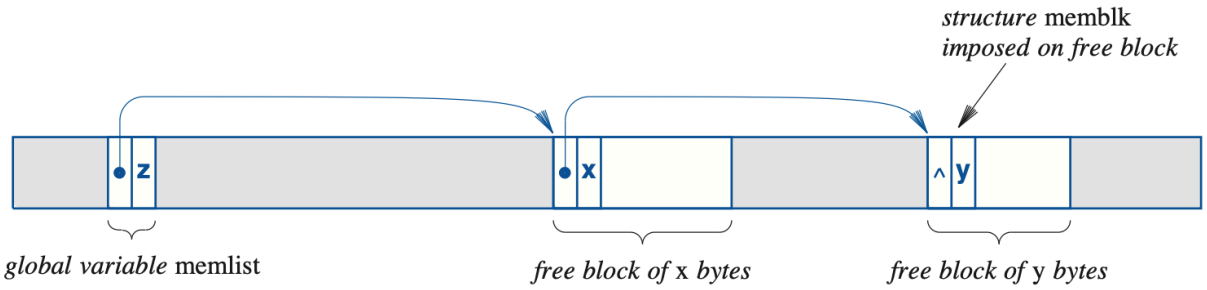
Optimal Xinu Paging

Lab 5

Vikrant Gajria
CS503
vgajria@purdue.edu

Part 1

Per-Process Kernel Structures for Space-optimal, Isolated VHeap Management



Description:

I have copied the implementation of shared heap management - getmem() and freemem() - into virtual heap management - vmhgetmem() and vmhfreemem(). These are *exactly the same* as how free list is optimally managed in Region C, but they cleverly work in Region VF. struct memblk is used to manage metadata in the free address space.

Advantages:

1. Per-process freelist is *isolated* in virtual address space of that process.
2. Memblk objects DO NOT take up space in region B or region C.
3. If there are less number of processes in the system, less memory is used because each process's freelist is only initialized after creation and after the process calls vmhgetmem and vmhfreemem.
4. *Optimal storage strategy*: Block coalescing and blocks are stored in free space
5. O(n) retrieval where n is the number of *contiguous free list nodes*

Disadvantages and Testing caveats:

1. This implementation places list nodes in the virtual heap itself.

2. Therefore each list node **will occupy a frame in E1**

3. And, each list node is **accessible even after calling vmhfreemem() on that memory** location

4. Please do not test for these conditions as they are the caveats of maintaining free list nodes in the virtual heap itself

Code details:

The following fields are added to the process table:

```
bool8        prmeminit; /* Virtual heap initialized? */
struct memblk prmemblk;  /* Free list */
bool8        pralloc[MAXHSIZE]; /* Boolean set of page ownership */
```

prmeminit is a flag set to false, indicates whether free list must be initialized or not. Free list is initialized in vmhinit() which is called by vmhgetfree() and vmhfreemem() in the current process's virtual address space.

prmemblk is the region VF analogy to region C's memlist global variable.

pralloc[MAXHSIZE] is used for checking segmentation faults — accessing memory locations that are not allocated using vmhgetmem. Vmhgetmem is responsible for editing pralloc, while pgfhandler checks if the faulting page was allocated using this set of allocated addresses.

Please note that my OS may not de-allocate the VHeap page, since the page may be used as memblk.

Inverted Page-table and O(1) Frame Finding

Mapping data structure:

In my implementation, I designed a kernel data structure invpt[NFRAMES] that has the mapping E1 → (pid, state). That is, given a frame number, find out the state of the frame (FREE or USED), and the owner of the frame if any. This is for frame management and searching free frames in regions D, E1, E2.

System calls:

There are 3 system calls that manipulate the frames and inverted page table. allocaframe() and deallocaframe() are abstractions that manage the inverted page table. getfreeframe(region r) finds a free frame in the given region.

O(1) approach:

The issue is that getfreeframe() may traverse the entire region's frames to find the first free frame in O(n) time. However, we don't want that. getfreeframe() is used in pgfhandler, which is a lower-half kernel function. It must be O(1) in all aspects - allocation, de-allocation, and getting the first free frame in the region.

For this reason, I use **3 stacks for keeping track of indices of free frames** in regions D, E1, and E2. allocaframe and deallocaframe are the ones responsible for manipulating these stacks.

When a process wants a frame in a region, we would **pop the stack of that region in O(1) time and then allocate it in inverted page table in O(1) indexing time**. When the process is killed or a virtual heap memory page is freed,

dellocaframe() will **push it back onto the stack in O(1) time and then deallocate it in inverted page table in O(1) indexing time**.

This, I believe, is the most optimal and simple approach to frame management without brute-forcing through the inverted page table. You can replace stack with queue, but stack is used for simplicity's sake.

Testing

Test 1. Fill up a region using frame manager

The first test is to fill up a region by allocating every frame. After locking every frame to a process, we would check if asking for another frame returns SYSERR.

Output:

----- FRAME MGMT -----

@ Allocating 1024 pages

@ Requesting one more

@ Got -1

@ Deallocating frame 2050 and 3000

@ Requesting one

@ Got 3000

@ Requesting one more

@ Got 2050

Explanation:

1024 frames of E1 are marked as allocated. Then when requesting one more frame, we get SYSERR, meaning we cannot allocate any more pages. Now, we de-allocate frames 2050 and 3000 (they are added back on E1's stack). Requesting first new frame gives 3000 (FIFO stack). Requesting second new frame gives 2050 (FIFO stack)

Therefore frame allocation does not exceed NFRAMES of a given region due to stack implementation.

Test 2: Trying out vmhgetmem and accessing all pages in given page

The second test requests 1 page from vmhgetmem and tries to access that page. The access would trigger a page fault, thus a frame would be allocated. A debug function crawls the process page directory and a given address's page table to find out the page tables allocated.

Output:

```
----- TEST GET MEM 1 -----
@ Requesting one page
@ Accessing all locations of this page
@ x = aaaaaaaaaa
@ Printing debug information
 --- DEBUGGING PAGE TABLES ---
Address: 0x01000000
Page Directory Index: 4
Page Table Index: 0
PDE: 0x00408023
PTE: 0x00be7063
Frame: 3047
Page directory crawl:
   PDE 0 present : 0x00400021
   PDE 1 present : 0x00401021
   PDE 2 present : 0x00402001
   PDE 3 present : 0x00403001
   PDE 4 present : 0x00408023
            PDE for addr present!!
   PDE 576 present : 0x00404021
```

Page table crawl:

PTE 0 present : 0x00be7063

PTE for addr present!!

PTE 1 present : 0x00be6063

Explanation:

One page is requested from vmhgetmem. It returns the location 0x01000000. A for loop accesses NBPG (4096) locations of the page and sets the first 10 locations to ‘a’ and the rest to ‘\0’. This is proven by the 3rd print statement which shows the value of *x.

There is a helper function that debugs the address returned. This debugging information prints a lot of information. It finds out the page directory and table indexing from the virtual address. It then prints the page directory and page table entries in the process’s page multilevel page table structure. It finds the frame mapping for the given virtual address, which in this case is 3047. Now, it goes through the page directory and sees that PDE is present for page tables 0, 1, 2, 3 - corresponding to identity maps for regions A to E2 - and PTE is set for pages 0 and 1 - where 0 is the page for 0x01000000 and 1 is the page for the next free-list node due to my region VF implementation of free list.

Test 3: Getmem fill up and Freemem

The next test is a combination of getmem and freemem. This test fills up the entire virtual heap and tries to see if getmem will fail to return -1.

Output:

----- TEST FREE MEM 1 -----

@ Requesting 2 pages

@ Got memory 0x01000000 requesting 1

@ Requesting 1022 pages

@ Got memory 0x01002000 requesting 1022

@ Requesting more than 1024 page

@ Got -1

@ Freeing memory 01002000

@ Requesting 1 pages

@ Got memory 01002000 on requesting 1

@ Requesting 1 pages

@ Got memory 01003000 on requesting 1

Observation:

2 pages are requested. First free list node is returned.

1022 pages are requested. Second free list node is returned.

1 more page is requested. There is no more space in the linked list. Therefore -1 is returned.

All 1022 pages requested are freed. Technically, 0x01002000 becomes the next free list node.

Then 2 pages are requested consecutively. The linked list node 0x01002000 is returned, followed by the next location 0x01003000.

Test 4: Getmem, freemem, then access

This test checks for segmentation faults in the implementation. Getmem should mark a given page as allocated or unallocated. The following implementation does so.

Output:

----- TEST FREE MEM 2 -----

@ Requesting 2 pages

@ x = 69 y = 96

@ Freeing 2 pages then accessing

@ Accessing locations

@ x = 0

- Segmentation fault PID 5

Observation:

Two pages are requested, stored in x and y. They are set to 69 and 96 respectively.

Then both of them are freed. Technically, x should become the free list memblk in my implementation, hence it is still accessible.

Now, accessing x no longer gives 69 because it is being used as link list node as expected.

Accessing location y gives a segfault error message and kills the process, since y is no longer marked as allocated.

Please ignore the access to x, since it is being used as linked list node.

Test 5: Isolation of processes

Two processes are spawned in this implementation. The first one requests a memory location and sets it to a constant. The second one tries to access the memory location. This should result in a segmentation fault.

Output:

----- TEST TWO PROCESSES -----

@1 Requesting 1 location

@1 Setting 0x1000000 = 69

@1 Allowing process 2 to run

@2 Process unblocked, trying to access location

- Segmentation fault PID 8

Observation:

Process 1 requests a memory location and gets 0x01000000, setting it to 69. Process 1 then blocks and signals process 2 to run.

Process 2 tries to access the same location 0x01000000. However, the kernel data structure pralloc says the page 0x01000 for process 2 is unallocated. Therefore, page fault handler does not assign to a frame, and kills process 2 in this test. This is as expected, process 2 cannot access any frame owned by process 1.

Bonus

Modularity

Please refer README.1 in the submission folder to gauge the modular approach of my implementation. Everything is in its own file and each data structure is managed by its own function. This is kind of an object oriented approach, except only the encapsulation and abstraction aspect of OOP is used.

Performance

- Free list for vmhgetmem and vmhfreemem is implemented in space efficient, isolated manner as described above; with blocks coalescing (its literally the same implementation given to us in Xinu code for getmem and freemem)
- Frame management is implemented using constant time manner
- invlpg is used

Attention to detail

- There are SYSERR returns and checks in each modular function
- In vmhfreemem, **invlpg** is called to flush the TLB for all freed pages for correctness
- Debug logging is present everywhere in the code, for different modules