Reducing the Storage Overhead of Page Tables

Paged Segmentation

- Page table size for a machine with 64-bit addresses and a page size of 4KB?
  - 2^{52} entries!
- Make pages bigger!
  - internal fragmentation
- Good news!
  - address spaces often organized in segments!
- Use a page table per segment!
  - but how can OS find those page tables?
- Use Base and Bound registers! For each segment
  - Base register stores physical address of corresponding PT
  - Bound Registers stores length (no. of page table entries) of corresponding PT
- Can significantly reduce storage overhead
  - if using only a few contiguous pages in each segment
- But…
  - does not work well if segment is large but sparsely used
  - reintroduces variable length allocation

Paged Segmentation Example

32 bits

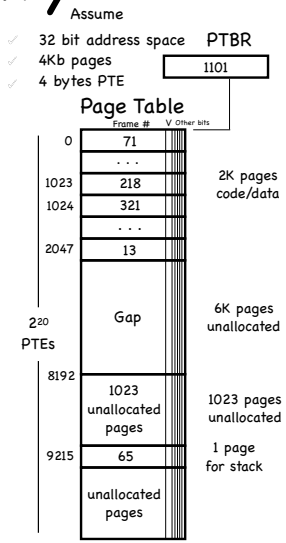
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- What is the size of the VA space, assuming it is byte addressable? 2^{32} bytes (or 4 GBytes)
- How large is a page? 2^{12} bytes (or 4 KBytes)
- What is the maximum value that can be stored in a Bound register? 2^{16} (or 65536)
- If each PTE takes 4 bytes, how many pages are required to store the largest page table that a segment can support?

Page tables can have 2^{16} entries, each 2^2 bytes; since a page holds 2^{12} bytes, the number of pages necessary to store the page table is $2^{18}/2^{12}$ = 64 pages

A More Flexible Way to Leverage Sparsity

- Use a better data structure to express the Page Table
  - a tree!



A More Flexible Way to Leverage Sparsity

- Use a better data structure to express the Page Table
  - a tree!

Assume
- 32 bit address space
- 4Kb pages
- 4 bytes PTE

PTBR: 1101

Page Table

	Frame #	V	Other bits	
0	71			Frame 1101
	. . .			
1023	218			
1024	321			Frame 1102
	. . .			
2047	13			
	Gap			
2^{20} PTEs				
8192	1023 unallocated pages			
9215	65			
	unallocated pages			

A More Flexible Way to Leverage Sparsity

- Use a better data structure to express the Page Table
  - a tree!

Assume
- 32 bit address space
- 4Kb pages
- 4 bytes PTE

PTBR: 1101

Page Table

	Frame #	V	Other bits	
0	71			Frame 1101
	. . .			
1023	218			
1024	321			Frame 1102
	. . .			
2047	13			
	Gap			
2^{20} PTEs				
8192	1023 unallocated pages			
9215	65			
	unallocated pages			

A More Flexible Way to Leverage Sparsity

- Use a better data structure to express the Page Table
  - a tree!

PDBR: 1100

	Frame #	V
0	1101	
1	1102	
2	–	
3	–	
4	–	
5	–	
6	–	
7	–	
8	1109	
9		
	Invalid entries	
1023		

Page Directory

	Frame #	Other bits	
0	71		Frame 1101
	. . .		
1023	218		
1024	321		Frame 1102
	. . .		
2047	13		
8192	1023 invalid PTEs		Frame 1109
9215	65		

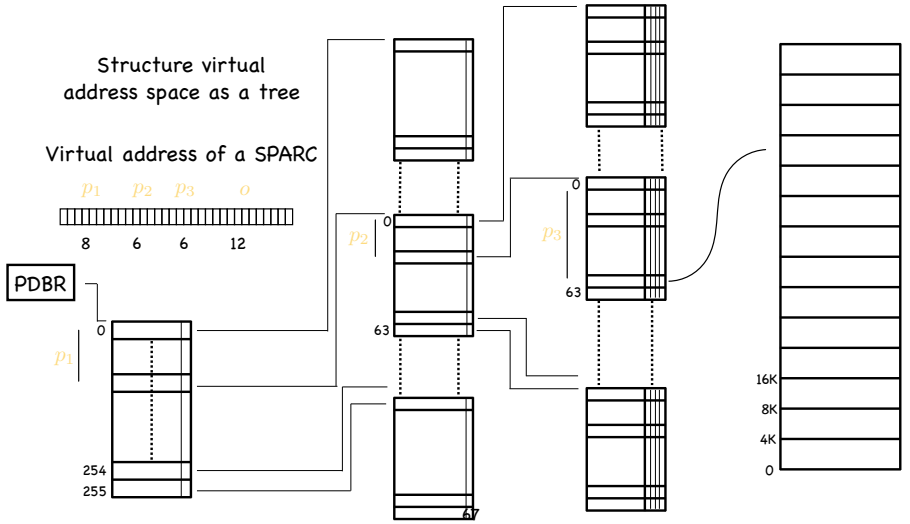
Assume
- 32 bit address space
- 4Kb pages
- 4 bytes PTE

PTBR: 1101

Page Table

	Frame #	V	Other bits	
0	71			Frame 1101
	. . .			
1023	218			
1024	321			Frame 1102
	. . .			
2047	13			
	Gap			
2^{20} PTEs				
8192	1023 unallocated pages			Frame 1109
9215	65			
	unallocated pages			
				Frame 2124

Multi-level Paging



Aside

Checkin with one condition variable

```
self.allCheckedIn = Condition(self.lock)

def checkin():
  with self.lock:
    nArrived++
    if nArrived < nThreads:
      while nArrived < nThreads:
        allCheckedIn.wait()
    else:
      allCheckedIn.broadcast()
      nArrived = 0
```

What's wrong with this?

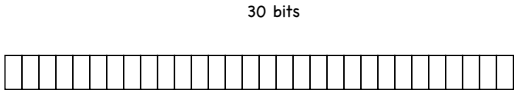
Checkin: 2 condition variables

```
                        self.allCheckedIn = Condition(self.lock)
                        self.allLeaving = Condition(self.lock)
def checkin():
    nArrived++
    if nArrived < nThreads:              // not everyone has checked in
        while nArrived < nThreads:
            allCheckedIn.wait()           // wait for everyone to check in
    else:
        nLeaving = 0                  // this thread is the last to arrive
        allCheckedIn.broadcast()    // tell everyone we're all here!

    nLeaving++
    if nLeaving < nThreads:              // not everyone has left yet
        while nLeaving < nThreads:
            allLeaving.wait()             // wait for everyone to leave
    else:
        nArrived = 0                  // this thread is the last to leave
        allLeaving.broadcast()      // tell everyone we're outta here!
```

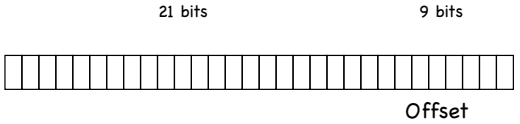
End of Aside

Multilevel Page Table: an Example



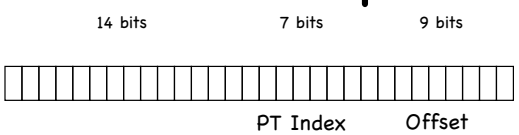
- Suppose page size is 512 bytes
  - offset consumes 9 bits

Multilevel Page Table: an Example



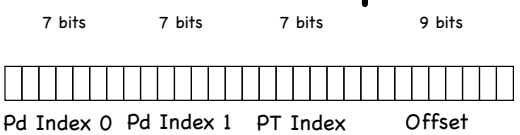
- Suppose page size is 512 bytes
  - offset consumes 9 bits
- Suppose PTE size is 4 bytes
  - How many bits needed by the PT index?
    - a page can store 128 PTEs: page table index consumes 7 bits

Multilevel Page Table: an Example



- Suppose page size is 512 bytes
  - offset consumes 9 bits
- Suppose PTE size is 4 bytes
  - a page can store 128 PTEs: page table index consumes 7 bits
- Page directory still requires 128 pages!
  - we page the Page Directory

Multilevel Page Table: an Example



- Suppose page size is 512 bytes
  - offset consumes 9 bits
- Suppose PTE size is 4 bytes
  - a page can store 128 PTEs: page table index consumes 7 bits
- Page directory still requires 128 pages!
  - we page the Page Directory!

Getting Slooooower

- Multilevel/segmented paging
  - reduce memory overhead of performing address translation
  - ... but increase the time necessary to perform address translation