PAGETABLE ENTRY

The difference between Interrupt and Page Fault is that Page Fault has to occur before the instruction completes.

A page fault occurs when an instruction references an unmapped address.

The options on page fault are:
1. Segmentation fault
2. Allocate memory
   - Find free page, zero it out, fill the page with proper values and RETURN
3. Copy on Write
If Process P1 has Read/Write access to certain pages, then another process P2 can have read only access to the same pages. On write i.e. if P2 wants to write, then the link to the common page is broken and Copy on Write is used.

4. Go to disk
Here, pages are accessed from file also.

```
1. Main

2. Allocate free page

\[ \times 08000000 \rightarrow /bin/frame + Offset \]
\[ \text{Page Table address} \]
\[ \text{Address in file} \]

Page Table Entries
```

Program (file)
If we are low on memory and have to give up a page, we can null out the entries. The virtual address to file address mapping would still be there and using this, the part of file can be accessed (by copying the page to memory) if it is required again.

5. Dirty/Clean pages

On write, when the pages get filled with values, we mark them as dirty.
If we try write on the page we were reading, page fault occurs and then, the page is marked as dirty.
If Dirty bit is set, we cannot null the entries directly. Before deallocating a page, the value has to be saved to the file, then page is made 'Clean' and then the entry is nullified.
Note - Anonymous pages start out 'Dirty'
Memory is allocated as and when required. When memory is squeezed, pages have to be given up. Space is allocated in page file. The page contents are then copied to the page file and then the pages are removed from the Page Table. Now, on Read/Write, buffer is allocated, it goes to the file and copies the page contents in the memory.

**PAGE REPLACEMENT**

```
Virtual Address Space                   Backed by memory
                                               Physical Memory

P1  
    

P2  
    

P3  
    
```

At time t, what is the best page to evict from memory?

*Page Access trace*

Record every single memory access. Repeated pages can be collapsed into one.
Page Replacement Techniques:

1. **FIFO** – Kick out the last page in the list of accessed pages.
2. **LRU** – If a page is accessed multiple times, a page which was accessed only once will be moved out even if it was accessed after the first access of the multiple accessed page.
3. **OPT** – Move out the page which is least likely to occur in future. It is practically not implementable but used for comparison.
   FIFO has same efficiency as randomly picking up a page.
   LRU is the one that is widely used.

An example:
Consider page table that can fit in only 3 pages.

<table>
<thead>
<tr>
<th>Pages to be accessed</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>H</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LRU</td>
<td>H</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OPT</td>
<td>H</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus,
Using FIFO, 4 hits and 16 Misses are obtained.
Using LRU, 5 hits and 15 Misses are obtained.
Using OPT, 9 hits and 11 Misses are obtained.

What exactly happens?

Page fault

Page 1

Setup Page 1

Page fault

Page 2

Setup Page 2 (Page 2 is now in page table)
Software cannot directly know if page is in Page Table or not. That's why, FIFO was used till long as it is easy to implement.

Implementation

Check page $P$:
- Clear $P.A$ Bit
- Wait a While
- Check $P.A$ Bit

FIFO

Go in sequence. If page not found, add the new page at top and remove old page from bottom.
If page not found, then go through the pages in sequence. If a page has been accessed (i.e. A bit is set), then move it to top and clear 'A'. If not accessed (A bit is already 0), then remove that page.

**CLOCK**

While (hand->A = 1)  
  hand->A = 0  
  hand = head->next  
  evict head

If page is found, then A bit is set. If page is not found, then page fault occurs and one of the above techniques is followed.

Topics covered in the lecture:
- Process Address Space
- Page Virtual Memory
- 2 – level page table
- Address translation
- TLB
- Page Faults, Demand loading
- C.O.W.
- Page Replacement