Review: high level issues

- In the memory hierarchy, which one is the fastest, slowest, largest, smallest?
  - Registers, cache, memory, disk, tape
- Difference between kernel mode and user mode?
- Does kernel mode need to be supported by hardware? Why?
- What is protected instruction? Give an example?
- What should be execute in kernel mode?
- Why does OS need to provide memory protection?
- Give an example that architecture changes/trends impact OS design
Review: Events

- True or false: the only way to get into OS is through events
- Difference between exceptions and interrupts?
- Difference between system call and interrupt?
- Difference between function call and system call?
## Simple Self-Test

<table>
<thead>
<tr>
<th>Statement</th>
<th>True or false?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The operating system on your laptop is running all the time.</td>
<td></td>
</tr>
<tr>
<td>2. Directly accessing device registers is a privilege instruction.</td>
<td></td>
</tr>
<tr>
<td>3. Accessing registers SP (stack pointer) and PC (program counter) is a</td>
<td></td>
</tr>
<tr>
<td>privilege instruction.</td>
<td></td>
</tr>
<tr>
<td>4. System calls are implemented via interrupts. True or false?</td>
<td></td>
</tr>
<tr>
<td>5. When user level programs make a system call, it first needs to change</td>
<td></td>
</tr>
<tr>
<td>the mode register to kernel mode, and then &quot;traps&quot; into the kernel.</td>
<td></td>
</tr>
<tr>
<td>6. A user level program needs to go through the operating system to access</td>
<td></td>
</tr>
<tr>
<td>the physical memory so that it won't mess other program's data.</td>
<td></td>
</tr>
<tr>
<td>7. A program's data resides only in the physical main memory.</td>
<td></td>
</tr>
<tr>
<td>8. <code>printf()</code> is a library call, not a system call.</td>
<td></td>
</tr>
<tr>
<td>9. Microsoft power point on your laptop can directly access the display.</td>
<td></td>
</tr>
<tr>
<td>10. When a user level program has a 'segmentation fault', it will be</td>
<td></td>
</tr>
<tr>
<td>killed by the hardware.</td>
<td></td>
</tr>
</tbody>
</table>
A simple program in C

```c
void fun()
{
    int locVar = 0;
    locVar++;
}
```

```
fun:
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl $0, -4(%ebp)
addl $1, -4(%ebp)
leave
ret
```
Program Stack

Before calling the function:
- esp

Just after call instruction executed:
- return addr
- esp (saved ebp)

After creating space for local variables:
- esp
- ebp (saved ebp)

After returning from function:
- esp
- ebp (space for local variables)
Background: Program Stack (or Call Stack)

- https://www.youtube.com/watch?v=Q2sFmqvpBe0