CS50300: Operating Systems

Lab5 Answers

Zichen Wang
wang4113@purdue.edu

November 19, 2018

1 Inverted page table

1. This data structure *inverted_page_table* maintains all physical frames. Since we use an array of size NFRAMES = 3072, the physical frames [1024, 2023] are mapped to index [0, 999] of this array, and the physical frames [2024, 4095] are mapped to index [1000, 3071].

2. For each entry in *inverted_page_table*, there are four variables. *fstate* is to define the status of a frame, which are *F_FREE*, *F_USED_PAGE*, *F_USED_PD*, *F_USED_PT* and *F_SHARED_PT*. *pid* stores the process using this physical frame. *virt_page_num* stores the virtual page number mapped to this physical frame. *reference_count* is the number of present entries in the page table.

2 Evaluation

2.1 First Step

Create several processes by create() without per-process private heap.

1. Create 3 processes by create(). Each process will allocate 5000 bytes from physical heap by getmem(). We will print the base address of this array in each process. We expect that these 3 base addresses are different, and page faults will not occur.

2. After the test, we find that the base address of each process is 0x199FD0, 0x19B358, and 0x19C6E0 respectively. Indicated by the hook print, page fault does not occur, and we only create some page directories for all processes and 5 shared page tables. The test code is in test_app.c .

2.2 Second Step

Add support for per-process private heap using vcreate().

1. Create another 3 processes by vcreate(). Again, each process will allocate 5000 bytes, but this allocation will use virtual heap by vgetmem(). We expect that these 3 base addresses are the same, but the contents will be different. The page fault will occurs during vgetmem().

2. The result is the same as expectation. The array base address of each process is 0x1000000. Since we give each process a different argument that will be stored in the array, we can see the array[100] of each process is different. The number of page faults is two for each process and these 2 virtual addresses are 0x01000000 and 0x01001388 respectively, because we need to modify the free list at two different locations. The test code is in test_vm.c .

3. Next, we create 2 more complicated processes. Each process will first allocate 4000 bytes by vgetmem() for an integer array. Then, it will construct a math sequence by the given arguments. Next, it will free these 4000 bytes by vfreemem(), and allocate another 40000 bytes by vgetmem(). At last, it will construct a new math sequence. We expect that the results of these two sequences are correct. After the test, we find that all numbers are correct and the number of page faults is corresponding to how many pages the process allocates. The test code is in test_vm_2.c .