Some Words on Mid-term Questions

Question 1:

For function `do_switch(int *sp1, int *sp2)`, what it does is:
1. Save the stack pointer (sp) to the memory location pointed to by sp1: `sp -> sp1`
2. Load sp from the memory location pointed to by sp2: `sp2 -> sp`

In our homework, we call this function using the following format:
`do_switch(&sp1, &sp2)`

For question a), we can call this function like this:
`do_switch(&sp1, sp2)`

The reason is, since sp2 is an input value, both passing by address and passing by value work. The only difference is, by passing sp2 by address, the new sp is the value pointed to by sp2; while by passing sp2 by value, the new sp is the value of sp2 itself.

For question b), we cannot call this function like this:
`sp1 = do_switch(&sp2)`

This is because the function is called in thread 1 but returns in thread 2; sp1 will never return (meaning the old stack pointer will not be stored anywhere).

Question 2:

This question is meant to let us implement a single function with some variables (integers etc.) and conditions or semaphores; not two functions that each executes on different thread. We count the variable in the single rendezvous function to determine whether its thread 1 or thread 2.

Question 3:

The main thing people didn’t get right is what went into the page entry.
(If you are not very clear on how address translation works, I recommend you review this link: [http://www.cs.nmsu.edu/~pfeiffer/classes/473/notes/intelvm.html](http://www.cs.nmsu.edu/~pfeiffer/classes/473/notes/intelvm.html))

Question 4:

For LRU, it is recommended to maintain the page numbers in order. It is unlikely to make a mistake, for example, to keep the most recently used page number on the top and the least recently used page number on the bottom (refer to the solution).
Hardware Virtualization and Virtual Machine

Virtualization
Systems like VMWare and VirtualBox let us run unmodified OS on top of another one. We can also run multiple copies of OS on the same machine.

Why can’t you run an OS on top of another OS (without the help of virtualization)?
An OS cannot run on an OS as an application does. Applications run in user mode; but OS needs to run in supervisor mode to do some privileged work. Consider the example if Linux runs on top of Windows. When an application in Linux tries to do a system call, it finally goes to Windows kernel instead of Linux kernel (because Windows kernel is the one that directly talks to hardware). Therefore, it doesn’t work.

Emulation
To solve the problem above, the easiest way is emulation.

Software emulation: Basically we write a program to emulate the hardware: variables for each of the registers in CPU, a loop to read instructions, code for instructions like MOV, JUMP and POP, etc. It is similar to how a simple JVM works.

The main disadvantage of emulation is it is slow. It works well only if we emulate a slower machine using a fast machine.

Hardware Virtualization
The better way to solve the problem is to use hardware virtualization.

The idea is to essentially run the entire machine so both the applications and OS are in user mode. They are run as much as they can and the rest is emulated.

Example of Loading CR3
Most of the time it’s running at full speed; the actual virtualization is happening only on the privileged operations that we need to emulate.

The diagram on the left below illustrates the structure of a single machine. On the right, it shows how the machine that runs virtual machine is structured. Each guest OS runs directly on top of the hypervisor. They are like processes and threads to the hypervisor. Therefore, we are able to run multiple copies of OS at the same time and these copies can be switched pretty much like context switching. Also, the virtual machines share the time slices in a way similar to the time-sharing mechanism we’ve learned.

**How to fully virtualize?**
- run guest OS in user mode
- privileged instructions trap
- emulate them

**Virtual Devices**
How do we get input and output from the virtual machine?
The answer is to use virtual devices.

To create a virtual device, we define a memory range in the hypervisor. And we make sure that no matter what the guest OS is doing with the page tables; we always setup entries such that there is a faulting entry with this address base. Then page fault handler will go through the hypervisor and check that if this fault is for an address in one of our virtual device bases.

Example of a simple virtual hard drive adaptor
The idea for virtual devices is, just like we trap and emulate instructions, we trap and emulated on address ranges.

**What do we have to virtualize?**

- **trap handlers**
  Any sort of trap that goes into the supervisor mode has to be emulated by the hypervisor.

- **page tables**
  We need to control page tables when the guest kernel is running.

- **system calls**
  Same reason as traps.

- **interrupts**

  ![Diagram of Virtual Timers and Real Timers](image)

  Example of Timer

- **page faults**

- **memory**

  Memory map in virtual machine

  ![Memory Map Diagram](image)

To map the virtual memory address base to the real memory address base, we make sure that the page tables that the hardware sees are not the same as the guest OS sees. The general idea is we need to keep different copies of everything for the hardware and the virtual machine.

For example, when the code is running in the kernel of guest OS, it thinks it has a CR3, a page directory and a page table. However, the real ones are in different memory locations. This is illustrated in the diagram below.
In order to achieve fully virtualization, we need to be able to trap all the privileged instructions when they run in user mode. We cannot have any privileged instruction that don’t do anything in user mode.

A paper for reference: Popek & Goldberg ‘74

Q & A

**Compared to fully virtualization, is there any other kind of virtualization?**
There is another type of virtualization called paravirtualization. In paravirtualization, the hypervisor exports a modified version of the underlying physical hardware. The exported virtual machine is of the same architecture, which is not necessarily the case in emulation. Instead, targeted modifications are introduced to make it simpler and faster to support multiple guest operating systems.

**Is there anything like, for example, memory sharing between virtual machines?**
There is no reason why we couldn’t share memory between two virtual machines in the same way that we share processes. However, typically this is not done.