NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

FINAL ASSESSMENT PAPER

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 **SIXTEEN (16)** 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 2B 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. Do not write your name.

A								

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

Part A: MCQ questions.

Questions 1-25 from this section should be answered using the MCQ bubble form provided (OCR form). Answers given in the paper will not be considered for grading. Each question is 2 marks.

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 space.
   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 above.

2. [2 marks] How many times will message "All good!" be printed by the following code fragment (assume `ls` is located in `/bin`):

Line#	Code Snippets
1	`pid = fork();`
2	`if (pid == 0)`
3	`    execl( "/bin/ls", "ls", "-l", NULL);`
4	`else`
5	`    execl( "/bin/ls", "ls", "-l", NULL);`
6	`printf("All good!\n");`

   A. 1
   B. 2
   C. 4
   D. "All good!" is never printed.
   E. Code snippet 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. What is the **most important source of overhead** in starting for the first time?
   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** (note 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

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 **average memory access 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.

8. [2 marks] The `mmap()` system call can be used to map a file to memory such that a program can use pointer operations to access its content. On a 32-bit Linux machine with 4GiB ($4 * 2^{30}$ bytes) of RAM and a 1 TiB (2^{40} bytes) disk, can you `mmap()` a 6GiB file stored using a EXT2 file system? (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 EXT2 file system cannot handle 6GiB files.
   E. False because the virtual address space is less than 6GiB.

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 EXT2 file system cannot handle 6GiB files.
   E. True because the virtual address space is more than 6GiB.

10. [2 marks] A program is processing a 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 almost 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 likely 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.

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 (for a given 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.
    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 (memory reference string), 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 (memory reference string), 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 algorithms give the same number of page fault as the optimal page replacement algorithm (OPT) for the following memory reference string (sequence)?

    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 HL is valid, but content of file A is removed.
    C. The hardlink HL is valid and content of 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.

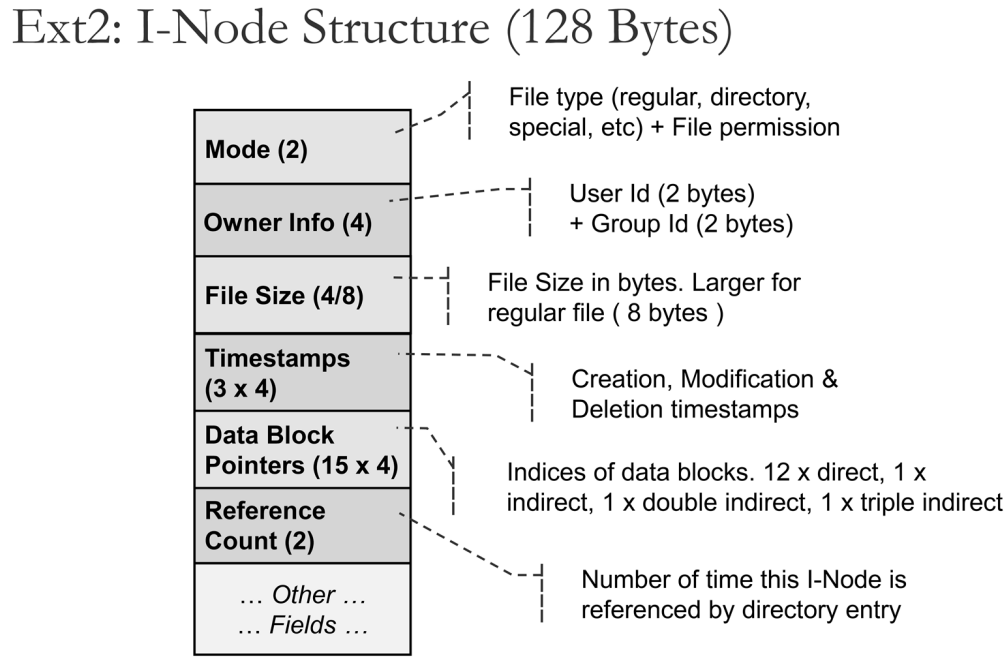
20. [2 marks] In general, the free space management on the disk is done using a bitmap or a linked list. Which statement is **FALSE?**
    A. FAT16 uses linked list to keep track of the free space on disk.
    B. EXT2 uses bitmaps in each block group to keep track of the free space on disk.
    C. Finding 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 processes; next, 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 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] A process opens twice the same file. 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 description:

`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 (inode) number of each file.

Consider the following structures in the EXT2 file system:

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 structures 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. Assume that semaphores can only take values equal or greater than zero. Synchronize the execution for the tasks (A-E) 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).

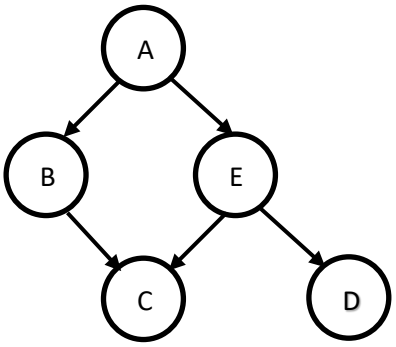


Figure 1: Task dependency graph

Complete the program below to ensure the constraints from Figure 1. (points are not deducted for minor issues in syntax).

Answer:

Line#	Code snippets
1 2 3 4 …	`// declare any semaphores you need here;` `// be sure to show how to initialize them`

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

B.1.
B.2.
B.3.
…
	`static void * thread_B(void *_) {`
	
	`printf("B\n");`

```
}
```

```
C.1.  static void *thread_C(void *_) {
C.2.
C.3.
…
          printf("C\n");
      }
```

```
D.1.  static void *thread_D(void *_) {
D.2.
D.3.
…
      printf("D\n");
      }
```

```
E.1.  static void * thread_E(void *_) {
E.2.
E.3.
…
      printf("E\n");
      }
```

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

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

Answer:

28. [4 marks] The 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 giving two examples where virtual memory (without SWAP) brings advantages.

Answer:

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 Linux, what is the very first page fault a process encounters after it makes a successful `exec()` system call?

    Answer:

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 system-wide open file table is not freed. Explain under which circumstances the previous situation might happen.

    Answer:

31. [4 marks] Implement an atomic increment function 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).
    `atomic_inc` function should return the incremented value of referenced location `t`. You can use busy waiting.

Answer:

Line#	Code Snippet
1	`int  atomic_inc(int* t)`
2	`{`
3	
4	
5	
6	
…	
	`}`

32. You are required to implement and optimize the storage and search of multiple fixed size files of 1MiB. Each file has a unique numeric identifier, and it is saved on disk using EXT2 file system. The number of files is unlimited, but it cannot grow beyond the disk size (maximum 10TiB). 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. [2 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:

33. [10 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	`}`

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

Answer:

Line #	**Data**	**Stack**	**Heap**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

—End of paper—