NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

FINAL ASSESSMENT PAPER

AY2020/21 Semester 1

CS2106 – INTRODUCTION TO OPERATING SYSTEMS

SOLUTION

November 2020 Time Allowed: **2 hours**

INSTRUCTIONS

1. Submit by **Wed, 25 Nov, 7:30pm** on **Luminus Files->Student Submissions -> Exam** a PDF file:
   - Name your PDF file using your student number: A1234567Z.pdf
   - The format of the answers has to **strictly** follow the format provided in the question.PDF file**. For example, if Question 23 (requirement and answer space) is on page 8 in the questions PDF file, it should be on page 8 in the answers PDF file submitted by you.**
   - In case you scan your answers, make sure that they can be read once included in the PDF. We will not grade answers that cannot be read.
   - When multiple submissions are made, we will grade only your most recent submission.

If Luminus is not accessible, submit your PDF using this form: https://forms.gle/fv9U4yLidycwsWya6

2. To solve the questions. You can either:
   - write your answers on the PDF directly,
   - print out, hand-write your answers, and scan into a PDF
   - hand-write your answers on clean paper, and scan into a PDF
   - write your answers in the DOCX, and save as PDF
   - have any other approach that would result in a PDF file
   - if you scan your answers, you may use Drive application on Android to add a new document, and choose Scan.
3. This assessment paper contains 42 questions and comprises FIFTEEN (15) pages. The total number of marks in the paper is 80 and the paper counts towards 40% of your final grade.
4. Students are required to answer **ALL** questions within the space in this file.
5. Failing to submit your answers to Luminus folder means you failed your exam (0 marks will be awarded). No late submission is allowed.
6. Address any questions you might have in Zoom chat or call 65168850.
7. Write your student number below. Do not write your name.

STUDENT NO: _____

Section 1: MCQ questions.

1. [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
2
3
4
	`pid = fork();`
	`if (pid == 0)`
	`    execl( "/bin/ls", "ls", "-l", NULL);`
	`printf("All good!\n");`

   A. 1
   B. 2
   C. 4
   D. "All good!" is never printed.
   E. The code snippet will produce a compilation error (when part of a program).

Answer: A: 1

2. [2 marks] Assume you are trying to create a shared memory space to be used by two processes. `shm_open` is failing because you have too many files open by the process. What can you do to make `shm_open` call succeed (choose the least invasive option)?
   A. Close a file before calling `shm_open`.
   B. Once another process closes a file, `shm_open` will succeed.
   C. Use `shm_unlink` to destroy a shared memory object.
   D. Reboot the system.
   E. Reinstall your operating system.

Answer: A: Close a file before calling `shm_open`.

3. [2 marks] Assume you are trying to create a process using `fork`. `fork` call is failing because you have too many orphan processes in the system. What can you do (choose the least invasive option)?
   A. Run a command in the interpreter to terminate a child process of process `init`.
   B. Terminate all zombie processes.
   C. Write a script to terminate an orphan process.
   D. Reboot the system.
   E. Reinstall your operating system.

Answer: A: Run a command in the interpreter to terminate a child process of process `init`. & D: Reboot the system.

A rationale: kill is usually shell builtin, so no new process and killing a few processes is definitely less invasive than taking the whole system down.

D rationale: according to the lecture, a command in shell should create a new process and that would not be possible for this system.

4. [2 marks] In a 32-bit system, a program is calling `malloc()` in a loop to allocate chunks of 1MiB (2^{20} bytes). Assume byte addressing. After how many loops will `malloc()` return NULL?
   A. `malloc()` will never return NULL because the space for heap can grow indefinitely.
   B. `malloc()` will never return NULL because pages are added to accommodate the need for more memory space.
   C. `malloc()` will never return NULL because `mmap()` is always successful.
   D. `malloc()` will return NULL before 4096 calls.
   E. It depends on how many other processes run at the same time in the system.

Answer: D: `malloc()` will return NULL before 4096 calls.

5. [2 marks] A computer supports 64-bit virtual memory addresses. The page size is 1KiB (2^{10} bytes), the size of a page table entry is two bytes. Assuming the addressing is at the word (8 bytes) level, calculate the size of the standard page table (in bytes).
   A. 2^{58}
   B. 2^{57}
   C. 2^{54}
   D. 2^{55}
   E. 2^{52}

Answer: A: 2^{58}

Explanation: Since the addressing is at word (8 bytes) level, the system can support $2^{64}*8$ bytes in total with 64-bit virtual address.
$(2^{64}*2^{3}*2^{1})/2^{10} = 2^{58}$

6. [2 marks] A computer supports 32-bit virtual memory addresses. The page size is 1KiB (2^{10} bytes), the size of a page table entry is four bytes. Assuming the addressing is at the byte level, calculate the overhead incurred by paging when using 2-level paging in a process that uses only three pages (in bytes).
   A. $2^{9} * 3 + 2^{16}$
   B. $2^{8} + 2^{14}$
   C. $2^{9} + 2^{2}$
   D. $2^{10} * 3 + 2^{16}$
   E. $2^{10} + 2^{16}$

Answer: E: $2^{10} + 2^{16}$

Explanation:
Number of PTEs that fit in a page = $2^{10}/2^{2}$
Number of entries in the page directory = $(2^{32}/2^{10})/(2^{10}/2^{2}) = 2^{14}$
Number of level-2 page tables needed = 1 (only 3 pages are used by the process.)
Total overhead = page directory size + level-2 page tables' size = $2^{14}*2^{2} + 1*2^{10} = 2^{16} + 2^{10}$

7. [2 marks] Consider a virtual memory system with the page table stored in memory (memory resident). If a memory access takes 200 nanoseconds, what is the longest time that a **paged memory reference** takes?
   A. 600 ns
   B. 600 ns + time to service a page fault
   C. 400 ns + time to service a page fault
   D. 200 ns + time to service a page fault
   E. More than 600 ns + time to service a page fault.

   *time to service a page fault includes the time to bring the page from SWAP to memory.

Answer: B: 600 ns + time to service a page fault & E: More than 600 ns + time to service a page fault.

Explanation: 1st access – page table; page fault (instruction is retires). 2nd access – page table; 3rd access – memory. Here we assume that non caching/TLB is involved ("longest time").

8. [2 marks] The page table base hardware register points to the page table of the process currently running on the CPU. This register needs to be updated when _____ takes place.

Answer: context switch

9. [2 marks] Given 5 physical frames for a process, and LRU (least 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, 1, 5, 7, 4, 7, 6, 3, 5
   A. 3, 4, 5, 6, 7
   B. 7, 6, 4, 3, 5
   C. 5, 4, 6, 7, 3
   D. 7, 6, 3, 5, 4
   E. 7, 6, 4, 5, 3

Answer: E: 7, 6, 4, 5, 3

10. [2 marks] Given 5 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, 1, 5, 7, 4, 7, 6, 3, 5
    A. 3, 7, 6, 5, 1
    B. 1, 3, 5, 6, 7
    C. 7, 6, 3, 5, 1
    D. 5, 1, 7, 6, 3
    E. 7, 6, 4, 5, 3

Answer: C: 7, 6, 3, 5, 1

11. [2 marks] Given 5 physical frames, which of the page replacement algorithms give the same number of page faults as the optimal page replacement algorithm (OPT) for the following memory reference string (sequence)?

    Sequence: 3, 2, 4, 5, 1, 5, 7, 4, 7, 6, 3, 5

    i. LRU
    ii. FIFO
    iii. CLOCK

    A. iii.
    B. i. and ii.
    C. ii. and iii.
    D. i. and iii.
    E. None of the above.
    F. All of the above.

Answer: E: None of the above.

12. [2 marks] A program maps a file to memory using `mmap`. Consider the following memory regions:

    i. Frames in RAM.
    ii. SWAP.
    iii. Disk blocks.

    Choose where the content of file might be located during the execution of the program:

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

Answer: A: i., iii. & E: All of the above.

Explanation:
A rationale: It will be in swap, if you map a file privately. Otherwise the kernel (or at least Linux) would just drop the pages and read it back from the file on disk.

E rationale: In general, the mapped file might be swapped from memory to disk.

13. [2 marks] The code handling a `fork()` system call is stored in the following memory region:

    A. OS memory region.
    B. Virtual memory space of the program that calls `fork()`.
    C. Frames of the process that calls `fork()`.

Answer: A: OS memory region.

14. [2 marks] A process is generating a page fault while accessing a memory region for which is does not have access rights. What happens in this case?

A. The operating system will handle the page fault and the process can continue the execution with the next instructions.
B. A signal SIGSEGV is sent to the process. If the operating system does not have a signal handler set for SIGSEGV, the process is terminated.
C. A signal SIGSEGV is sent to the process. The process might terminate.
D. A signal SIGSEGV is sent to the operating system. If the operating system does not have a signal handler set for SIGSEGV, the process is terminated.
E. The operating system will handle the page fault, and the process retries the instruction that caused the page fault.

Answer: C: A signal SIGSEGV is sent to the process. The process might terminate.

15. [2 marks] Throughout their execution, parent and child processes share ALL entries in the system-wide open file table, but different file descriptor tables.
    A. True
    B. False

Answer: False

16. [2 marks] When opening a file using `open`, the existing entry in the system-wide open file table is returned. If no such entry exists, a new entry is returned.
    A. True
    B. False

Answer: False

17. [2 marks] A process is started using an executable file. Immediately after starting, the physical memory occupied by this process is 8 KiB, even though the executable program's size is 630KiB (1kiB is 2^{10}B). This is observed because the operating system is using _____

Answer: demand paging

18. [2 marks] A symbolic link SL to a file `/home/e1234567/B/A` (given with full path) is created in the current path. SL is moved to the parent folder.
    A. SL becomes invalid.
    B. SL is valid, but it does not point to the correct file A.
    C. SL cannot be moved.
    D. When SL is moved, file A moves too.
    E. SL is valid and A can be accessed using SL.

Answer: E: SL is valid and A can be accessed using SL.

19. [2 marks] A process creates multiple threads (using pthread library); next, each thread opens the same file and reads 32 bytes from the file (using `read` system call). What will each thread read?

A. Same information (same bytes).
B. Different parts of the file, depending on the order the threads get to read.
C. There is a race condition because the threads read from the same file.
D. If the file is open for reading by another process, it is impossible to say.

Answer: A: Same information (same bytes).

20. [2 marks] A file A.txt uses 14 data blocks on a EXT2 file system. Assume you have already read the directory entry for A.txt and only the I-node for A.txt is cached in RAM. What is the number of accesses to disk/RAM do you need to make to read the whole file?
    A. 14 accesses to disk and 14 accesses to RAM.
    B. 14 accesses to disk and 13 access to RAM.
    C. 16 accesses to disk and 15 accesses to RAM.
    D. 16 accesses to disk and 14 accesses to RAM.
    E. 17 accesses to disk and 14 accesses to RAM.

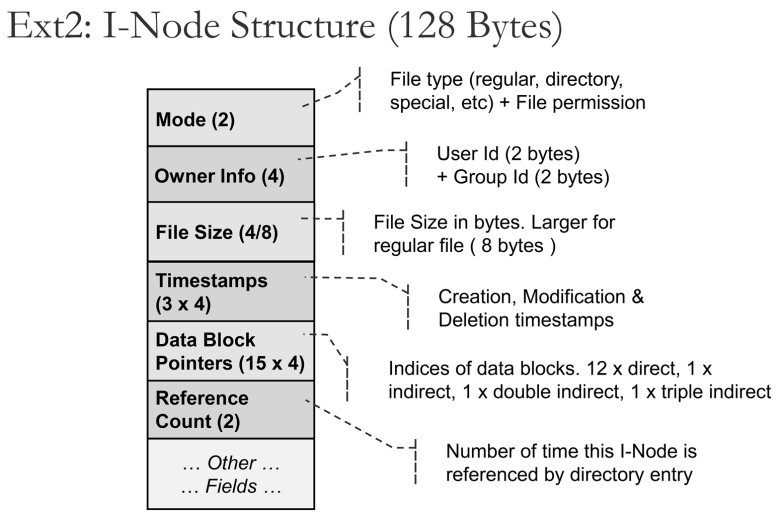
Answer: D: 16 accesses to disk and 14 accesses to RAM.

Explanation: Caching is done ONLY for the inode. Thus, 2 block accesses to disk are needed to access the single indirect blocks.

21. [2 marks] A process opens twice the same file. Assume the process is not opening or closing any files during its execution. The **total number of entries** added by the process to the system-wide open file table is:
    A. 1
    B. 2
    C. 5
    D. Impossible to give a number.

Answer: B: 2

22. [2 marks] According to the lecture, i-node in ext2 file system contains the following information:



Consider system call `ftruncate`, with the following description:

`int ftruncate (int fd, off_t length)`

Description: **`ftruncate`**`()` cause the regular file referenced by `fd` to be truncated to a size of precisely `length` bytes. If the size of the file previously exceeded `length`, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, `ftruncate()` shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to `ftruncate()`.

Consider the following statements:

i. **`ftruncate`** call will modify exactly one i-node field.
ii. **`ftruncate`** call might modify the data blocks pointers in the i-node.
iii. **`ftruncate`** call will modify the entry in the per-process file descriptors table for the process that calls it.
iv. **`ftruncate`** call will modify the entry in the system-wide open files table.

Which statements are TRUE?

A. i. and ii.
B. i., iii. and iv.
C. ii.
D. ii. and iv.
E. None of the above.

Answer: C: ii.

23. [2 marks] In a program, the following call is used:

```
fd = open("/home/e1234567/myfile", O_RDONLY);
```

The call is successful and `myfile` is a hard link. How many I-nodes have been read by this call?

A. 1
B. 3
C. 4
D. 5
E. Impossible to say.

Answer: C: 4

24. [2 marks] Consider the following code:

Line#	Code snippets
1	`int t = 0;`
2	`int thread()`
3	`{`
4	`    int *s;`
5	`    s = &t;`
6	`    (*s)++;`
7	`}`
8	`int main()`
9	`{`
10	`    pthread_t t1, t2;`
11	`    pthread_create(&t1, NULL, thread, NULL);`
12	`    pthread_create(&t2, NULL, thread, NULL);`
13	`    t++;`
14	`    printf("%d", t);`
15	`}`

Assume the code is part of a program that compiles and runs successfully. The code will:

A. Have a race condition and we cannot predict what it will print
B. Always print 3, because the threads share the data section of the memory space.
C. Always print 2, because the threads have independent stack space.
D. Always print 1, because the global variables are not affected by threads' execution.
E. None of the options.

Answer: A: Have a race condition and we cannot predict what it will print

Section 3. Short Questions

Q25-27: Answer questions 25-27 based on the following description:

A process has 3 threads, T1, T2, and T3, and its code does not contain any `exec*()` commands. T1 opens 3 files and T2 creates a pipe. After these actions, T1 executes a `fork()` command and T2 executes a `fork()` whose child immediately calls `execvp()` to execute a single-threaded program. T1 closes the three files and then terminates. T3 terminates without opening any file. Note that all threads are POSIX threads.

25. [2 marks] How many different processes execute? Why?
    Assume no other processes have been created without being mentioned in the problem description.
    a. Answer: ____*3*____ processes
       Explanation: ___*The original process does two forks each of which creates a new process. The other actions do not create/terminate processes*____
       Full credit is given for answer of 2 if the explanation clearly shows that the initial program is not counted.

b.

26. [1 mark] How many different programs are being executed? Why?
    Assume no other programs have been called without being mentioned in the problem description.
    a. Answer: ____*2*____programs
       Full credit is given for answer of 1 if the explanation clearly shows that the initial program is not counted.
27. [3 marks] How many file descriptors have been used (have been opened) by this program? Why?
    Assume no other files have been opened without being mentioned in the problem description. Each process has STDIN, STDOUT, STDERR.
    a. Answer: ____*24*___file descriptors (in the file descriptors table)
    b. Explanation: _____ *The initial process has 3 fds (STDIN, STDOUT, STDERR). T1 creates 3 fds, T2 creates 2 fds. There are 8 fds in the process now. After this, there are 2 fork() calls -> each process will duplicate the fd table with 8 fds: 24 fds are created in total.* _____
       Partial credit of 1 mark given for answers of 8, 16 and 32 if the counting of the file descriptors is correct for one process.

Section 3. Synchronization

Q28-38: Answer the questions 28-38 based on the following problem description and pseudo-code (questions are independent)

Barbershop Problem

A barbershop consists of a waiting room with `n` chairs, and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Customers and barber are modeled using threads.

There are multiple Customer threads that invoke a function named `getHairCut()`. If a customer thread arrives when the shop is full, it can invoke `exit()`.
There is one barber thread serving Customers. This thread invokes `cutHair()`. When the barber invokes `cutHair()` there should be exactly one thread invoking `getHairCut` concurrently.

A program in pseudo-code modeling this problem follows:

Line#	Initialization	
1	`customers = 0`	`#count the customers`
2	`mutex = Semaphore (1)`	`#protect access to counter`
3	`customer = Semaphore (0)`	`#wait for customer`
4	`barber = Semaphore (0)`	`#wait for barber`
5	`customerDone = Semaphore (0)`	`#customer is done`
6	`barberDone = Semaphore (0)`	`#barber is done`

Line#	Customer Pseudo-code	Line#	Barber Pseudo-code
1	`wait(mutex);`	21	
2	`if (customers == n) {`	22	
3	`  signal(mutex);`	23	
4	`  exit();`	24	
5	`}`	25	
6	`customers += 1;`	26	
7	`signal(mutex);`	27	
8		28	`while (TRUE) {`
9	`wait(barber);`	29	`  wait(customer);`
10	`signal(customer);`	30	`  signal(barber);`
11		31	
12	`getHairCut ();`	32	`  cutHair();`
13		33	
14	`signal(customerDone);`	34	`  wait(customerDone);`
15	`wait (barberDone);`	35	`  signal(barberDone);`
16		36	`}`
17	`wait(mutex);`	37	
18	`customers -= 1;`	38	
19	`signal(mutex);`	39	

28. [1 mark] The code works according to the requirements and there is no synchronization problem.
    A. TRUE
    B. FALSE

Answer: FALSE

29. [1 mark] Assume the code is currently synchronizing correctly (either because the given code is correct, or you can make it correct with minor changes). Line 1 can be moved before line 6 without affecting the correctness of the code.
    A. TRUE
    B. FALSE

Answer: FALSE

30. [1 mark] Assume the code is currently synchronizing correctly (either because the given code is correct, or you can make it correct with minor changes). Line 3 can be removed without affecting the correctness of the code.
    A. TRUE
    B. FALSE

Answer: FALSE

31. [1 mark] The code will deadlock.
    A. TRUE
    B. FALSE

Answer: TRUE

32. [1 mark] Swapping lines 9 and 10 will make the code deadlock-free (the rest of the code stays the same).
    A. TRUE
    B. FALSE

Answer: TRUE

33. [1 mark] Swapping lines 29 and 30 will make the code deadlock-free (the rest of the code stays the same).
    A. TRUE
    B. FALSE

Answer: TRUE

34. [1 mark] Swapping lines 14 and 15 will make the code deadlock-free (the rest of the code stays the same).
    A. TRUE
    B. FALSE

Answer: FALSE

35. [1 mark] Swapping lines 34 and 35 will make the code deadlock-free (the rest of the code stays the same).
    A. TRUE
    B. FALSE

Answer: FALSE

36. [1 mark] Assume the code is synchronizing correctly (either because the given code is correct, or you can make it correct with minor changes). Assume that each thread is given equal opportunities to run (i.e. scheduling is fair). The code suffers of starvation.
    A. TRUE
    B. FALSE

Answer: FALSE

37. [1 mark] Assume the code is synchronizing correctly (either because the given code is correct, or you can make it correct with minor changes). The barber is busy waiting.
    A. TRUE
    B. FALSE

Answer: FALSE

38. [1 mark] Assume the code is synchronizing correctly (either because the given code is correct, or you can make it correct with minor changes). Choose the statement that is TRUE:
    A. The customers are served in the order they arrive because scheduling is fair.
    B. The customers are served in the order they arrive because the semaphore is using a FIFO queue.
    C. To serve the customers in the order they arrive, a queue of customers can be used.
    D. To serve the customers in the order they arrive, a queue of semaphores can be used.
    E. One more semaphore can be used to ensure that customers are served in the order they arrive.

Answer: D: To serve the customers in the order they arrive, a queue of semaphores can be used.

Section 4. Essay

39. [4 marks] You are working on implementing a special purpose dynamic allocator. Suppose all allocations and deallocations of objects follow a particular pattern characterized by two properties:
    - Allocation and deallocation happen in a strict LIFO fashion, i.e., any more recently allocated object will be freed before, or simultaneously with, a less recently allocated object.
    - Objects will be freed in groups of one or more objects without any intervening computation. For an example, consider the following memory allocation pattern, where `[compute]` denotes sections of computation, `Ai` denotes allocation of object `i`, and `F{j,k}` denotes deallocation of objects `j` and k:
      `A1, [compute], A2, [compute], A3, [compute], A4, [compute], F{3,4}, [compute], A5, [compute], A6, [compute], F{6,5,2,1}`

    Describe how would you implement an optimized allocator that exploits this application's particular allocation pattern. In your description, touch on the following points:

    - How would you maintain the information about the allocated objects (i.e. what type of data structure)? (2 marks)
    - How should the allocation and deallocation functions be implemented?
      There is no need to use pseudo-code, a simple explanation would do. (2 marks)
    - Any other assumptions you are making about the allocated space.

    Note that the number and size of allocated objects are not known beforehand, only the allocation pattern is.

Answer: A so-called object stack allocator could be used which allocates memory using a chain of chunks. Within the last chunk, allocation is done simply by bumping a pointer; if the allocation request is larger than the amount remaining in the chunk, a new one is allocated chunk. Deallocation is very efficient – simply reset the the 'next allocation' pointer and chunk to the location of the lowest-number object and discard all more recent chunks, if any.

Full marks were awarded if the student identified the use of stack and clearly explained how the allocation and deallocation is performed. For allocating memory, two main approaches were given marks: (i) divide the memory into equal sized chunks and push a number of chunks (or a reference to chunks) to the stack. (ii) allocate unequal memory chunks and push a data structure with a reference and a size (offset) into the stack.

3 marks were awarded for answers that correctly identified the use of stack, but the deallocation and allocation was not fully correct/feasible, but had the correct approach.

2 marks were awarded for identifying the use of stack with minimal information about allocation and deallocation functions.

Furthermore, answers that suggested a linked list implementation (and it was not obvious that the linked list is implementing a stack) were awarded 2 marks if the rest of the answer was correct.

1 mark was awarded for linked list implementation with minimal details on allocation and deallocation. Some remotely correct answers that hinted about a data structure like stack were given 1 mark.

No marks were awarded for alternate data structures such as bitmaps, arrays, hashtables etc.

40. [3 marks] What would be the fastest way to **transfer** a large amount of data between two processes running on the same operating system? Explain your answer.
    Note: The information does not need to be copied during the transfer.

Answer: mmap() is used to map a shared memory region or a shared file to both processes such that both processes are able to access the shared data on RAM without any system calls.

Merely specifying that a file/shared memory can be used is not sufficient to obtain full marks.

Full marks were awarded for correct usage of mmap() with either a shared memory region or file.
2 marks were awarded for suggesting to use mmap() but no concrete explanation on how exactly is used.
1 mark was awarded for proposing to use shared memory or files, but no indication of using mmap().
No marks were awarded for alternate answers that included

- using pipes
- Pass a pointer
- Share pages
- Pass a file descriptor
- Using registers
- IPC

41. [2 marks] In general `exec()` syscall takes longer time to execute than `fork()` syscall. Briefly explain why.

Answer: fork() duplicates the current process' image and creates a chile process whereas exec() replaces the current process' image with a new executable. Duplicating the current process image requires only memory access whereas bringing in a new executable requires access to secondary storage, which is much slower than memory access. Therefore, exec() takes much longer time than fork() to complete.

Full marks were awarded for explaining how both fork() and exec() work and highlighting the difference of performance between memory access and disk access

1 mark was awarded for answers that explained one of the mentioned syscalls highlighting the overheads involved and for answers that compared exec() and fork() without highlighting the need to access RAM and disk. Answers that mentioned optimizations such as copy-on-write as reasoning for fork to be efficient along with correct explanation about exec() were also given 1 mark.

Answers that were not awarded any marks included:
- Due to context switches
- Page fault/thrashing
- Need to access FAT/inode table
- Merely mentioning fork() can use COW without any explanation on exec()
- TLB flushing

42. [6 marks] A programmer observes that many file blocks pertaining to different files have identical data content in an EXT2 file system. The programmer wants to implement a mechanism that would reduce the space occupied by such blocks on the disk. You are tasked to help this programmer to define a new system call `fork_file()` that replicates the file without allocating new blocks on disk.

    a. Define the parameters, return value and give a description for `fork_file()`. The description must explain how to use the new system call and what it returns. (2 marks)
    b. What type of mechanism would you implement to allow for the files to be written with different information? (i.e. not all blocks will be different after write). Avoid duplication of blocks that are still the same. (2 marks)
    c. Would the current I-node structure and EXT2 filesystem allow you to implement your write mechanism? If not, explain the changes that are needed in EXT2 to support the solution proposed at point b. (2 marks)

Answer:

a) int fork_file(const char *file_path, const char* new_file_path)

Description: fork_file() takes in a file path and creates a new directory entry in the path of the new_file_path duplicates the inode of file_path keeping the same data block pointers, but new timestamps (creation/modification set to current time) and reference count (set to 1). fork_file returns 0 on success, or -1 if unsuccessful.

This is very similar to a hard link in a unix system, but the inode is duplicated (not same).

When the "fork" (new_file_path) is modified or truncated, the source file will not change, and new disk blocks will be allocated when necessary.

Example usage:

fork_file("/home/u/hello.txt", "/home/hello2.txt") will create a new file, "/home/hello2.txt", that has a separate inode in the inode table, but points to the same data blocks as /home/u/hello.txt

b) Essentially, we need to implement a copy-on-write mechanism. When one of the files is written (original or the fork), the block needs to be duplicated and the pointers within the inode need to be changed accordingly.

c) The inode allows for such implementation (no changes are needed for inode), but the EXT2 management does not allow for it. Basically, we need to know which disk blocks are shared among multiple files. This info needs to be part of the group (partition) information in EXT2

There are a few alternatives to achieve this (all accepted as correct answers):

- a reference count for each disk block. Whenever a file is being forked, the disk blocks in that file should have their reference counts incremented.
- a 'shared' bit (flag) associated with each data block. If the data block is flagged as 'shared', then don't write to it directly, but create a copy of it and modify that copy, then modify the data block pointer to pointer to our new data block. This solution needs to be combined with additional checks to handle the case when a block is shared among multiple files.
- each data block needs to have an associated permission bit. Initially for a writable file, the permission bit for all blocks is 1. However, after a fork_file, all data blocks of the forked file and new file have permission bits set to 0. If a write is called on a writable file, at a file offset that points to a data block that has permission bit 0 that block will be copied to a new block and the copy will have permission bit set to 1.

Implementing and using the reference count for block (not needed to obtain full score):

- A super naive way to do this is to just store reference counts in a flat table, but this immediately incurs a huge overhead.
- A slightly less naive way to do this is to create a giant B-tree that maps blocks to reference counts, which incurs overhead proportional to disk usage.
- And an even slightly less naive way is to only store blocks with reference count at least 2 in this tree, and then if a block is allocated but not in this tree, it is considered to have reference count 1, and this incurs overhead proportional to the blocks being duplicated.
- This table/tree should be stored next to the block bitmap in each block group, and it should be responsible only for the disk blocks in that block group.
- Then whenever a file is being deleted, it needs to check for each disk block in the file, if they exist in this B-tree, and to decrement / remove the entry from the B-tree. If the I-node now has reference count 0, it should set the bit in the free disk block bitmap appropriately. Otherwise, it should remain allocated.

- Whenever a file is being forked, the disk blocks in that file should have their reference counts incremented.
- Whenever a file is being modified, we need to first check if the corresponding disk block at the offset we are writing to has reference count > 1. If so, the block should be copied, the I-node of the file modified to point to the newly copied block, the free block bitmap updated, the reference count table/tree updated (if using the naive methods) to map the new block to reference count 1, then the write should be applied to this new block, and finally the reference count of the original block should be decremented.

Grading notes:

-1 mark is applied if:

- a. return file descriptors or use file descriptors instead of path. Fork_file should not open the new file (just duplicate the file)
- a. Not clear how the new_file name is passed to the function.
- c. the inode is modified instead of the EXT2 management
- b. Copy-on-write is mentioned, but it is not clear how it is achieved. Basically, i-node duplication for CoW is not explained.
- a./b. It is not clear how the fork happens: new i-node with links to the same blocks.

-2 marks are applied if:

- a. no function signature and description is provided
- b./c. Duplication does not focus on the blocks, but on other issues (such as open file table, or adding blocks to a file, or opening the file)
- c. The new (inode) structure does not make sense.
- c. Student says no changes are needed.
- b. Use a symlink instead of directly linking the same blocks in a new inode (and trying to link the symlink inode with original file)
- c. Totally changing the way inode is storing the blocks into a linked list, without mentioning the reference count issue.

If one good point is mentioned, but no other explanations (or explanations do not make sense) 1 mark is allocated.

Concepts tested by this question:

1. Understanding fork for processes and transferring the concept to files
2. Understanding COW for memory and applying it to disk blocks
3. Understanding file systems and how to modify them to achieve a new requirement

—End of paper—