Think Global, Act Local (?)

Local vs. Global Page Replacement

- Local: Select victim only among allocated frames
  - Equal or proportional frame allocation
- Global: Select any free frame, even if allocated to another process
  - Processes have no control over their own page fault rate

Brother, can you spare a frame?

FIFO

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Requests		a	b	c	d	a	b	c	d	a	b	c	d
0	a	a	a	a	d	d	d	c	c	c	b	b	b
1	b	b	b	b	b	a	a	a	d	d	d	c	c
2	c	c	c	c	c	c	b	b	b	a	a	a	d
Faults					X	X	X	X	X	X	X	X	X

Brother, can you spare a frame?

FIFO

Time	0	1	2	3	4	5	6	7	8	9	10	11	12
Requests		a	b	c	d	a	b	c	d	a	b	c	d
0	a	a	a	a	a	a	a	a	a	a	a	a	a
1	b	b	b	b	b	b	b	b	b	b	b	b	b
2	c	c	c	c	c	c	c	c	c	c	c	c	c
3	-	-	-	-	d	d	d	d	d	d	d	d	d
Faults					X								

So, what's wrong with global replacement?

Demand May Exceed Resources

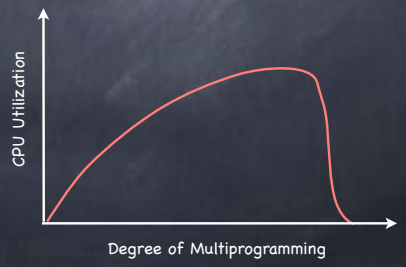
- Demand paging enables frames to cache the currently used part of a process VA space
- If the cache is large enough, hit ratio is high
  - few page faults
- What if not enough frames to go around?
  - should decrease degree of multiprogramming
    - release frames of swapped out processes
    - reduce contention over limited resources

What May Happen Instead

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!

What May Happen Instead

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!
- Thrashing
  - process spends all its time swapping pages in and out



Locality of Reference

- If a process access a memory location, then it is likely that
  - the same memory location is going to be accessed again in the near future (temporal locality)
  - nearby memory locations are going to be accessed in the future (spatial locality)
- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions

Tracking Locality

- When a process executes it moves from locality (set of pages used together) to locality
  - the size of the process' locality (a.k.a. its working set) can change over time
- Goal: track the size of the process' working set, dynamically acquiring and releasing frames as necessary

The Working Set Model

- Define a WS window of Δ references
- Goal: Keep in memory a process' WS
  WS_i = distinct pages p_i referenced in latest Δ
  - Δ too small does not cover locality
  - Δ too large covers many localities
- Thrashing if $\Sigma_i WS_i$ > # frames
  - if so, swap out one of the processes
- If enough free frames, increase degree of multiprogramming

WS Page Replacement

$\Delta = 4$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	$t=0$ ●										
Page b											
Page c											
Page d	$t=-1$ ●										
Page e	$t=-2$ ●										
Faults											

● page mapped to a frame

● page fault & page mapped to a frame

● page referenced & mapped to a frame

WS Page Replacement

$\Delta = 4$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	$t=0$ ●	●	●	●						●	●
Page b					●	●	●	●			
Page c		●	●	●	●	●	●	●	●	●	●
Page d	$t=-1$ ●	●	●	●	●	●	●				●
Page e	$t=-2$ ●	●					●	●	●	●	●
Faults		X			X		X			X	X

● page mapped to a frame

● page fault & page mapped to a frame

● page referenced & mapped to a frame

Computing the WS

- Use interval timer τ, the R bit, and k extra bits per page
- Define $\Delta = \tau \times k$
- When τ elapses, shift right once the k bits, copy R bit in most significant bit, and reset R
- If one of the k bits is 1, the corresponding page is in WS

WS and Page Fault Frequency

- When too many page faults, increase WS; when too few, decrease it

Keep time t_{last} of last page fault

On page fault:

1) add faulting page to the working set

2) if $t_{\text{current}} - t_{\text{last}} > \tau^*$ (threshold), then unmap all pages not referenced in $[t_{\text{last}}, t_{\text{current}}]$

PFF Page Replacement $\tau^* = 2$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	•										
Page b											
Page c											
Page d	•										
Page e	•										
Faults											
$t_{\text{current}} - t_{\text{last}}$											

PFF Page Replacement $\tau^* = 2$

Time	0	1	2	3	4	5	6	7	8	9	10
Requests		c	c	d	b	c	e	c	e	a	d
Pages in Memory: Page a	•	•	•	•							
Page b					•						
Page c		•	•	•	•						
Page d	•	•	•	•	•						
Page e	•	•	•	•							
Faults		X			X						
$t_{\text{current}} - t_{\text{last}}$		1			3						

PFF Page Replacement

$$\tau^* = 2$$

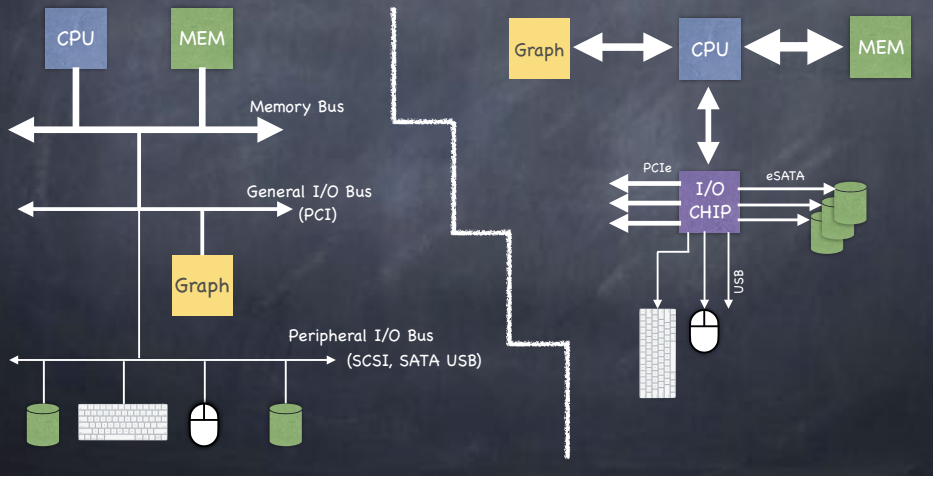
Time		0	1	2	3	4	5	6	7	8	9	10
Requests			c	c	d	b	c	e	c	e	a	d
Pages in Memory	Page a	•	•	•	•						•	•
	Page b					•	•	•	•	•		
	Page c		•	•	•	•	•	•	•	•	•	•
	Page d	•	•	•	•	•	•	•	•	•		•
	Page e	•	•	•	•			•	•	•	•	•
Faults			X			X		X			X	X
$t_{\text{current}} - t_{\text{last}}$			1			3		2			3	1

I/O Devices

You Need to Get Out More!

- How does a computer connect with the outside world?

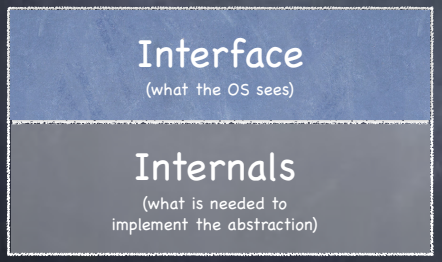
I/O Architecture



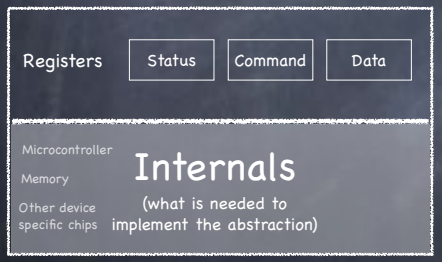
Interacting with a Device

Abstraction
(what the user sees)

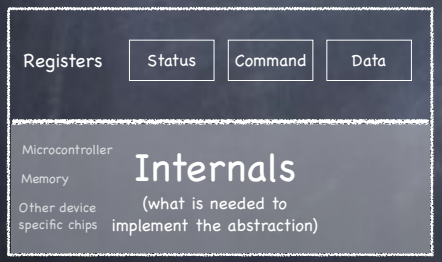
Interacting with a Device



Interacting with a Device



Interacting with a Device



- OS controls device by reading/writing registers

```
while (STATUS == BUSY)
    ; // wait until device is not busy
write data to DATA register
write command to COMMAND register
    // starts device and executes command
while (STATUS == BUSY)
    ; // wait until device is done with request
```

Tuning It Up

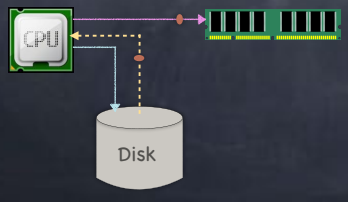
- CPU is polling
  - use interrupts
  - run another process while device is busy
  - what if device returns very quickly?
- CPU is copying all the data to and from DATA
  - use Direct Memory Access (DMA)

```
while (STATUS == BUSY)
    ; // wait until device is not busy
write data to DATA register
write command to COMMAND register
    // starts device and executes command
while (STATUS == BUSY)
    ; // wait until device is done with request
```

From interrupt-driven I/O to DMA

- Interrupt driven I/O
  - Device ⟷ CPU ⟷ RAM

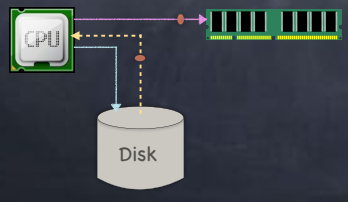
  for $(i = 1 \ldots n)$
  - CPU issues read request
  - device interrupts CPU with data
  - CPU writes data to memory



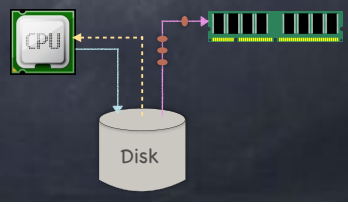
From interrupt-driven I/O to DMA

- Interrupt driven I/O
  - Device ⟷ CPU ⟷ RAM

  for $(i = 1 \ldots n)$
  - CPU issues read request
  - device interrupts CPU with data
  - CPU writes data to memory



- + Direct Memory Access
  - Device ⟷ RAM
    - CPU sets up DMA request
    - Device puts data on bus & RAM accepts it
    - Device interrupts CPU when done



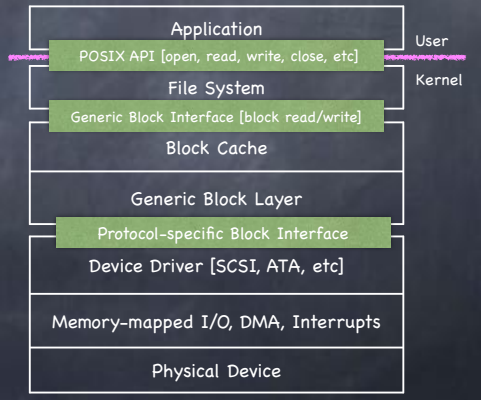
Communicating with devices

- Explicit I/O instructions (privileged)
  - `in` and `out` instructions in x86
- Memory-mapped I/O
  - map device registers to memory location
  - use memory load and store instructions to read/write to registers

How can the OS handle a multitude of devices?

- Abstraction!
  - Encapsulate device specific interactions in a device driver
  - Implement device neutral interfaces above device drivers
- Humans are about 70% water...
  - ...OSs are about 70% device drivers!

File System Stack (simplified)



Persistent Storage

Storage Devices

- We focus on two types of persistent storage
  - magnetic disks
    - servers, workstations, laptops
  - flash memory
    - smart phones, tablets, cameras, laptops
- Other exist(ed)
  - tapes
  - drums
  - clay tablets



The Oldest Library?

- Ashurbanipal, King of Assyria (668-630 bc)

Magnetic disk

- Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum