CSE 221: Homework 0

Hao-en Sung [A53204772] (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

1 (Hardware/software interaction) Which of the following instructions should be privileged? Very briefly, why?

1.1 Set value of timer

It should be a privileged instruction. The change of timer will affect other programs.

1.2 Read the clock

It should not be a privileged instruction. There might be multiple programs reading the clock at the same time and would not affect each other.

1.3 Clear memory

It depends. If one process wants to clean up its own memory, it needs not be a privileged instruction. Otherwise, it should be, for it might affect the execution of other processes.

1.4 Turn off interrupts

It should be a privileged instruction. Turning off the interruptions might stop other processes from invoking current process.

1.5 Switch from user to kernel mode

It should be a privileged instruction. Switching a process to kernel model means allowance for it to gain higher privilege.

2 (Synchronization)

2.1 Use pseudo-code to implement a thread-safe CountdownEvent using locks and condition variables by implementing the following methods.

The pseudo code is written as Algorithm 1.

2.2 Semaphores also increment and decrement. How do the semantics of a CountdownEvent differ from a Semaphore?

CountdownEvent enters signalled mode once the count reaches 0 and will never return to unsignalled mode again. On the contrary, Semaphore is designed to maintain the available resources, which allows state transits back and forth.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

Algorithm 1 class CountdownEvent

```
1:  bool st;                                          ▷ st is true for nonsignalled mode
2:  int cnt;
3:  Mutex mutex;
4:  ConditionVariable cv;
5:  function CountdownEvent(int count)
6:      st = (cnt > 0);
7:      cnt = count;
8:  function increment
9:      mutex.lock();
10:     if st is true then
11:         cnt += 1;
12:     mutex.unlock();
13: function decrement
14:     mutex.lock();
15:     if st is true then
16:         cnt -= 1;
17:         if cnt == 0 then
18:             st = false;
19:             cv.notify();
20:     mutex.unlock();
21: function wait
22:     mutex.lock();
23:     if st is true then
24:         cv.wait();
25:     mutex.unlock();
```

2.3 Consider a common Barrier synchronization primitive with a constructor Barrier(n) and a method Done(). A group of n threads cooperate on a task. After completing the task, they wait for each other before proceeding by calling Done. Once all threads have called Done, all threads on the Barrier wake up and return from Done. Implement a Barrier using a CountdownEvent.

The pseudo code is written as Algorithm 2.

Algorithm 2 class Barrier

```
1:  CountDownEvent cdEvent;
2:  function Barrier(int n)
3:      cdEvent = CountDownEvent(n);
4:  function done
5:      cdEvent.decrement();
6:      cdEvent.wait();
```

3 (Virtual memory)

I would like to assume that the TLB here is a software TLB. The process that might occur when the instruction is executed can be summarized as following steps.

1. Since the page frame number is not found in TLB, a TLB miss occurs.
2. OS dives into page table to look for corresponding page frame number.
3. The valid bit for page P is off, a page fault occurs.
4. The data is then found on disk and loaded into a free page frame with valid bit on.
5. Mapping between page and page frame is recorded in TLB.
6. The instruction is executed again.

4 (File systems)

4.1 Assuming that the namei cache is empty, succinctly describe what steps Unix will take, in terms of the disk data structures it must read, in order to resolve the path name "/a/b/c" and read the first byte of the file "c". How many disk reads are required?

First step: read '/' from master block in memory, which is not counted as a disk read, and cache it in inode table. Then it follows by multiple disk reads as below.

1. Read inode table on disk to get block for '/'.
2. Read '/' on disk to retrieve content '/a' and cache it in inode table.
3. Read inode table on disk to get block for '/a'.
4. Read '/a' on disk to retrieve content '/a/b' and cache it in inode table.
5. Read inode table on disk to get block for '/a/b'.
6. Read '/a/b' on disk to retrieve content '/a/b/c' and cache it in inode table.
7. Read inode table on disk to get block for '/a/b/c'.
8. Read '/a/b/c' on disk to return its first btye.

Thus, overall 8 disk reads are required.

4.2 Assuming "/a/b/c" has been resolved previously, now describe the steps Unix will take to resolve "/a/b/x" and read the first byte of the file "x". How many disk reads are required?

First step: read '/' from master block in memory, which is not counted as a disk read, and cache it in inode table. Then it follows by multiple disk reads as below.

1. Read inode table on disk to get block for '/a/b'.
2. Read '/a/b' on disk to retrieve content '/a/b/x' and cache it in inode table.
3. Read inode table on disk to get block for '/a/b/x'.
4. Read '/a/b/x' on disk to return its first byte.

Thus, overall 4 disk reads are required.

4.3 A common use of the command "grep" is to search all files in a directory for a string. Assume that the directory "/a/b" has n files. Assuming further that the namei cache starts out empty, how many disk reads are required to search all files using "grep /a/b/*"? State your answer as an expression involving n.

First step: read '/' from master block in memory, which is not counted as a disk read, and cache it in inode table. Then it follows by multiple disk reads as below.

1. Read inode table on disk to get block for '/'.
2. Read '/' on disk to retrieve content '/a' and cache it in inode table.
3. Read inode table on disk to get block for '/a'.
4. Read '/a' on disk to retrieve content '/a/b' and cache it in inode table.
5. Read inode table on disk to get block for '/a/b'.
6. Read '/a/b' on disk to retrieve n different contents '/a/b/x' and cache them in inode table.
7. Read inode table on disk to get block for contents '/a/b/x' (for all x, n files).
8. Read '/a/b/x' on disk to return their contents (for all x, n files).

Thus, overall $6 + 2n$ disk reads are required.

4.4 Now assume that we are using a file buffer cache as well. The file buffer cache will keep in memory all file, directory, and inode blocks that have been previously accessed. When they are accessed again, no disk I/Os are required. Assuming that both the file buffer cache and the namei cache start out empty, how many disk reads are required for the grep command in the previous problem now that we are using a file buffer cache as well?

1. Read information about '/' from master block. Then, retrieve content for '/' from disk and cache information about '/a' in file buffer.
2. Read information about '/a' from file buffer. Then, retrieve content for '/a' from disk and cache information about '/a/b' in file buffer.
3. Read information about '/a/b' from file buffer. Then, retrieve content for '/a/b' from disk and cache information about '/a/b/x' in file buffer (for all x, n files).
4. Read information about '/a/b/x' from file buffer. Then, retrieve content for '/a/b/x' from disk then return those contents. (for all x, n files).

Operations to read information from file buffer are not counted as disk reads. Thus, there are only $3 + n$ disk reads are required.

4.5 Ignoring the issue of space (cache capacity), describe an example situation when an entry in the namei cache should be invalidated?

When a file is renamed (or said moved), namei cache might become wrong.