

Final Review

Tu May 2, 6:50-8:40pm

**Willard 060, 062, 158, 262, 358,
360**

Final Exam Structure

- Format:
 - 24 pts – fill-in-blanks (8 questions)
 - 30 pts – short answer (6 questions)
 - 46 pts – constructions (6 questions)
 - Multipart
- Covers
 - Memory management (hex)
 - Threads
 - Concurrency* (~25 points)
 - Deadlock
 - File systems* (~30 points)
 - IO

Threads, Concurrency, Deadlock

- Threads
 - Shared memory regions
 - Threading Models and Signals
- Concurrency
 - Terms (Critical Sections)
 - Spinlocks
 - Condition Variables
 - Semaphores
- Combination
 - Properties of threads impact concurrency
- Deadlocks
 - Terms
 - Avoidance and Detection

Concurrency

- Quiz question

```
1: load deposit r1      // load deposit amount (from memory) into r1
2: push r1              // push r1 onto top of the stack
3: load balance r2      // load balance (from memory) into r2
4: pop r3                // pop the top of the stack to r3
5: add r3 r2 r4          // add r3 + r2, assign to r4
6: store r4 balance     // store r4 into balance
```

Operation on stack (bracketed next to lines 2-4)

shared (written next to line 4)

- Deposit is said to be thread-specific
 - What memory is thread-specific?
- Balance is shared
 - What memory is shared?

balance

Concurrency

- Semaphores
 - Variable with an initial value
 - What value do you set?
 - How to use?
- Spinlocks
 - Using software techniques – need some context
 - Using atomic instructions like test&set

Bounded Buffer using Semaphores

```
int BB[N];
int count, head, tail = 0;
Semaphore_t m; // value initialized to 1
Semaphore_t notfull; // value initialized to N
Semaphore_t notempty; // value initialized to 0
```

```
Append(int elem) {
    P(notfull);
    P(m);

    BB[tail] = elem;
    tail = (tail + 1)%N;
    count = count + 1;

    V(m);
    V(notempty);
}
```

```
int Remove () {
    P(notempty);
    P(m);

    int temp = BB[head];
    head = (head + 1)%N;
    count = count - 1;

    V(m);
    V(notfull);
    return(temp);
}
```

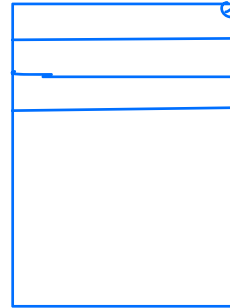
Q 1

- File currently consists of 100 blocks.
- FCB is already in memory => any changes made to the FCB *does not* go to the disk and hence, doesn't count as an I/O operation.
- Index block (in case of indexed allocation) is in memory => any change made to it won't count as an I/O operation.
- In the contiguous-allocation scheme, there is **no room** to grow in the beginning.
- Block information to be added is stored in memory.

FAT
Add/delete
first pointer



UNIX
add/delete
first



Q 1 (a)

- Block is added at the beginning
1. Contiguous: shifting of blocks required.
100 READ and 100 WRITE (for shifting)
+ 1 WRITE to create the new block
 2. Linked: 1 WRITE to create the new block
 3. Indexed: 1 WRITE to create the new block

Q 1 (b)

- Block is added at the middle
1. Contiguous: shifting of blocks required.
50 READ (for block # 51 – 100) and 50 WRITE (for block # 51 - 100)
+ 1 WRITE to create the new block
 2. Linked: 50 READ (linked-list traversal)
and 1 WRITE (pointer update to new block) + 1 WRITE to create the new block
 3. Indexed: 1 WRITE to create the new block

Q 1(c)

- Block is added at the end
1. Contiguous: no shifting of blocks required.
1 WRITE to create the new block
 2. Linked: 100 READ (linked-list traversal) and 1 WRITE (pointer update to new block) + 1 WRITE to create the new block
 3. Indexed: 1 WRITE to create the new block

Q 1 (d)

- Block is removed from beginning.
Assume that removal operation is not included in I/O operation.
1. Contiguous/Indexed: no shifting of blocks required.
0 READ/WRITE operations
 2. Linked: 1 Read to the first block to find the next block pointer (and update FCB)

Q 1 (e)

- Block is removed from the middle (say, 50th block)
 1. Contiguous: shifting of blocks required.
50 READ and 50 WRITE (for shifting)
 2. Linked: 50 READ (linked-list traversal)
and 1 WRITE (pointer update to 49th block)
 3. Indexed: 0 READ/WRITE operation

Q 1 (f)

- Block is removed from the end
-
1. Contiguous: no shifting required.
0 READ/WRITE operation.
 2. Linked: 99 READ (linked-list traversal)
and 1 WRITE (pointer update to 99th
block)
 3. Indexed: 0 READ/WRITE operation

Q 2 (a)

- Consider a system where free space is kept in a free-space list.
- a) Suppose that the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain.

Q 2 (a)

- Blocks (fsck):
 - for every block keep 2 counters:
 - a) # occurrences in files
 - b) # occurrences in free list.
 - For every inode, increment all the (a)s for the blocks that the file covers.
 - For the free list, increment (b) for all blocks in the free list.
 - Ideally $(a) + (b) = 1$ for every block.
 - However,
 - If $(a) = (b) = 0$,
missing block, add to free list.

Q 2 (b)

- b) Consider a file system similar to the one used by UNIX with indexed allocation. How many disk I/O operations might be required to read the contents of a small local file at /a/b/c? Assume none of the disk blocks are initially cached.

Q 2 (b)

inode → data



- Get “/” i-node from disk (usually fixed, e.g., #2)
- Get block after block of “/” using its i-node till entry for “a” is found (gives its i-node #).
- Get i-node of “a” from disk
- Get block after block of “a” till entry for “b” is found (gives its i-node #)
- Get i-node of “b” from disk
- Get block after block of “b” till entry for “c” is found (gives its i-node #)
- Get i-node of “c” from disk
- Find out whether block you are searching for is in 1st 10 ptrs, or 1-level or 2-level or 3-level indirect.
- Based on this you can either directly get the block or retrieve it after going through the levels of indirection.

Q 2 (c)

- c) Suggest a scheme to ensure that the pointer is never lost as a result of memory failure.


journal

- Before removing a block from the free list, log an event to disk/stable storage with current next pointer of free list, and then do
operation.

Q 3

- Block size: 8KB
- inode details
 - 12 direct block pointers
 - 1 each of single, double and triple indirect pointers
 - Pointer size 4 bytes $\rightarrow 2k$

- What is the maximum file size?

$$(12 + 2k + 2k \cdot 2k + 2k \cdot 2k \cdot 2k) \cdot 8KB$$


Q 3

- Capacity in 12 direct pointers: $8\text{KB} * 12 = 96\text{KB}$
- 1 indirect block can hold $8\text{KB}/4\text{B} = 2\text{K}$ pointers
- Single Indirect = $2\text{K} * 8\text{KB}$
- Double Indirect = $2\text{K} * 2\text{K} * 8\text{KB}$
- Triple Indirect = $2\text{K} * 2\text{K} * 2\text{K} * 8\text{KB}$
- Max Size = $96\text{KB} + 16\text{MB} + 32\text{GB} + 64\text{TB}$

Q 1

Cylinders = 0...4999; Head is at 2150 and previous request was at 1805

Queued up requests = 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681

(a) FCFS order = 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681. Seek distances =

$81 + 857 + 1084 + 504 + 2256 + 1074 + 1262 + 1167 + 3442 + 1284$

Total seek distance is 13011 cylinders

(b) SSTF order = 2069, 2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356.

Seek distances = $81 + 227 + 504 + 881 + 1284 + 3347 + 95 + 311 + 668 + 188$

The total seek distance is 7586 cylinders

(c) SCAN order = 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356.

Seek distances = $146 + 504 + 881 + 1284 + 34$
 $+ 2930 + 451 + 95 + 311 + 668 + 188$

The total seek distance is 7492 cylinders.

Q 1

(d) LOOK order = 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356.

Seek distances =

$146+504+881+1284+2896+451+95+311+668+188$

The total seek distance is 7424 cylinders.

(e) C-SCAN order = 2296, 2800, 3681, 4965, 356, 544, 1212, 1523, 1618, 2069.

Seek distances = $146+504+881+1284+34+4999$

$+356+188+668+311+95+451$

The total seek distance is 9917 cylinders.

(f) C-LOOK order = 2296, 2800, 3681, 4965, 356, 544, 1212, 1523, 1618, 2069.

Seek distances =

$146+504+881+1284+4609+188+668+311+95+451$

The total seek distance is 9137 cylinders.

Q 2

- What complications can occur when DMA is given virtual address and the translation is done during DMA?
- This would require DMAs to understand and translate virtual addresses to physical addresses, i.e., it needs access to the Page Table. Understanding Paging mechanism of the host processor (which may itself involve multi-level page tables) for the peripheral I/O DMA is asking for too much sophistication.