17. [2 marks] Threads of the same process share the same file descriptor table.
    A. True
    B. False

18. [2 marks] A process is opening a file and creates multiple threads reading from the same file (file descriptor) using system call read. The threads will read the same content.
    A. True
    B. False

19. [2 marks] A hardlink HL to a file A located in folder B is created. Folder B is deleted.
    A. The hardlink HL becomes invalid because file A was removed.
    B. The hardlink is valid, but file A is removed.
    C. The hardlink is valid and file A still exists.
    D. The hardlink HL is removed as well when file A is removed.
    E. Folder B cannot be removed because HL to file A exists.

20. [2 marks] In general, the free space management on the disk is done using a bitmap or a linked list. Which statement is **FALSE?**
    A. FAT16 uses linked list to keep track of the free space on disk.
    B. EXT2 uses bitmaps in each block group to keep track of the free space on disk.
    C. Finding a free block when using a linked list is done in constant time O(1).
    D. Bitmap is more space efficient the linked list.

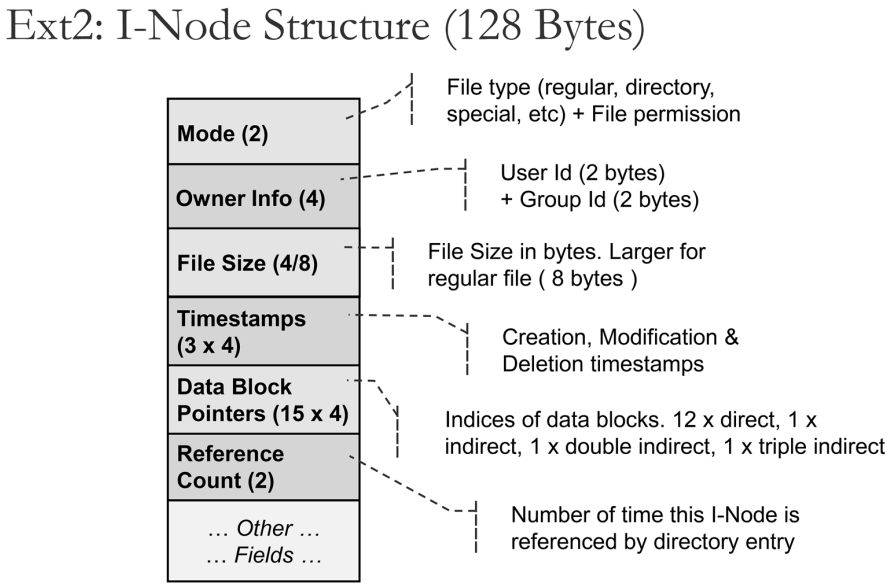
E. Updating the bitmap when a block is freed is done in constant time O(1).

21. [2 marks] A process creates multiple process, each process opens the same file and reads from the file (using read). What will each process read:
    A. Same information
    B. Different parts of the file, depending on the order the processes get to read.
    C. There is a race condition because the processes read from the same file.

22. [2 marks] A file A.txt uses 1000 data blocks on a FAT16 file system. Assuming that you know have already read the directory entry for A.txt, how many accesses to disk do you need to make to read the whole file?
    A. 2000 accesses to disk.
    B. 1001 accesses to disk only.
    C. 1000 accesses to disk and 1000 accesses to RAM.
    D. 1000 accesses to disk and 999 accesses to RAM.
    E. 1001 accesses to disk and 999 accesses to RAM.

23. [2 marks] You open twice the same file in a process. Assume the process is not closing any files during its execution. The **total number of entries** in the per-process file descriptors table is:
    A. 1
    B. Greater or equal to 2 (2 is a possible value)
    C. Greater or equal to 4 (4 is a possible value)
    D. Greater or equal to 5 (5 is a possible value)
    E. Impossible to give a number.

24. [2 marks] According to the lecture, i-node in ext2 file system contains the following information:



Consider system call `lseek`, with the following definition:

```
off_t lseek(int fd, off_t offset, int whence);
```

Description: **lseek**() repositions the file offset of the open file description associated with the file descriptor **fd** to the argument **offset** according to the directive **whence.**

Consider the following statements:

i. lseek call will modify one or more i-node fields.
ii. lseek call will modify the entry in the per-process file descriptors table for the process that calls it.
iii. lseek call will modify the entry in the system-wide open files table.

Which statements are TRUE?

A. i.
B. ii.
C. i. and iii.
D. iii.
E. All of the above.

25. [2 marks] Assume you are running `ls` in a shell interpreter on Linux with an EXT2 file system.

```
$ ls -i
```
Description: List information about the FILEs (the current directory by default). Print the index number (inode) of each file.

i. Inode bitmap
ii. Inode table
iii. Directory entries for each file in the directory
iv. Inode for each file in the directory
v. Data blocks for each file in the directory.

What information from the above is "`ls -i`" reading?

A. i., ii, iii. and iv.
B. iii.
C. iii. and iv.
D. ii., iii. and iv.
E. iii., iv, and v.

Part B. Short Questions

26. [12 marks] Semaphores can be used to express constraints between tasks performed by different threads. Synchronize the execution for the tasks shown in the graph in Figure 1. A vertex in the graph represents a task (executed by a thread), and the edges represent the dependencies among tasks (e.g. task B executes only after task A finishes execution).

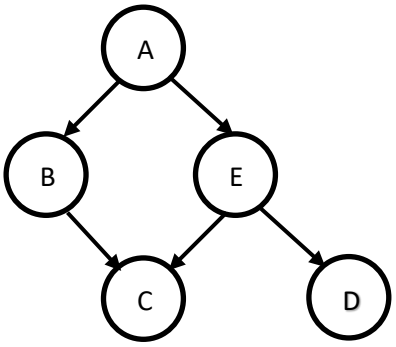


Figure 1: Task dependency graph

Complete the program below to ensure these constraints!

Answer:

Line#	Code snippets

1
2
3
4
…
 	
	`// declare any semaphores you need here;`
	`// be sure to show how to initialize them`
	`// initial value of all semaphores is 0,`
	`// for proper code see main function; shorthand is accepted`
	`sem_t a_done, b_done, e_done;`
	`sem_init(&a_done, 0, 0);`
	`sem_init(&b_done, 0, 0);`
	`sem_init(&e_done, 0, 0);`

A.1.
A.2.
A.3.
…
	`static void *thread_A(void *_) {`
	`printf("A\n");`
	`sem_post(&a_done);`
	`sem_post(&a_done); }`

B.1.
B.2.
B.3.
…
	`static void * thread_B(void *_) {`
	`sem_wait(&a_done);`
	`printf("B\n");`
	`sem_post(&b_done); }`

C.1.
C.2.
C.3.
…
 	
	`static void *thread_C(void *_) {`
	`sem_wait(&b_done);`
	`sem_wait(&e_done);`
	`printf("C\n");`
	`sem_post(&c_done);`
	`}`

D.1.
D.2.
D.3.
	`static void *thread_D(void *_) {`
	`sem_wait(&e_done);`
	`printf("D\n");`

…	}

E.1.
E.2.
E.3.
…
	static void * thread_E(void *_) {
	 sem_wait(&a_done);
	 printf("E\n");
	 sem_post(&e_done);
	 sem_post(&e_done);
	}

M.1.
M.2.
M.3.
M.4.
M.5.
M.6.
M.7.
M.8.
M.9.
M.10.
	int main()
	{
	 pthread_t t[N];
	 pthread_create(t+0, NULL, thread_E, NULL);
	 pthread_create(t+1, NULL, thread_D, NULL);
	 pthread_create(t+2, NULL, thread_C, NULL);
	 pthread_create(t+3, NULL, thread_B, NULL);
	 pthread_create(t+4, NULL, thread_A, NULL);
	 pthread_exit(0);
	}

4 marks: some synchronization is done, but insufficient to represent the dependency graph

8 marks: some synchronization issues are present

27. [4 marks] Explain an advantage and a disadvantage of two-level paging.

    Answer:

    Advantage: Smaller page tables that fit in a page in RAM.

    Disadvantage: A memory access requires multiple accesses to RAM to read the page table, and it might possibly page fault as well.

    In priority order of mark deduction:

    -1 mark if advantage and disadvantage are contradictory

    -1 mark to advantage section if advantage is not specific enough (e.g. "overhead is less").

    -1 mark to disadvantage section if disadvantage is not specific enough

28. [4 marks] The Google Android Operating System uses a Linux kernel to support its applications. Each application runs in its own process. Android devices do not typically have a disk, and they (typically) do not use SWAP to stretch the amount of available

physical memory when they run short. Instead, the OS may terminate processes, requiring the applications to store and restore their state when this happens. Under these circumstances, would Android still derive any benefits from Linux's virtual memory capabilities? Justify your answer by using a two examples where virtual memory (without SWAP) brings advantages.

Answer:

Even though Android does not typically swap, it still benefits from the other services virtual memory provides: for instance separate address spaces, which protect and isolate applications from one another, or the ability to share memory between processes so that the Dalvik runtime environment needs to be loaded only once.

Many answers given here.

In priority order of mark deduction:

No marks given if answer focus on talking about why not having SWAP is an advantage (without mention of virtual memory)

2 marks given for each of these mentioned: separate address spaces for programmer ease, added security, shared memory implementation

29. [2 marks] Modern operating systems use on-demand paging in which a process's code and data is not brought in until it is needed. In Unix, what is the very first page fault a process encounters after it makes a successful `exec()` system call?

    Answer:

    The very first page fault will be at the new program's entry point in its text segment where the new program starts executing and fetching instructions (typically a routine called 'start' which calls 'main') That's when the system will load the executable from disk (unless it's already in use by some other process).

    In priority order of mark deduction:

    -1 mark if "data" segment is also mentioned (e.g. "code AND data")

    2 marks given if only "text" segment is mentioned

30. [2 mark] A process is opening a file and continues to execute its code. Later, the process closes the file and finishes its execution, but the entry in the open file table is not freed. Explain why.

    Answer:

    The process had a fork after opening the file.

In priority order of mark deduction:

0 marks given if there is mention of “two processes happening to have opened the same file at the same time” i.e. indicating that is purely a temporal effect

-1 mark if student mentions the concept of “two processes pointing to the same entry” without mentioning fork()

0 marks for any other explanations

31. [4 marks] Implement an atomic increment using the following GCC atomic function:

```
bool _sync_bool_compare_and_swap (int* t, int r, int n)
```

Description: Atomically compare a referenced location `t` with a given value `r`. If equal, replace the contents of the location with a new value n, and return 1 (true), otherwise return 0 (false).

You are allowed to use busy waiting.

Answer:

Line#	Code Snippet

1
2
3
4
5
6
	`int  atomic_inc(int* t)`
	`{`
	`  do {`
	`  int s = *t+1;`
	`  } while (!_sync_bool_compare_and_swap(t,s-1,s));`
	`  return s;`
	
	`}`

In priority order of mark deduction:

No marks deducted for simple syntax errors (& used to deference instead of *, for e.g.)

0 marks given if answer is _sync_bool_compare_and_swap(*t, *t, *t + 1) or anything equivalent involving 3 ts in the compare and swap

-1 mark if return statement is missing

-2 marks instead if function does not work due to one non-return-related error

-3 marks instead if function does not work due to two non-return-related errors

0 marks given if more than two errors cause function to not work

32. You are required to implement and optimize the storage and search of multiple fixed file sizes of 1MB. Each file has a unique numeric identifier, and it is saved on disk using ext2 file system. The number of files is unlimited (say, 10 million), but it cannot grow beyond the disk size (maximum 10TB). We want to optimize this file system to store and search fast for files by identifier. You should use of some of the methods studies under file system implementation for your optimizations.
    a. [4 marks] First, assume that all files are stored in a directory. What would be the main disadvantage when we want to add a new file? What would be the main disadvantage when searching for a file by identifier?
    b. [2 marks] Assume that you do not have any restrictions in terms how many directories you can use. How would you mitigate the disadvantages shown at point a. for storing a file? Explain your optimizations.
    c. [2 marks] How would you mitigate the disadvantage(s) shown at point a. for searching a file by identifier? Explain your optimizations.
    d. [4 marks] You are allowed to use additional data structures to speed up the storing and searching of files. What data structures would you use to implement your optimizations? Be specific about what you store in each data structure.
    e. [2 marks] Explain how your suggestions (b, c, d) improve the storage and search times for a file. Estimate the overhead in terms of space and time of creating, maintaining, and using the additional data structures from point d.

Note: You can answer points a.-e. at the same time (in a different order). If you do that, make sure that you touch on all the questions we have under each point.

Answer:

a. For storing a new file: need to traverse the whole directory (all DE in the linked list) For searching, possibly need to search through the whole linked list.
b. Store a limited number of entries (X) in each directory. Use a hierarchical structure similar to what EXT2 is using.
c. Use a hash table to map the file identifier to the directory in which it is located. The hash table can be stored in memory because its size is: #files * (file_id_size + pointer_to_directory_block). 10 million files would occupy only limited size in RAM.
d. Hash table + hierarchical structure.
   The directory structure can be similar to direct and indirect blocks in ext2:
   - Direct folders contain X files
   - Single indirect – contains X other folders, each with X files.
   - Double indirect - …
e. Access time to a file depends on the on X and the number of indirect folders. The access time for a file is X ($\log_X N + 1$) . If N=10 million, we can choose X to minimize the access time.

2 marks: almost no good points

4 marks: a. has good points, but does not mention the directory entries all list + propose hash map or multiple directories to be used for storying the files

6 marks: a. is fine and one of the hash map or multiple directories is mentioned

8 marks: 6 marks + both hash and multiple directory ideas are mentioned

10 marks: 8 marks + overhead explained or mentions the idea of hierarchical directories.

In this problem, it is not enough to use:

- A binary search tree as a hierarchical directory structure. Maintaining the binary search tree would have a huge overhead.
- Split the file in folders based on their first digit in the identifier. This structure can be hierarchical or linear. If it hierarchical, the overhead of traversing such a structure is too high when you have very few files.
- Maintain the files in a sorted order in the directory as the overhead of insertion sort in the list of directory entries is too high

There is no guarantee the identifiers are distributed uniformly to bins (based on first digits).

33. [12 marks] Examine the code below.

Line#	Code Snippets
1	`int A = 2;`
2	`int B;`
3	`int C = 20;`
4	`int D;`
5	`void func(int X) {`
6	`    int r = X;`
7	`    int* list = malloc( A * sizeof(int) );`
8	`    B = r * r;`
9	`    list[0] = list[0] + C;`
10	`    D = list[0] + list[1];`
11	`}`

During linking/loading, different parts of the code above are mapped to different memory sections in the address space of the program. In the table below, place the variable name whenever the given line of code is related to the corresponding memory section. (It's possible for a single line to relate to more than one memory section.)

Answer:

Line #	Data	Stack	Heap
1	A		
2	B		
3	C		
4	D		
5		X	
6		X, r	
7	A	list	list*
8	B	r	
9	C	list	list[0]
10	D	list	list[0,1]

Grading was done on variable:
8 variables – 1 mark each (no points deducted if the variable is not mentioned the second time).
1 mark for adding list in both stack and heap (no points deducted for notation)
1 mark for not adding A,B, C,D in any other category
-1 mark for adding func

--End of paper--