OS/161 – NEW MEMORY MANAGER

Peter Coffman Ryan Basile

OVERVIEW

- OS/161 is a simple operating system
  - Many of its features are unimplemented, and it's up to you to implement them
- Memory management is especially primitive
  - Memory is never freed
  - Memory is always allocated in contiguous chunks

PROBLEM 1

- OS/161 does not properly deallocate memory when threads exit
  - This caused the test programs for lab 4 to crash after running them 3 times

```
OS/161 kernel [? for menu]: sy2
Starting lock test...
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
panic: locktest: thread_fork failed: Out of memory
sys161: 429496729742640698 cycles (16830378 run, 429496729725810320 global-idle)
sys161:   cpu0: 16830378 kern, 0 user, 0 idle)
sys161: 1774 irqs 0 exns 0r/0w disk 12r/1247w console 0r/0w/1m emufs 0r/0w net
sys161: Elapsed real time: 6.696244 seconds (6.41399e+10 mhz)
sys161: Elapsed virtual time: 5.705627937 seconds (25 mhz)
os161user@os161user-VirtualBox:~/cs4300-os161/root$
```

```
OS/161 kernel [? for menu]: sy3
Starting CV test...
Threads should print out in reverse order.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
scheduler: Dropping thread synchtest.
panic: cvtest: thread_fork failed: Out of memory
sys161: 644245206064255968 cycles (746399833 run, 644245205317856135 global-idle
)
sys161:   cpu0: 746399833 kern, 0 user, 0 idle)
sys161: 447869 irqs 0 exns 0r/0w disk 17r/4470w console 0r/0w/1m emufs 0r/0w net
sys161: Elapsed real time: 4466.591779 seconds (1.44236e+08 mhz)
sys161: Elapsed virtual time: 4466.570238737 seconds (25 mhz)
os161user@os161user-VirtualBox:~/cs4300-os161/root$
```

PROBLEM 2

- OS/161 eagerly allocates all memory of a process up front
  - Sparse access patterns are not supported

```
OS/161 kernel [? for menu]: p testbin/sparse
Running program testbin/sparse failed: Out of memory
Operation took 0.723996600 seconds
OS/161 kernel [? for menu]:
```

OS ANALYSIS

- OS/161 comes with a very basic virtual memory manager,"dumbvm," located in the dumbvm.c file
- It has two empty functions:

```
21  void
22  vm_bootstrap(void)
23  {
24      /* Do nothing. */
25  }
26
```

```
54  void
55  free_kpages(vaddr_t addr)
56  {
57      /* nothing */
58
59      (void)addr;
60  }
61
```

- These are called in main.c and kheap.c
- Also includes some basic address space functions

WHERE THEY'RE REFERENCED

MAIN.C

- Responsible for system boot, shutdown, and running the main kernel.
- It runs vm_bootstrap() during its boot() function

```
ram_bootstrap();
scheduler_bootstrap();
thread_bootstrap();
vfs_bootstrap();
dev_bootstrap();
vm_bootstrap();
kprintf_bootstrap();
```

KHEAP.C

- Subpage allocator
- subpage_kfree() function uses free_kpages() to free whole pages marked "free" given a virtual page address "prpage"
- kfree() also uses it
  - More details next...

KFREE.C

- Relates to the test programs sy2 and sy3:
  - Creates threads, which exit when completed (thread_exit)
  - Exited threads are marked as "zombie threads"
  - The exorcise() function tries to destroy the zombie threads with "thread_destroy(z)"
    - However, thread_destroy() uses kfree(), which doesn't work because free_kpages() is unimplemented!
    - Pages of memory get filled but are never freed
    - Eventually leads to "Out of memory" error

```
74  static
75  void
76  thread_destroy(struct thread *thread)
77  {
78      assert(thread != curthread);
79
80      // If you add things to the thread structure, be sure to dispose of
81      // them here or in thread_exit.
82
83      // These things are cleaned up in thread_exit.
84      assert(thread->t_vmspace==NULL);
85      assert(thread->t_cwd==NULL);
86
87      if (thread->t_stack) {
88          kfree(thread->t_stack);
89      }
90
91      kfree(thread->t_name);
92      kfree(thread);
93  }
```

```
565 void
566 kfree(void *ptr)
567 {
568     /*
569      * Try subpage first; if that fails, assume it's a big allocation.
570      */
571     if (ptr == NULL) {
572         return;
573     } else if (subpage_kfree(ptr)) {
574         assert((vaddr_t)ptr%PAGE_SIZE==0);
575         free_kpages((vaddr_t)ptr);
576     }
577 }
```

POSSIBLE SOLUTIONS

1. Implement the missing functions in dumbvm
   - Create a core map to keep track of pages (single table)
     - Modify getpages() to reference the core map
     - Add other functions to handle pages in the new core map
   - vm_bootstrap() to initialize the core map
2. Replace dumbvm with a more robust virtual memory manager

OUR SOLUTION

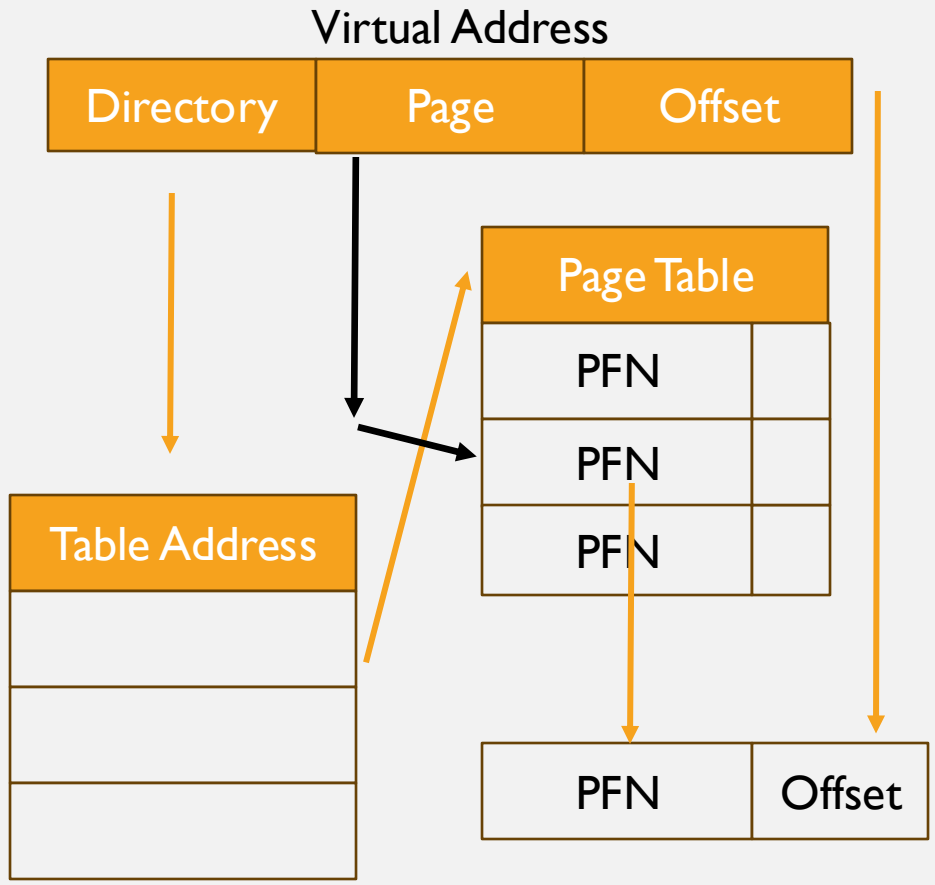
Replace dumbvm.c with vm.c

Uses a two-level page table

Saves memory

Add necessary functions, modifying preexisting ones from dumbvm

vm_bootstrap, free_kpages, alloc_kpages, alloc_ppages...



FILE CHANGES

Many files had to be altered to allow two-level page tables.
A complete list is in the included documentation file.

- vm.c (new file)
  - Bootstrap function that allocates the physical memory on boot
  - Page allocation & free functions
    - For virtual pages & physical pages
- Page table nodes
- Fault handler
  - What happens if address is invalid?
- addrspace.c
  - Main structure for two levels
  - Defines virtual bases & number of pages for each level in an address space
  - Defines the stack

FILE CHANGES

- vm.h (new file)
  - Structure for page tables
  - Defines macro KVADDR_TOPADDR
    - Converts virtual address to physical address
  - Struct pt_node
  - Struct page_table
- addrspace.h
  - If compiled without dumbvm, redefine the variables:
    - vaddr_t, size_t, vaddr_t, page_table *pt
  - New function to check for legal virtual memory access: as_check_access
    - Prevent reading from subpage which isn't part of a page

TEST PROGRAMS

Two new programs were created to test the functionality of the new virtual memory system.

vm_t1.c

- A simple function that returns 0

vm_t2.c

- Creates a large array that's twice the size of physical memory
  - 2 x 128 x 4096 (2 x # of pages x page size)
- Iterates through it, setting the value of the array at "i" to one
- Increments "i" to 10 times the page size (4096 x 10)
- Return 0 when completed

THE RESULTS

- Lab 4 test programs sy2 and sy3 no longer crash after three runs
- The sparse program returns 0
  - After a string of unknown syscall messages
- Both of our custom test programs return 0
  - No crashes or "out of memory" errors

```
OS/161 kernel [? for menu]: sy2
Starting lock test...
Lock test done.
Operation took 0.047091040 seconds
OS/161 kernel [? for menu]: sy2
Starting lock test...
Lock test done.
Operation took 0.047067200 seconds
OS/161 kernel [? for menu]: sy2
Starting lock test...
Lock test done.
Operation took 0.047067200 seconds
OS/161 kernel [? for menu]:
```

```
OS/161 kernel [? for menu]: p /testbin/vm_t2
0
Operation took 0.975590040 seconds
OS/161 kernel [? for menu]:
```

```
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
Unknown syscall 6
0
Operation took 0.542768080 seconds
OS/161 kernel [? for menu]:
```