CS 502 - Operating Systems
2021S Final Exam
112 Points

Instruction

There are **Tian:** ***5*** pages, with a total of **Tian:** ***13*** questions. It is important to read all questions carefully, **as we might make different assumptions than what you have read in the book**.

Submission: Please use the provided template for answering each question. It is highly recommended that you typeset the answers using a text editor (rather than hand-written and scan). You can submit either the worksheet either in `.txt` or `.pdf` via Canvas.

Question 1: Concurrency

(8 Points) A concurrent program (with multiple threads) looks like this:

```c
volatile int counter = 1000;
void *worker(void *arg) {
  counter--;
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t p1, p2;
  pthread_create(&p1, NULL, worker, NULL);
  pthread_create(&p2, NULL, worker, NULL);
  pthread_join(p1, NULL);
  pthread_join(p2, NULL);
  printf("%d\n", counter);
  return 0;
}
```

Assuming `pthread_create()` and `pthread_join()` all work as expected (i.e., they don't return an error), what are the possible outputs? Briefly explain.

Grading Notes: Ouput: 2 points; the explaination: 2 points.

In this code sequence, there is a race on the decrement of counter.

(4 Points) If the threads perform the decrement one after the other, the "right" result will occur: 998.

(4 Points) However, if the decrements take place concurrently, each will read 1000 as the value of the counter, decrement it to 999, and then write 999 back to the counter, thus losing one decrement.

All other outputs should not be possible.

Question 2: Ticket Lock

(10 Points) Here is a ticket lock. Assuming a maximum of 5 threads in the system, and further assuming the ticket lock is used "properly" (i.e., threads acquire and release it as expected), what values of lock→ticket and lock→turn are possible (at the same time)? Briefly explain.

```c
typedef struct __lock_t {
  int ticket, turn;
} lock_t;

void lock_init(lock_t *lock) {
  lock->ticket = 0;
  lock->turn = 0; }

void lock(lock_t *lock) {
  int myturn = FetchAndAdd(&lock->ticket);
  while (lock->turn != myturn)
    ; // spin
}

void unlock(lock_t *lock) {
  lock->turn = lock->turn + 1;
}
```

Grading Notes: Possible/Not Possible: 1 point; Explain: 1 point

(a) ticket=0 and turn=0

Possible. Before any thread does anything.

(b) ticket=0 and turn=1

Not Possible. ticket is incremented before turn, thus this should not happen.

(c) ticket=1 and turn=0

Possible. After one thread calls lock(), ticket is now 1, and turn is 0.

(d) ticket=16 and turn=5

Not Possible. With only 5 threads, ticket can't get this far ahead of turn.

(e) ticket=1000 and turn=999

Possible. As before, ticket is at 999, then one thread calls lock(), resuting in this state.

Question 3: Producer/Consumer Problem

(6 Points) Here is the classic semaphore version of the producer/consumer problem:

```
void *producer(void *arg) { // core of producer
  for (i = 0; i < num; i++) {
    sem_wait(&empty);
    sem_wait(&mutex);
    put(i);
    sem_post(&mutex);
    sem_post(&full);
  }
}

void *consumer(void *arg) {  // core of consumer
  while (!done) {
    sem_wait(&full);
    sem_wait(&mutex);
    int tmp = get(i);
    sem_post(&mutex);
    sem_post(&empty);
    // do something with tmp ...
  }
}
```

For the following statements about this working solution, which statements are true, and which are not? To answer, first indicate true or false, then briefly explain.

Grading Notes: This just reviews how the producer/consumer solution works, when semaphores are used. True/false: 1 point; explaination: 1 point

(a) The semaphore `full` must be initialized to 0.

True. `full` tracks how many full buffers there are; at the beginning, zero

(b) The semaphore empty must be initialized to 1.

False. empty tracks how many empty buffers there; if there are more than one, it should be initialized to the number of empty buffers. Thus, false.

(c) The semaphore mutex must be initialized to 1

True. mutex provides mutual exclusion, and thus must be set to 1 at the beginning to work properly.

Question 4: Deadlocks

(8 Points) One way to avoid deadlock is to schedule threads carefully. Assume the following characteristics of threads T1, T2, and T3:

- T1 (at some point) acquires and releases locks L1, L2
- T2 (at some point) acquires and releases locks L1, L3
- T3 (at some point) acquires and releases locks L3, L1, and L4

For which schedules below is deadlock possible? To answer, first indicate possible or not possible, then briefly explain.

Grading Notes: Possible/Not Possible: 1 point. Explain 1 point.

(a) T1 runs to completion, then T2 to completion, then T3 runs

Not Possible. T2 and T3 do not run at the same time.

(b) T1 and T2 run concurrently to completion, then T3 runs

Not Possible. T2 and T3 do not run at the same time.

(c) T1, T2, and T3 run concurrently

Possible. T2 and T3 run at the same time.

(d) T1 runs to completion, then T2 and T3 run concurrently

Possible. T2 and T3 run at the same time.

Question 5: Increment and Reset

(8 Points) Here is source code for another program, called increment.c:

```c
int value = 0;
int main(int argc, char *argv[]) {
  while (1) {
    printf("%d", value);
    value++;
  }
  return 0;
}
```

While increment.c is running, another program, reset.c, is run once as a separate process. Here is the source code of reset.c:

```c
int value;
int main(int argc, char *argv[]) {
  value = 0;
  return 0;
}
```

Which of the following are possible outputs of the increment process? To answer, first indicate possible or not possible, then briefly explain.

Grading Notes: Possible/Not Possible: 1 point. Explain 1 point.

(a) 012345678 ...

possible.

(b) 012301234 ...

Not possible. value is reset, but how?

(c) 01234567891011 ...

possible. value not reset; it's just "9 10 11" squished together

(d) 123456789 ...

Not Possible (increment starts at 0)

Question 6: File System API

(10 Points) In this question, we explore the file system API. Briefly answer each question.

Grading Notes: 2 points each.

(a) What are some differences between a file descriptor and a inode number?

A file descriptor is per-process integer used to access files an inode number is a low-level number, unique for each file/directory

(b) What are some differences between `fstat()` and `stat()`?

Both are for obtaining file information. `fstat()` is identical to `stat()`, except that the file about which information is to be retrieved is specified by the file descriptor fd rather than pathname.

(c) What are some differnces between `lseek()` and the disk seek operation?

`lseek()` is a file system API that changes the current file offset while the disk seek operation is about repositioning the disk head.

(d) Let us say we wish to write data to a file and then force the contents of the file to disk immediately. What sys calls should we call?

first `write()` then `fsync()`.

(e) Describe how you could use the file system APIs to implement the command line utility `ls` with the flag `-l`.

First using `readdir()` to get the `dirent` data structure for each file, then using `stat()` on each file to obtain more information.

Question 7: Disks

(9 Points) In this question, we will perform some simple calculation on a simplified disk.

Grading Notes: answer: 1 point; explaination: 2 points

(a) Assume a simple disk that has only a single track, and a simple FIFO scheduling policy. The rotational delay on this disk is R, there is no seek cost (only one track!), and transfer time is so fast we just consider it to be free. What is the approximate worse case executation for three requests (to different blocks)? Briefly explain.

The answer is: 3R.

If consecutive blocks are far apart that requires the entire rotation. For example assuming the single track has 1 to 10 blocks, and the disk head positions at block 10; then the worse case arises with the following access pattern: the first request is to block 1, the second request is to block 10, and the third requst is to block 9.

(b) Now assume that a shortest-access-time-first scheduler is being used by the disk (but it still only has a single track). What is the approximate worse case executation for three requests (to different blocks) now? Briefly explain.

The answer is: R.

Let's first re-examine the workload above, block 1, 10, and 9. If we re-order the request to be block 10, 1, 9, then we can do all three requests in R.

(c) Now assume the disk has three tracks. The time to seek between two adjacent tracks is S; it takes twice that to seek across two tracks (e.g., from the outer to the inner track). Given a FIFO scheduler, what is the worse-case time for three requests? Briefly explain.

The answer is: 6S + 3R = 3x(2S + R).

Similar to (a), if the disk head is currently positioning in inner track, but the next request is to the outer track, it will take 2S to seek. If unlucky, it will further incur a full rotation delay.

Question 8: RAIDs

(12 Points) For this question, we will examine how long it takes to perform a small workload consisting of 12 writes to random locations within a RAID. Assume that these random writes are spread "evenly" across the disks of the RAID. To begin with, assume a simple disk model where each read or write takes D time units.

Grading Notes: answer: 1 point; explaination: 2 points

(a) Assume we have a 4-disk RAID-0 (striping). How long does it take to complete the 12 writes? Show your work.

The asnwer is 3D.

This is because for each write, the latency is D. And 12 writes can happen in parallel in 4 disks. Therefore, it takes 12D/4 =3D.

(b) How long on a 4-disk RAID-1 (specifically RAID-10)? Show your work.

The asnwer is 6D.

This is because for each logical write, we will need to issue 2 physical writes. Therefore, the total number of physical writes we need to perform is 24. These 24 writes can happen in parallel in 4 disks. Therefore, it takes 24D/4 =6D.

(c) How long on a 4-disk RAID-4 (parity)? Show your work.

The asnwer is 24D.

First, we need to know that the parity disk is the bottleneck, i.e., it forces all writes to be serialized.

Second, for each logical write, using the full-strip write, we will need to perform 2 phyiscal reads and 2 physical writes. Both reads and writes can be completed in parallel, but do not interwined.

Therefore, for each logical write, it takes 2D. And we need to perform 12 of those at the parity disk.

(d) How long on a 4-disk RAID-5 (rotated parity)? Show your work.

The asnwer is 6D.

Unlike RAID-4, the parity disk bottleneck problem does not exist any more. However, we still need to perform 4 logical reads/writes. Both reads and writes can be completed in parallel, but do not interwined. Therefore, for each logical write, it still takes 2D.

However, all 12 logcial writes can happen in parallel, therefore we have 12/4 x 2D = 6D.

Question 9: A Basic File System

(9 Points) For this question, assume a simple disk model where each disk read of a block takes D time units. Also assume the basic layout is quite like the very simple file system. Further, assume that inodes can be cached once they are read in from the disk proper and will not be written back to disk once the corresponding operations complete.

Grading Notes: answer: 1 point; explaination: 2 points

(a) Assume that all data and metadata begin on disk. Assume further that all inodes are in separate blocks, and that each directory is only one block in size. How long does it take to **open** the file `/a/b/c/d.txt`? Explain your work.

9D.

read root inode, root data block, read a, b, and c, for inode and data block; and finally read the inode for d.txt

(b) Assume after opening the file, we read the file in its entirety. It is a big file, containing 1036 blocks. The inode itself has room for 12 direct pointers and 1 indirect pointer. Disk addresses are 4 bytes long, and disk blocks are 4KB in size. **After opening it**, how long does it take to read the entire file?

1037D = 1036+1 .

1036 - 12 = 1024; meaning that we need to read in a data block for the 1024 indirect pointers.

(c) Now assume a new inode structure is introduced, in which there is only one pointer: a double indirect pointer, which points to the double indirect block, which can point to 1024 indirect blocks, each of which can point to 1024 blocks. After opening the file, how long does it take to perform 50 random reads within a very large file?

101D = 1 + 2x50.

Since it is random, so it is probably going to be in 50 different bocks. In order to get to each block, we will need to first read the double indirect block (D), then for each read, we need to first get read the indirect block and then the data block (2D).

Question 10: Journaling File Systems

(12 Points) For this question about journaling file systems, assume the following disk model: it takes S time units to seek to any location on the disk; the full rotational delay is R time units; transffering data takes T units, regardless of the amount of data transferred.

Grading Notes: answer: 1 point; explaination: 3 points

(a) If we don't use journaling at all. How long does it take (on average) to append a block to an existiing file, assuming that all relevant metadata and data was in the OS page cache to begin with (i.e., no reads from disk need to occur)? Show your work.

$(S + \frac{R}{2} + T) * 3$

We need to write an inode, the user data to the data block, and a data bitmap to the disk. So we will be performing 3 disk writes, most likely to different disk locations.

For each disk write, we will first seek, then rotate, and finally transfer. The average time will be $S + \frac{R}{2} + T$.

(b) Now assume we use the data journaling in which all data and metadata are logged before being written to disk. How long does the append take in this case? Show your work.

The journal write and commit = $S + \frac{R}{2} + T + R + T$,

The checkpoint = $(S + \frac{R}{2} + T) * 3$

Total = The journal write + The checkpoint

To perform the journaling write, we will need to perform two writes. The first write is for TxB, inode, data bitmap, and the data block. The second write is for TxE, to a disk location immediately after the first write. For the first write, we will need to first seek (S), then rotate ($\frac{R}{2}$), and finally transfer (T). However, most likely, after the first write completes, the disk will just rotate pass the disk location for TxE. Therefore, it will now take a full rotational delay (R) to start writing TxE.

The checkpoint phase is identical to (a).

(c) Now assume we have journaling, but only for metadata; data blocks are written directly to their final location. How long does the append take in this case? Show your work.

max(S + $\frac{R}{2}$ + T, S + $\frac{R}{2}$ + T) + R + T + (S + $\frac{R}{2}$ + T) * 2

We will write data block first, taking S + $\frac{R}{2}$ + T. Concurrently, we will write the journal information (inode and the data bitmap) to the transaction. The journal write takes S + $\frac{R}{2}$ + T, same as (b).

After the these two writes complete, we will start the journal commit (i.e., TxE) which takes R+T.

The last step is to checkpoint the metadata (inode and data bitmap) to two disk locations, which takes (S + $\frac{R}{2}$ + T) * 2

Question 11: Virtual Memory

(9 Points) You are given a system with 64 bytes of physical memory, 4 byte pages, and 16-byte virtual address spaces. Further, you are told that the page table structure uses the high bit to indicate Valid/NOT, and the rest bits for PFN. However, one of the page table entries is missing.

[0]	0x00000000
[1]	0x800000?? (missing!)
[2]	0x00000000
[3]	0x00000000

Grading Notes: answer is 1 points; explaination is 2 points.

(a) If you know that a virtual address 0x7, using the above page table, was translated to 0x33. What two hex digits are missing from page table entry 1 above? Show your work.

The answer is: 0x0C or 0C

total 64 bytes, 4 byte pages, then we will have 16 phyiscal pages. to index 16 phyiscal pages, we need PFN to be 4 bits, similarly, to index 4 virtual pages, we need VPN to be 2 bits 0x7 = 0111 is the 3 bytes (11) of the 1 page (01) 0x33 = 00110011; so the question becomes, what does "001100" in hex?

(b) After sometime, the page table now becomes the following. What will the virtual address 4 translate to phyiscal address in decimal? Show your work.

[0]	0x00000000
[1]	0x80000005
[2]	0x8000000a
[3]	0x00000000

The answer is: 20.

First, virtual address 4 can be represented as 0100 where VPN=01, and offset is 00. Second, from the page table, we know the PFN is 000101. So the physical address is 00010100. In decimal it is 20.

(c) Again, the page table changes to the following. What ranges of virtual addresses are valid? Show your work.

[0]	0x00000000
[1]	0x8000000e
[2]	0x00000000
[3]	0x80000009

The answer is: [4, 7] and [12, 15]

Since only the VPN=1 and VPN=3 are valid; we know that all valid virtual addresses have to start with either 01 or 11.

For each virtual page, the offset is between 00 and 11. Therefore the valid address ranges are 0100 to 0111 (4 to 7) and 1100 to 1111(12 to 15).

Question 12: Multi-level Page Tables

(4 Points) The multi-level page table is something that cannot be avoided. No matter what you do, there it is, bringing joy and horror to us all. In this question, you'll get your chance at a question about this foreboding structure. Fortunately, you don't have to perform a translation[1]. Instead, just answer these true/false questions about the multi-level page table. Briefly explain your choice.

Grading Notes: True/false: 0.5 point; explaination: 0.5 point

(a) A multi-level page table may use more pages than a linear page table

True. If a byte is valid on each page of the address space, the multi-level structure adds the overhead of multiple levels atop the linear structure.

(b) It's easier to allocate pages of the page table in a multi-level table (as compared to a linear page table)

True The linear table must be allocated contiguously; the multi-level page table is chopped into chunks. Thus, it is easier to allocate the multi-level page table.

(c) Multi-level page table lookups take longer than linear page table lookups.

True. Generally, true, because of the multiple levels.

(d) TLBs are useful in making multi-level page tables even smaller.

False. TLBs make translation faster, not page tables smaller.

Question 13: Putting it All Together

(7 Points) In this course, we studied four pillars for Operating Systems. Please take a moment to reflect and write down three of your favorite techniques that you have learnt. For each technique, briefly describe the problem it solves, how it works, and why you like it.

Grading Notes: Each technique is worth 3 points + 1 free point!

Any techniques that were relevant to OS.

[1]You must be glad you didn't have to do a multi-level translation again, no?