

CS 3204 Operating Systems

Lecture 18
Godmar Back



Announcements

- Fix remaining project 2 bugs (if any)
 - Reminder: by end of semester, passing students will have provided a deliverable that achieves $\geq 90\%$ test score on project 2 (or a 100% test score on project 3 or 4's regression tests.) – multi-oom not part of these
- Project 3 Help Sessions
 - Thursday 22nd 7-9pm McB 216
 - Friday 23rd 5-7pm McB 216
- Project 3 Design Milestone
 - Monday 26th 11:59pm – no extensions!
- Midterm March 29
 - See announcement + sample midterms on class website



CS 3204 Spring 2007

3/24/2007

2

Virtual Memory

Paging Techniques



Demand paging

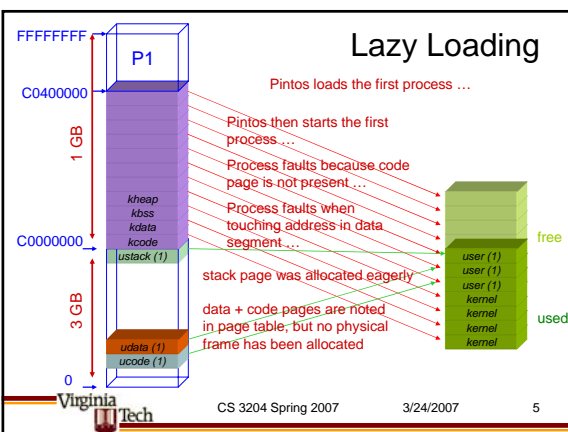
- Idea: only keep data in memory that's being used
 - Needed for virtualization – don't use up physical memory for data processes don't access
- Requires that actual allocation of physical page frames be delayed until first access
- Many variations
 - Lazy loading of text & data, mmap'd pages & newly allocated heap pages
 - Copy-on-write



CS 3204 Spring 2007

3/24/2007

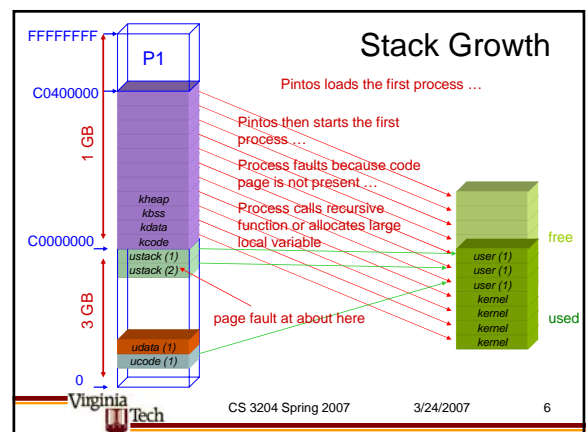
4



CS 3204 Spring 2007

3/24/2007

5



CS 3204 Spring 2007

3/24/2007

6

The diagram illustrates a stack overflow and the resulting page fault handling in a microkernel. On the left, a green vertical bar represents memory, with the address 0x8000 at the top. Four horizontal arrows point from the stack to a box on the right, indicating the stack pointer (esp) values at different stages: 0x8004, 0x8000, 0x7FEC, and 0x7FE4. The box on the right contains the following assembly instructions: `push $ebp`, `sub $20, %esp`, `push %eax`, and `push %ebx`. A red arrow points from the `push %ebx` instruction to a red box labeled "Page Fault!". Below this, a box labeled "intr0e_stub:" contains the following assembly instructions: `...`, `call page_fault()`, and `ret`. A red arrow points from the `call page_fault()` instruction to a light blue box labeled "void page_fault() { get fault addr; determine if page fault is user's; if yes, allocate page frame; install page in page table; terminate process; MMU will walk hardware page table again".

Can resume after page fault and unless $f \rightarrow eip$ is changed) this will retry the faulting instruction (here: push %eax)

– MMU will walk hardware page table again

Virginia Tech

CS 3204 Spring 2007


3/24/2007

7

- # Fault Resumption
- Requires that faulting CPU instruction be restartable
 - Most CPUs are designed this way
 - Very powerful technique
 - Entirely transparent to user program: user program is frozen in time until OS decides what to do
 - Can be used to emulate lots of things
 - Programs that just ignore segmentation violations (!?) (here: resume with next instruction – retrying would fault again)
 - Subpage protection (protect entire page, take fault on access, check if address was to an valid subpage region)
 - Virtual machines (vmware, qemu – run entire OS on top of another OS)
 - Garbage collection
 - Distributed Shared Memory

Distributed Shared Memory

- Idea: allows accessing other machine's memory as if it were local
- Augment page table to be able to keep track of network locations:
 - local virtual address → (remote machine, remote address)
- On page fault, send request for data to owning machine, receive data, allocate & write to local page, map local page, and resume
 - Process will be able to just use pointers to access all memory distributed across machines – fully transparent
- Q.: how do you guarantee consistency?
 - Lots of options



CS 3204 Spring 2007

3/24/2007

9

Heap Growth

FFFFFFFFFF

C0400000

1 GB

P1

kheap
kbss
kdata
kcode
ustack (1)

C0000000

3 GB

0

udata (2)
udata (1)
ucode (1)

Process calls sbrk(addr)

Process faults when touching new memory

Process needs memory to place malloc() objects in

Process faults because code page is not present ...

Pintos then starts the first process ...

Pintos loads the first process ...

user (1)
user (1)
user (1)
kernel
kernel
kernel

free

used

Virginia Tech

CS 3204 Spring 2007

3/24/2007

10

mmap()

Pintos loads the first process ...

Pintos then starts the first process ...

Process faults because code page is not present ...

Process opens file, calls `mmap(fd, addr)`

Process faults when touching mapped file

Page fault handler allocates page, maps it, reads data from disk

free

used

CS 3204 Spring 2007 3/24/2007 11