EECE.4810/EECE.5730: Operating Systems

Spring 2018

Exam 3 Solution

1. (18 points) ***Segmentation***

Write the function below, which could be used in a program simulating segmentation as a memory management method. The function uses this structure for each segment table entry:

```
typedef struct {
   unsigned char valid;        // Valid bit—either 0 or 1
   unsigned int base;          // Segment base address
   unsigned int bound;         // Segment upper bound (or size)
} segTableEntry;
```

The three function arguments represent the segment table (the array `ST`), the segment number (`segNum`), and the offset into the segment (`segOff`). The function should use the segment number to index into the segment table and return the physical address. If an error would occur in the address translation, the function should return 0. You may assume the segment number is valid and that it does not go outside the array boundaries.

The comments in the function outline how it should behave.

Students had to complete the bold, underlined code.

```
unsigned int translate(segTableEntry ST[], unsigned int segNum,
                       unsigned int segOff) {
   // Variable declarations (none needed)

   // Access correct segment table entry & check error conditions
   // If no errors, calculate & return physical address
   if (ST[segNum].valid && segOff < ST[segNum].bound)
      return ST[segNum].base + segOff;

   // If an error would occur in the translation, return 0
   else
      return 0;
}
```

2. *(36 + 7 points)* ***Paging***

a. *(18 points) Explain what conditions would create the slowest possible translation for each of the page table organizations we discussed. (For example, if a tree-based page table existed, the slowest possible translation would happen if the tree was imbalanced such that every node had only one child, and all nodes had to be searched to find the right page table entry.)*

You should not give a numeric answer, but may provide an example to explain your answer.

i. *Two-level page table*

Solution: The access time for any entry in a two-level page table is the same—one access to the first-level table to find the location of the correct second-level table, then an access to that second-level table to perform the translation.

ii. *Hashed page table using chaining*

Solution: The worst case would be that every page number hashed to the same entry in the hash table, meaning the page table essentially behaves like a linked list. If the desired page table entry is the last one in the chain for that entry, that would require the worst-case translation time.

iii. *Inverted page table*

Solution: Since inverted page tables must be searched to find an entry with a matching page number, the worst case would be that the desired PTE is the last one searched.

2 (continued)

b. *(18 points) The clock algorithm for page replacement uses a circular linked list (a linked list in which the last node points to the head of the list) to track currently used pages and to find a replacement candidate. Assume the following structure represents each node in the list:*

```
typedef struct n {
   unsigned char ref;          // Reference bit—either 1 or 0
   unsigned int pageNo;        // Page number
   struct n *next;             // Pointer to next node
} clockNode;
```

Complete the function below, which takes as an argument the current "clock hand" (a pointer to the first node to consider for replacement) and returns the address of the node representing the page to be replaced. The comments in the function outline how it should behave.

Students were responsible for bold, underlined code.

```
clockNode *findRepl(clockNode *clockhand) {

   // Variable declarations
   clockNode *n = clockhand;

   // Start at current "clock hand" and loop until node found
   //   with ref == 0
   while (n->ref) {

      // Clear reference bit for current node, then move to next
      //   node in list
      n->ref = 0;
      n = n->next;
   }

   // When loop ends, should have pointer to node with reference
   //   bit == 0, so return that pointer
   return n;
}
```

c. *(7 points, **EECE.5730 only**) In paged memory management, is the operating system involved in every address translation? If so, explain how; if not, explain what the role of the operating system is in a system using paged memory management.*

Solution: The role of the operating system is to manage the page table, not perform the actual translation. The OS is therefore only invoked on page faults to potentially choose a candidate for eviction, request the desired block from disk, and update the page table accordingly.

3. *(31 **+ 8** points)* ***File systems***

a. *(17 points) Given a system using FFS as the file system, if the block size is 2 KB and the address size is 64 bits, what is the maximum possible file size?*

Remember, each FFS inode contains 12 direct pointers to disk blocks, as well as three other pointers—an indirect pointer, a doubly indirect pointer, and a triply indirect pointer.

*You do **NOT** need to give an exact numeric result—you can express your result as a sum of powers of 2 (for example, 2^{16} bytes) or numbers of KB, MB, GB, or TB (for example, 64 KB), since the exact value would be difficult to find without a calculator.*

However, you must show all of your work to earn full credit.

Solution: The maximum file size requires using all indirect pointers to point to data blocks. Note that each indirect block contains (2 KB/block) / (8 bytes/pointer) = 2^{11} / 2^3 = 2^8 pointers

- The 12 direct pointers point to 12 disk blocks → (12 blocks) * (2^{11} bytes/block) = 24 KB
- The indirect block points to 2^8 blocks → (2^8 blocks) * (2^{11} bytes/block) = 2^{19} bytes
- The doubly-indirect block points to 2^8 indirect blocks, each of which points to 2^8 blocks → (2^8 * 2^8 blocks) * (2^{11} bytes/block) = 2^{27} bytes
- The triply-indirect block points to 2^8 doubly-indirect blocks, which behave as above → (2^8 * 2^8 * 2^8 blocks) * (2^{11} bytes/block) = 2^{35} bytes

Therefore, the maximum file size in this system would be:

$2^{35} + 2^{27} + 2^{19} + 12*2^{11}$ bytes = 32 GB + 128 MB + 512 KB + 24 KB = 34,494,504,960 bytes

b. *(6 points) Why is the FAT file system typically only used on smaller disks?*

Solution: The FAT system uses a linked list of table entries to store the structure of a file. Since linked lists must be searched in order, this file system does not scale well to larger volumes.

c. *(8 points) In a log-based file system, explain (i) what gets written to the log, and (ii) how logging allows a file system to tolerate crashes and ensure changes to the file system are completed properly.*

Solution:

(i) All file system updates are written to the log. (In some cases, just metadata updates are logged.)

(ii) The changes are actually made to the system by "replaying" the log (going through and applying each update to the in-place file system structure). As long as the log contains the list of changes since the file system was last updated, those changes can be applied at any point, even in the event of a crash mid-update.

3 (continued)

*d. (8 points, **EECE.5730 only**) Explain (i) how file systems that use a linked list to manage free space can make the list faster to search, and (ii) what aspect of free space management becomes slower if you make this change to make list searching faster.*

Solution:

(i) Free space lists can be made faster by reducing the size of the list, either by clustering blocks so that each list node points to a group of *n* blocks, not just an individual block, or by having each node maintain a count of the number of contiguous free blocks after the current one, not pointing to each individually.

(ii) Updating the list to remove used blocks becomes slower, particularly with counting, as an update may not be as simple as removing a single node from the list for each used block.

4. *(15 points)* ***Protection and security***

a. *(9 points) Given the access matrix below, answer questions (i)-(iii).*

Domains	**Objects**						
	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read	owner					
D_2	write* execute	read*	owner	control			
D_3			write				control
D_4	owner	write*					

i. *What privileges can a process in domain D_4 grant to a process in domain D_2 for object F_2? Explain your answer.*

Solution: The write privilege that D_4 has for F_2 can be copied to D_2, as indicated by the asterisk associated with that privilege.

ii. *Can a process in domain D_1 revoke the write privileges of a process in domain D_2 for object F_1? Explain why or why not.*

Solution: No—D_1 is not the owner of F_1, nor does D_1 have control rights over D_2.

iii. *Can a process in domain D_2 grant read privileges for object F_3 to a process in domain D_1? Explain why or why not.*

Solution: Yes—D_2 is the owner of F_3.

b. *(6 points) Explain one benefit of symmetric encryption over asymmetric encryption, and one benefit of asymmetric encryption over symmetric encryption.*

Solution: Key distribution is simpler with asymmetric encryption, as the key used to encrypt messages to a particular receiver can be made public. The decryption key remains private.

Key generation is simpler in symmetric encryption, as the keys are typically just randomly generated bit strings with no special properties.

(Other answers may be acceptable; this answer was largely taken from:
https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf)