Page Table - per process data structure

Physical memory

CPU

OS
July 5, 2017: CS 537-Intro to Operating Systems
Worksheet 1 - Segmentation

Assume a system that uses segmented virtual memory. The segmentation that this system uses is pretty simple: an address space has just two segments: segment 0 and segment 1; further, the top bit of the virtual address generated by the process determines which segment the address is in: 0 for segment 0 (where, say, code and the heap would reside) and 1 for segment 1 (where the stack lives). Segment 0 grows in a positive direction (towards higher addresses), whereas segment 1 grows in the negative direction.

```
----------- 1KB
|          |
| Stack    |
|          |  v
-----------
| (free)   |
| (free)   |
| (free)   |
-----------

^      |Code & Heap|
----------- 0KB
```

address space size = 1K.
phys. memory size = 16K.

Segment register information:

Segment 0 base (grows positive) : 0x00001ae (decimal 6890)
Segment 0 limit : 472

Segment 1 base (grows negative) : 0x00001254 (decimal 4692)
Segment 1 limit : 450

For each virtual address, either write down the physical address it translates to OR write down that it is an out-of-bounds address (a segmentation violation).

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address OR Seg Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000b (decimal: 523)</td>
<td>Seg Fault</td>
</tr>
<tr>
<td>0x0000019e (decimal: 414)</td>
<td>6898 + 414 = 7304</td>
</tr>
<tr>
<td>0x00000322 (decimal: 802)</td>
<td>4470</td>
</tr>
<tr>
<td>0x00000136 (decimal: 310)</td>
<td>6890 + 310 = 7200</td>
</tr>
<tr>
<td>0x000001e8 (decimal: 488)</td>
<td>Seg Fault</td>
</tr>
</tbody>
</table>

3 → 01 → Stack.
32-bit machine ISSUES WITH PAGING!

\[ 2^{32} = 4 \text{ GB} = (2^2 \times 2^{30}) \]

Page size = 4 KB = \[ 2^{12} \]

\[ \frac{2^{32}}{2^{12}} = 2^{20} \]

Virtual pages = \[ \frac{2^{20}}{2} \approx 1 \text{ million.} \]

**PT**

```
5
3
7
2
```

**PTE**

1 PTE = 4 bytes

\[ 2^{20} \times 4 = 4 \text{ MB} \]

10 processes

\[ 40 \text{ MB} \]

400 MB

Memory Waste!
What h/w does on each memory reference?

H/W:
1. extract VPN from V.A.
2. access memory to get the PTE
   EXPENSIVE
3. Extract PFN from PTE.
4. Form the PA using PFN and Offset (from V.A).
5. PA \rightarrow \%eax
   EXPENSIVE
Virtual Memory

Code - entire page 0
+ part of page 1

Heap -  part of page 1
+ page 2

Stack - part of page 3.

FREE  → part of page 2  &  part of page 3.

Physical Memory

page 3 FREE

page 2

Code page 0

Stack

heap

VPN 0, 3 → valid
VPN 1, 2 → invalid

PTE

[VR|RW|S other bits] PFN

DISCUSSION AFTER CLASS