CSE 120 - Discussion session 9

June 3, 2019

Logistics

- June 5, Wednesday at 11:59pm - Homework 4
- June 7, Friday at 11:59pm - Project 3
- NO EXTENSIONS

Part 1 - Overview

- Initialize all TranslationEntries as invalid in VMKernel.loadSections
- Allocate physical page frames but don’t load page
- Machine will trigger page fault exception
- Modify VMProcess.handleException to handle this
- Add method to prepare the requested page
- Note - can fault on code page, data page, stack page or argument page
- Set TranslationEntry to true after paging in faulted page
- Modify rVM/wVM to handle invalid pages and page faults

Part 2 - Overview

- Don’t allocate physical pages in loadSections
- Extend page fault handler to allocate a physical page frame on the fly

- Implement page replacement(Clock algorithm)
- Page fault handler should pick victim page and evict it
- Check if victim page is dirty or not(to decide whether or not to write it to swap)
- Read in contents of faulted page - from exe file or swap file

Part 2 - Overview

- Implement swapping
- Maintain swap file
  - Evict pages out of memory into swap
  - Swap pages in from swap to memory
- Maintain IPT to keep track of which page in memory corresponds to which process
- Implement pinning

- Pay attention to Race Conditions!

Swapping - Additional notes

NOTE : If you have already implemented this some other way, it’s fine! You do not need to change it. Just make sure you test thoroughly!

- When a page is swapped in, its dirty bit is set to false
- We do not free the swap page(maintain the vpn➔spn mapping)
- If the page is modified, set dirty bit to true. Else, can simply overwrite the page in memory because we can always retrieve it from the swap file
- This way, we avoid writing to the swap file unnecessarily

- When to free a swap page?
  - When a process terminates, it frees all the physical pages and swap pages it holds.

Pinning

- Why do we need pinning?
- When do we pin a page?
  - In rVM/wVM
- Synchronization for page pinning?
  - Avoid holding the lock for the entire duration that the page is pinned

Testing

- Simple tests first. Then complicated tests.
  - Simple test : halt, write1, write4
  - Hard tests : write10, snake, swap4, swap5
- Single process first. Then multiple processes.
  - Start testing with just a single process
  - Then fork multiple processes concurrently

Testing

- Change number of physical pages
  - Use command line argument
    - nachos -m 16 ...
  - Or directly edit nachos.conf
    - Processor.numPhysPages = 16
- Stress test your code
  - Keep decreasing number of pages
  - Don’t fork too many processes concurrently if numPhysPages is too small
    - Will lead to thrashing
  - Think about(and try out) what is the minimum number of physical pages needed to run a single process

Testing

- We will not be testing join, or any of the corner cases for the I/O syscalls
- We already tested and graded that functionality in project 2
- Focus in project 3 is on functionality given in the project description

Debugging

- Print ➜ most useful tool
  - Print out which phy page is allocated to which virtual page of which process?
  - Print out which swap page holds which virtual page
  - Print freePageList and freeSwapList in critical places
  - Print IPT whenever there is a change
  - ...
- Compare printed results with expected results

Debugging

- Add breakpoints and trace execution
  - Add a breakpoint in handleException: What is this exception? Syscall or pageFault? Which page? Which process? Is it reasonable to have this exception at this time?
  - Add a breakpoint in the clock algorithm: Which page is selected as the victim? What is the current state of the IPT?
  - ...
- Debugging tools
  - Lib.debug()
  - Lib.assertTrue()

```
// Change from invalid to valid
Lib.assertTrue(pageTable[vpn].valid == false);
handlePageFault(vpn);
Lib.assertTrue(pageTable[vpn].valid == true);
```

General tips

- Additional proj2 tests added to proj2 autograder
  - To check proj2 functionality
- First focus on implementing proj3 functionality
- Read Piazza posts @607 and @608

Before submission

- Create a file named README in the proj3 directory
- List the members of your group
- Provide a short description of the following:
  - what code you wrote
  - how well it worked
  - how you tested your code
  - how each group member contributed to the project
- Why?

  Easier for us to understand what you did as we grade your project in case there is a problem with your code