HW0

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

1.a Set value of timer

Privileged, or program may have power to manipulate other programs, and even OS's behavior

1.b Read the clock

Unprovileged, no harm to read the clock as it can't affect other programs' behavior

1.c Clear memory

Unprivileged, assumeing clearing programs' own memory space, no effect toward other programs

1.d Turn off interrupts

Privileged, or program will have the power to manipulate other programs, and even OS's behavior

1.e Switch from user to kernel mode

Privileged, or user program may have the power to execute command on kernel's behalf arbritarily.

2. (Synchronization) Microsoft .NET provides a synchronization primitive called a CountdownEvent. Programs use CountdownEvent to synchronize on the completion of many threads (similar to CountDownLatch in Java). A CountdownEvent is initialized with a count, and a CountdownEvent can be in two states, nonsignalled and signalled. Threads use a CountdownEvent in the nonsignalled state to Wait (block) until the internal count reaches zero. When the internal count of a CountdownEvent reaches zero, the CountdownEvent transitions to the signalled state and wakes up (unblocks) all waiting threads. Once a CountdownEvent has transitioned from nonsignalled to signalled, the CountdownEvent remains in the signalled state. In the nonsignalled state, at any time a thread may call the Decrement operation to decrease the count and Increment to increase the count. In the signalled state, Wait, Decrement, and Increment have no effect and return immediately.

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

```
class CountdownEvent {
   ...private variables...
   CountdownEvent (int count) { ... }
   void Increment () { ... }
   void Decrement () { ... }
   void Wait () { ... }
}
```

> The CountdownEvent constructor takes an integer count as input and initializes the CountdownEvent counter with count. Positive values of count cause the CountdownEvent to be constructed in the nonsignalled state. Other values of count will construct it in the signalled state.
> Increment increments the internal counter. Decrement decrements the internal counter. If the counter reaches zero, the CountdownEvent transitions to the signalled state and unblocks any waiting threads. Wait blocks the calling thread if the CountdownEvent is in the nonsignalled state, and otherwise returns.

Solution:

```
class CountdownEvent {
```

```
  private mutex lock
  private int count
  private boolean signal // true for signal, false for non-signalled
  private ConditionVariable cv
  CountdownEvent (int count) {
    if count > 0 {
      this.signal = false
    } else {
      this.signal = true
    }
    this.count = count
  }
  void Increment () {
    this.lock.lock()
    this.count += 1
    this.lock.unlock()
    }
  void Decrement () {
    this.mutex.lock()
    if !signal {
      this.count -= 1
      if count <= 0 {
        this.signal = true
        this.cv.notifyAll()
      }
    }

    this.mutex.unlock()
    }
  void Wait () {
    this.mutex.lock()
    if !this.signal {
      this.cv.wait(this.mutex) // Here, mutex will be unlocked and waited on cv notify
    }
    this.mutex.unlock()
  }
}
```

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

Semaphore may be able to get back to locked state and do the signal (reuse for limits resources) while CountdownEvent will never back to block state once its internal counter reach 0.

2.c 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.

```
class Barrier {
  ...private variables...
  Barrier (int n) { ... }
  void Done () { ... }
}
```

Solution:

```
class Barrier {
  private CountDownEvent cde
  Barrier (int n) {
    this.cde = new CountDownEvent(n)
  }

  void Done () {
    this.cde.Decrement()
    this.cde.Wait()
  }
}
```

3. (Virtual memory) Consider when a process issues a read instruction to a virtual address on page P in its address space. Assume that the page table entry (PTE) for P is not in the translation lookaside buffer, that P has been paged out to disk, and that the process has read permission on P. Describe the steps taken by a modern CPU and operating system to ensure that the read instruction successfully completes. State any assumptions you feel you have to make, including whether you assume a hardware or software-managed TLB.

Assumptions:

1. There is a hardware TLB exists, so we don't need to look up the table when there is a miss.

2. The corresponded page is not valid in memory

Steps:

1. When executing the load instruction, TLB failed to find the entry
2. The MMU will try to find the entry in PTE and loaded in TLB
3. Instruction will be executed again and TLB finds the entry, checks the protection, valid bit, and raises a page falult, since the page is on the disk.
4. OS then allocates a frame, loads the data from disk to memory and fix the PTE entry
5. Instruction will be executed, again, and this time TLB will hit and page contains valid data

4. (File systems) The Unix kernel routine that performs path name resolution is called namei. namei takes as input a path name and returns the location on disk of the inode for the last element of the path.

> To speed up path name resolution, namei uses a cache of previously resolved paths. When resolving a path, namei caches all prefixes of the path as well. For example, when resolving "/one/two", namei will cache resolutions for "/", "/one", and "/one/two". Furthermore, when resolving a path name, namei will match in its cache the longest prefix of the path name that it can before making disk requests. For example, if "/one" is in the cache and namei needs to resolve "/one/two", it will start with the cached value for "/one" to resolve "/one/two". For the problems below, you may assume that the master block is already in memory and no disk I/O is required to read it. Further assume that all directories and files are one block in size.

4.a 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?

8 disk reads

1. 2 for path "/" (inode and data block)
2. 2 for path "/a" (inode and data block)
3. 2 for path "/a/b" (inode and data block)
4. 2 for path "/a/b/c" (inode and data block for first byte)

4.b 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?

3 disk reads

1. 1 for path "/a/b" (data block)
2. 2 for path "/a/b/x" (inode and data block)

4.c 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.

`4 + 4 * n` disk reads

1. 2 for path "/" (inode and data block)
2. 2 for path "/a" (inode and data block)
3. `4 * n` for each file matched path `/a/b/*` (for each file, read inode in `/a/b` and its data block; access the file inode and data block for first byte)

4.d 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?

`6 + 2 * n` disk reads

1. 2 for path "/" (inode and data block)
2. 2 for path "/a" (inode and data block)
3. 2 for path "/a/b" (inode and data block, cached)
4. `2 * n` for each file matched path `/a/b/*` (for each file, access the file inode and data block for first byte)

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

Directory renaming/moving to different location