CS50300: Operating Systems
Lab6 Answers

Zichen Wang
wang4113@purdue.edu

December 5, 2018

WARNING: If run two or more processes using virtual heap, the print device may be stuck due to the context switch during `kprintf` in the test processes. The reason might be the conflict between `kprintf` and the backing store. If this happens, please reduce the number of `kprintf` or just comment `kprintf` and rerun the test cases. Another way is to modify the time slice in order to avoid context switch during `kprintf`.

1 Finding the backing store for a virtual address

1. Since we only have 8 backing stores and no store can be mapped to more than one process at any time, we can use a mapping table to record the process id and the range of virtual memory. We create an array of 8 entries of the type of the mapping.
2. When a new process created by `vcreate()`, we will allocate backing stores for it. Specifically, if `hsize_in_pages` is larger than 200, we can split it into more than one segments and save each segment into a backing store.
3. When finding the backing store for a virtual address, we just enumerate each backing store to see whether this backing store belongs to the current process and whether this virtual address is located in this backing store.

2 Evaluation

2.1 Evaluation on the backing store

1. Create a process by `vcreate()` with `hsize_in_pages=500` and a parameter c. In this process, we allocate a 60*4096 char array, and assign each element with a smaller number (< 128) related to i and c. Then, we check whether the value of each element is correct or not. This case will test two things. First, the allocation of backing stores is correct. Second, the basic reading and writing a backing store is correct. We expect that the value is correct when reading from the backing store.

2. Create two such processes and test the correctness of concurrency. We expect that the values of both processes are correct after writing and reading the backing stores.

2.2 Evaluation on page replacement policy FIFO

1. Create a process by `vcreate()` with `hsize_in_pages=100` and a parameter c. In this process, we allocate a 60*4096 char array, and assign each element with a smaller number (< 128) related to i and c. Then, we check whether the value of each element is correct or not. Hence, there will be some pages that will be evicted during page fault. We expect that the number of evicted frame is increasing from 2024 to 2073 and then goes back to 2024.
2. Create two such processes and test the correctness of concurrency. We expect that each process use 25 frames, since they are the same. Create two different processes where one ends quickly than the another. We expect that the allocated frames are organized as a queue.

2.3 Robustness

Create 5 processes using 8 backing stores. Each process will get virtual memory for an array and assign some values in it. We expect that all of elements have the correct values.

3 Bonus

1. Global Clock page replacement algorithm has been implemented. We maintain a global variable `frame_last_stopped` defined in `paging_init.c`. When all free frames are used, we start find a frame from `frame_last_stopped`, and traverse the list of frames in a round-robin fashion.
2. We gives a frame whose `pt_acc=1` a "second chance" before reclaiming. We gives a frame whose `pt_dirty=1` a "third chance" before reclaiming. To avoid the side effect of modifying page table, we add a dirty bit in `inverted_page_table` to indicate whether this frame should be write back to the backing store when using Global Clock algorithm.
3. Call `pgrpolicy(1)` in `paging_init.c` which is change the page replacement policy to Global CLOCK algorithm for evaluation. We create the same process as 2.1, and tracks the number of page faults on each policy. For FIFO, the number of page faults is 123 and the number is 112 for Global Clock algorithm.