CSE422S Lab3 Report

The names of people working together on this studio:
Ziyang Jiao 475589 jiaoziyang@wustl.edu
Jing Zhang 475536 zjing@wustl.edu
Bingxin Liu 475532 bingxin.liu@wustl.edu

Initial Configuration

Step1:

```
pi@jiaoziyang:~/pj3/frame/lab-3 $ ls
include  Makefile  mod  user
pi@jiaoziyang:~/pj3/frame/lab-3 $ cd user/
pi@jiaoziyang:~/pj3/frame/lab-3/user $ ls
dense_mm  dense_mm.c  Makefile
pi@jiaoziyang:~/pj3/frame/lab-3/user $ sudo ./dense_mm 1
Bus error
```

```
[ 6909.287334] Loaded kmod_paging module
[ 6920.752164] paging_mmap() invoked: new VMA for pid 1928 from VA 0x76fa2000 to 0x76fa3000
[ 6920.752187] paging_mmap() invoked: new VMA for pid 1928 from VA 0x76fa1000 to 0x76fa2000
[ 6920.752203] paging_mmap() invoked: new VMA for pid 1928 from VA 0x76fa0000 to 0x76fa1000
[ 6920.752216] paging_vma_fault() invoked: took a page fault at VA 0x76fa0000
[ 6920.752499] paging_vma_close() invoked
[ 6920.752506] paging_vma_close() invoked
[ 6920.752512] paging_vma_close() invoked
```

Datatype Design

Step2:

We design the structure as follows including: 1) a counter filed, which is used to record the total number of reference related to this structure and when the field becomes zero, it means that we should free the memory; 2) a array of struct page pointer, which is used to record the allocated pages in `do_fault` or `mmap` function.

```
struct state{
  //The data structure must have an atomic reference counter (of type atomic_t) in it.
  atomic_t counter;
  //allocated pages
  struct page **pages;
};
```

In the open and close callback:

- when a new vma is created as a copy of an existing vma, the kernel invokes the open callback, and the open callback simply retrieve a pointer to our new data

structure via vma->vm_private_data and increment the reference count stored in the counter field of this struct.

● when a vma is being deleted, the kernel invokes the close callback, the close callback retrieve our data structure, decrement the reference count stored in the counter field of this struct, and if it becomes zero, free the dynamically allocated memory.

In the mmap callback:

- Firstly, We dynamically allocate memory for the struct and initialize the atomic field to one, which represents reference counter.
- Secondly, we assigns this structure to the private field of the given vma structure.

In the do_fault function:

- When we allocate a new page(physical memory), the function assigns the pointer of the new allocated page to the element of the pages* array stored in our customized struct.

Page Fault Handler

Step3:

The page fault handler(do_fault function) complete operations as follows:

● Firstly, it allocates a new page for the task using alloc_page() function

● Secondly, we record the pointer of the newly allocated page using our customized struct pointed by the vma->vm_private_data

● Thirdly, we use remap_pfn_range() to update the process' page tables to map the faulting virtual address to the new physical address we just allocated. The parameters of this function are calculated as follows:

```
/*
* remap_pfn_range(): Update the process' page tables to map the faulting virtual address to the new physical address you just allocated.
* @parameter1: user vma to map to: the vma pointer getting from do_fault input;
* @parameter2: target user address to start at: PAGE_ALIGN(fault_address);
* @parameter3: physical address of kernel memory: pfn = page_to_pfn(new_page);
* @parameter4: size of map area: PAGE_SIZE
* @parameter5: page protection flags for this mapping: vma->vm_page_prot
*/
```

Finally, the result of dense_mm program becomes normal, which is `Multiplication done`.

Close Callback

Step4:

Firstly, we add two atomic variables. One is allocate_page_counter with atomic_t struct which will automatically increase by 1 whenever we allocate a new struct page, and another is free_page_counter with the same struct, which will automatically decreased by 1 whenever we free a struct page.

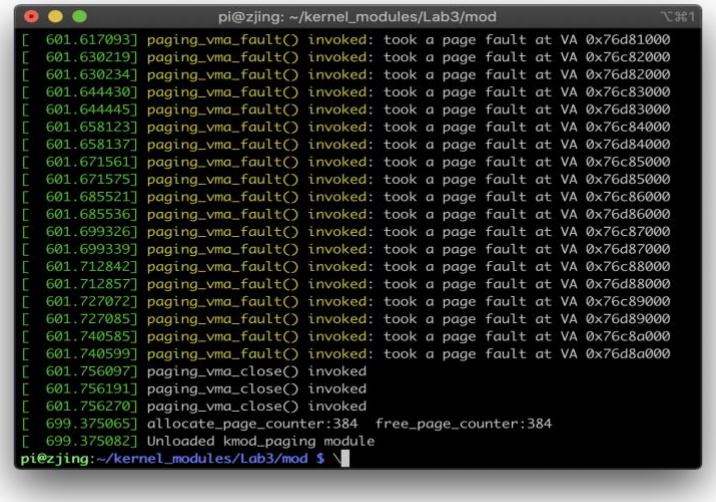
Then, within do_default function, we add a segment that whenever the program call do_default function, the allocate_page_counter will be incremented by 1 atomically. Since whenever we call do_default, it means that we hope to allocate new struct page.

Then, we implemented a function called “get_page_count” which will receive an vm_area_struct pointer, then return the number of pages we allocate for this kind of vm_area_struct, which will be used in paging_vma_close function.

Finally, we make paging_vma_close function which has a vm_area_struct pointer used to point to the vma that we allocated before. Within the paging_vma_close function, we first get the number of pages that we have allocated via get_pages_count function. Then we get the state data that we already stored in the variable named vm_private_data. After that, we got the pages’ counter, namely allocate_page_counter. Then, we decremented the counter stored in our state structure by 1, which means that the number of references for this vma will decrease by 1. If the counter is decreased to zero, which means no one uses this memory area, we can free it safely. So, we use __free_page() system call to free those pages. Note that we have gotten the number of pages we have allocated, so we could easily use a while loop to free them one by one. Then, whenever we free a page, we increase the variable of free_page_counter by 1. At the end, we use kfree() to free our tracker structure and any other dynamically allocated memory, which means free pages field under retrieved pointer which is established by us to record pages that we allocated, and the private data state itself.

At the end, we print the number of the allocate_page_counter variable and the free_page_counter variable.

The result is shown below:



Pre-paging

Step5:

In this section we finished the function of pre-paging memory. Firstly, we use a kernel module argument, which allow us to add a command line argument to indicate our module we want demand paging or we want pre-paging.

Then, instead of basically doing nothing, we add an if segment in our paging_mmap function according to the variable previous mentioned to judge which one the program should go. If we choose to do the pre-paging one, the program will then allocate all of them.

Within the “if” segment, we use a for loop to finish our function, that is to say, making repeated calls to alloc_page and remap_pfn_range to allocate one page at a time.

Then, in each allocating behavior, we first use alloc_page function to allocate a new page, which will return a page structure pointer pointing to the new page. After that, we make a judgment to see if we allocate successfully, i.e. if the area that the pointer point to is an empty page(pointer == null). If not, we will return an -ENOMEN to say allocation failure. Once the alloc_page succeed, we could use page_to_pfn function to get the page frame number, which will be used as an argument with other necessary arguments in remap_pfn_range function in the next step. Then, we check if we remap the virtual memory to physical memory successfully. If the return number from remap_pfn_range function is zero, then it is successfully remapped, and we use the pointer array within the tracer in step3 to trace this page. If the return value is not a zero, it means that remap failed, so we return an -EFAULT.

By far, we have successfully allocated one new page, so we increase the allocate_page_counter by 1 atomically to record the number that we have allocated.

At the end, we go back to the for loop to judge if have already allocate enough pages. If that is true, we will go out of for loop and finish our paging_mmap function. In this case, we already have allocated enough pages, the page fault signal will not be triggered by hardware. Then, we finish our allocating with pre-paging strategy.

Experiments

Step6:

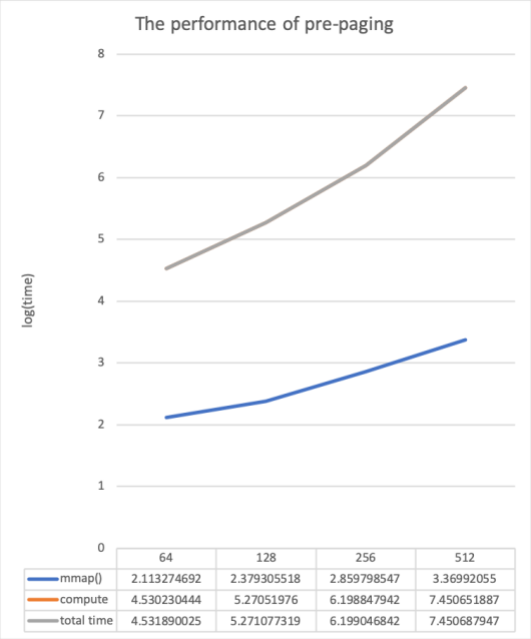
The minimum, maximum, mean, median, and standard deviation for the 10 runs in each experiment:

(unit: μs)

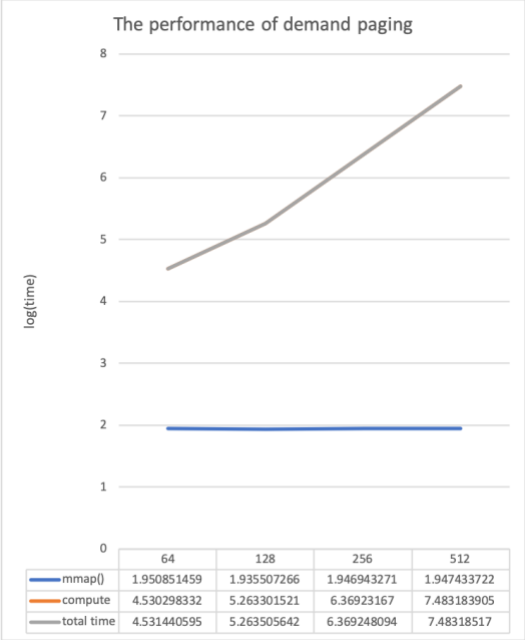
module mode	value	timing measurement	min	max	mean	median	SD
demand paging	64	mmap call	81.00	119.00	89.30	85.00	11.27
		matrix multiplication	32337.00	35330.00	33907.70	33987.00	1079.36
	128	mmap call	84.00	88.00	86.20	86.50	1.48
		matrix multiplication	165671.00	200425.00	183358.70	181557.00	10737.42
	256	mmap call	85.00	102.00	88.50	86.00	5.40
		matrix multiplication	1822194.00	2794298.00	2340085.20	2283865.50	309732.39
	512	mmap call	84.00	102.00	88.60	87.50	5.04
		matrix multiplication	29859068.00	31439573.00	30421729.80	30134167.50	615331.84
pre-paging	64	mmap call	122.00	140.00	129.80	130.00	5.63
		matrix multiplication	32558.00	35974.00	33902.40	33681.00	1001.97
	128	mmap call	231.00	248.00	239.50	239.00	4.99
		matrix multiplication	169167.00	207847.00	186431.70	177150.50	15189.03
	256	mmap call	641.00	1251.00	724.10	668.00	185.67
		matrix multiplication	1372617.00	1774667.00	1580694.50	1568775.50	138508.53
	512	mmap call	1254.00	2540.00	2343.80	2456.00	389.65
		matrix multiplication	28116476.00	28534918.00	28226165.70	28193658.50	125957.40

Names of the graph files

1)pre-paging.png



2)demand paging.png



Error bar:

	matrix size	64		128		256		512	
		mean	SD	mean	SD	mean	SD	mean	SD
demand paging	mmap()	89.3	11.27485	86.2	1.47573	88.5	5.400617	88.6	5.037636
	computation	33907.7	1079.361	183358.7	10737.42	2340085	309732.4	30421730	615331.8
	total	33997	1077.772	183444.9	10737.29	2340174	309730.2	30421818	615330.9
pre-paging	mmap()	129.8	5.633235	239.5	4.994441	724.1	185.6738	2343.8	389.6494
	computation	33902.4	1001.971	186431.7	15189.03	1580695	138508.5	28226166	125957.4
	total	34032.2	1001.385	186671.2	15185.74	1581419	138592	28228510	126037.8

From the table and line graphs above, we can see that demand paging outperform the pre-paging when executing calls to mmap. The mmap() time of using prepaging method is large because it needs to allocate pages as many as we request even if we do not use it now. According to our outputs, the system call takes longer time to complete prepaging because it does not just allocate one page as we did in demand paging mode. The time-consuming increases as the memory we request increase. However, when we use demand paging, the time consumed by mmap() seems to be consistent. That is because we just allocate virtual memory at that time. The physical memory is allocated in de_fault function when we try to write to it; otherwise, the page fault will not occur. Thus, the time used in mmap() is different with these two methods. When matrix size increases, the time consumed by matrix multiplication increases exponentially. Although prepaging method will use more time in mmap function, the overhead is smaller when it comes to the process of multiplication. In conclusion, if programs just need to access and write to a small amount of memory, then demand paging is a more optimal method because it only allocated the memory really be used; However, if our goal is to

get a better performance in computing and our program really need a large amount of memory, then we should use prepaging method, because the overhead and context switching will decrease a lot in this case.

Insights

Step7:

Because demand paging method will increase the overhead and context switching of our program, we believe that it is a good idea to test the balance point of between these two methods in order to make us choose a better allocation method in reality.

Time

Step8:

50hours

The output of dens_mm when demand_paging=0

```
pi@zjing:~/kernel_modules/Lab3/output $ /bin/bash run.sh
Multiplication done
Time consuming for mmap(microseconds):115
Time point of finishing computing (microseconds):34971
Multiplication done
Time consuming for mmap(microseconds):108
Time point of finishing computing (microseconds):34827
Multiplication done
Time consuming for mmap(microseconds):119
Time point of finishing computing (microseconds):33057
Multiplication done
Time consuming for mmap(microseconds):121
Time point of finishing computing (microseconds):32673
Multiplication done
Time consuming for mmap(microseconds):108
Time point of finishing computing (microseconds):34013
Multiplication done
Time consuming for mmap(microseconds):109
Time point of finishing computing (microseconds):32805
Multiplication done
Time consuming for mmap(microseconds):108
Time point of finishing computing (microseconds):33059
Multiplication done
Time consuming for mmap(microseconds):110
Time point of finishing computing (microseconds):35062
Multiplication done
Time consuming for mmap(microseconds):120
Time point of finishing computing (microseconds):32039
Multiplication done
Time consuming for mmap(microseconds):108
Time point of finishing computing (microseconds):33234
```

done 64
Multiplication done
Time consuming for mmap(microseconds):232
Time point of finishing computing (microseconds):194606
Multiplication done
Time consuming for mmap(microseconds):226
Time point of finishing computing (microseconds):166955
Multiplication done
Time consuming for mmap(microseconds):222
Time point of finishing computing (microseconds):169878
Multiplication done
Time consuming for mmap(microseconds):222
Time point of finishing computing (microseconds):177589
Multiplication done
Time consuming for mmap(microseconds):204
Time point of finishing computing (microseconds):200793
Multiplication done
Time consuming for mmap(microseconds):203
Time point of finishing computing (microseconds):186379
Multiplication done
Time consuming for mmap(microseconds):226
Time point of finishing computing (microseconds):169398
Multiplication done
Time consuming for mmap(microseconds):222
Time point of finishing computing (microseconds):194107
Multiplication done
Time consuming for mmap(microseconds):472
Time point of finishing computing (microseconds):187526
Multiplication done
Time consuming for mmap(microseconds):231
Time point of finishing computing (microseconds):170812
done 128
Multiplication done
Time consuming for mmap(microseconds):653
Time point of finishing computing (microseconds):1434350
Multiplication done
Time consuming for mmap(microseconds):636
Time point of finishing computing (microseconds):1467559
Multiplication done
Time consuming for mmap(microseconds):620
Time point of finishing computing (microseconds):1435225
Multiplication done
Time consuming for mmap(microseconds):631
Time point of finishing computing (microseconds):1537094
Multiplication done
Time consuming for mmap(microseconds):646
Time point of finishing computing (microseconds):1627257
Multiplication done
Time consuming for mmap(microseconds):631

Time point of finishing computing (microseconds):1607441
Multiplication done
Time consuming for mmap(microseconds):652
Time point of finishing computing (microseconds):1451553
Multiplication done
Time consuming for mmap(microseconds):644
Time point of finishing computing (microseconds):1461573
Multiplication done
Time consuming for mmap(microseconds):642
Time point of finishing computing (microseconds):1339714
Multiplication done
Time consuming for mmap(microseconds):640
Time point of finishing computing (microseconds):1437324
done 256
Multiplication done
Time consuming for mmap(microseconds):2350
Time point of finishing computing (microseconds):25034968
Multiplication done
Time consuming for mmap(microseconds):2326
Time point of finishing computing (microseconds):25001156
Multiplication done
Time consuming for mmap(microseconds):2319
Time point of finishing computing (microseconds):24998842
Multiplication done
Time consuming for mmap(microseconds):2324
Time point of finishing computing (microseconds):25012513
Multiplication done
Time consuming for mmap(microseconds):2456
Time point of finishing computing (microseconds):25113347
Multiplication done
Time consuming for mmap(microseconds):2328
Time point of finishing computing (microseconds):25023771
Multiplication done
Time consuming for mmap(microseconds):2322
Time point of finishing computing (microseconds):24992040
Multiplication done
Time consuming for mmap(microseconds):2331
Time point of finishing computing (microseconds):25013943
Multiplication done
Time consuming for mmap(microseconds):2321
Time point of finishing computing (microseconds):24998014
Multiplication done
Time consuming for mmap(microseconds):2325
Time point of finishing computing (microseconds):25015183
done 512

Dmesg output when demand_paging=0
[9068.621885] allocate_page_counter:20400 free_page_counter:20400
[9068.621893] Unloaded kmod_paging module

```
[ 9100.779831] Loaded kmod_paging module
[ 9120.304120] paging_mmap() invoked: new VMA for pid 3014 from VA 0x76f98000 to 0x76fa0000
[ 9120.304156] paging_mmap() invoked: new VMA for pid 3014 from VA 0x76f57000 to 0x76f5f000
[ 9120.304191] paging_mmap() invoked: new VMA for pid 3014 from VA 0x76f4f000 to 0x76f57000
[ 9120.340151] paging_vma_close() invoked
[ 9120.340197] paging_vma_close() invoked
[ 9120.340213] paging_vma_close() invoked
[ 9121.417745] paging_mmap() invoked: new VMA for pid 3024 from VA 0x76fcb000 to 0x76fd3000
[ 9121.417781] paging_mmap() invoked: new VMA for pid 3024 from VA 0x76f8a000 to 0x76f92000
[ 9121.417808] paging_mmap() invoked: new VMA for pid 3024 from VA 0x76f82000 to 0x76f8a000
[ 9121.454634] paging_vma_close() invoked
[ 9121.454652] paging_vma_close() invoked
[ 9121.454667] paging_vma_close() invoked
[ 9122.531430] paging_mmap() invoked: new VMA for pid 3034 from VA 0x76f7a000 to 0x76f82000
[ 9122.531466] paging_mmap() invoked: new VMA for pid 3034 from VA 0x76f39000 to 0x76f41000
[ 9122.531505] paging_mmap() invoked: new VMA for pid 3034 from VA 0x76f31000 to 0x76f39000
[ 9122.566589] paging_vma_close() invoked
[ 9122.566610] paging_vma_close() invoked
[ 9122.566629] paging_vma_close() invoked
[ 9123.641096] paging_mmap() invoked: new VMA for pid 3044 from VA 0x76fe2000 to 0x76fea000
[ 9123.641135] paging_mmap() invoked: new VMA for pid 3044 from VA 0x76fa1000 to 0x76fa9000
[ 9123.641173] paging_mmap() invoked: new VMA for pid 3044 from VA 0x76f99000 to 0x76fa1000
[ 9123.675835] paging_vma_close() invoked
[ 9123.675879] paging_vma_close() invoked
[ 9123.675899] paging_vma_close() invoked
[ 9124.752433] paging_mmap() invoked: new VMA for pid 3054 from VA 0x76f6a000 to 0x76f72000
[ 9124.752470] paging_mmap() invoked: new VMA for pid 3054 from VA 0x76f29000 to 0x76f31000
[ 9124.752497] paging_mmap() invoked: new VMA for pid 3054 from VA 0x76f21000 to 0x76f29000
[ 9124.788386] paging_vma_close() invoked
[ 9124.788406] paging_vma_close() invoked
[ 9124.788421] paging_vma_close() invoked
[ 9125.866199] paging_mmap() invoked: new VMA for pid 3064 from VA 0x76f86000 to 0x76f8e000
[ 9125.866236] paging_mmap() invoked: new VMA for pid 3064 from VA 0x76f45000 to 0x76f4d000
[ 9125.866263] paging_mmap() invoked: new VMA for pid 3064 from VA 0x76f3d000 to 0x76f45000
[ 9125.900019] paging_vma_close() invoked
[ 9125.900038] paging_vma_close() invoked
[ 9125.900053] paging_vma_close() invoked
[ 9126.981577] paging_mmap() invoked: new VMA for pid 3074 from VA 0x76f23000 to 0x76f2b000
[ 9126.981613] paging_mmap() invoked: new VMA for pid 3074 from VA 0x76ee2000 to 0x76eea000
[ 9126.981641] paging_mmap() invoked: new VMA for pid 3074 from VA 0x76eda000 to 0x76ee2000
[ 9127.015606] paging_vma_close() invoked
[ 9127.015626] paging_vma_close() invoked
[ 9127.015642] paging_vma_close() invoked
[ 9128.093436] paging_mmap() invoked: new VMA for pid 3084 from VA 0x76f72000 to 0x76f7a000
[ 9128.093472] paging_mmap() invoked: new VMA for pid 3084 from VA 0x76f31000 to 0x76f39000
[ 9128.093501] paging_mmap() invoked: new VMA for pid 3084 from VA 0x76f29000 to 0x76f31000
[ 9128.129612] paging_vma_close() invoked
[ 9128.129633] paging_vma_close() invoked
[ 9128.129648] paging_vma_close() invoked
```

[9129.206462] paging_mmap() invoked: new VMA for pid 3094 from VA 0x76fed000 to 0x76ff5000
[9129.206499] paging_mmap() invoked: new VMA for pid 3094 from VA 0x76fac000 to 0x76fb4000
[9129.206525] paging_mmap() invoked: new VMA for pid 3094 from VA 0x76fa4000 to 0x76fac000
[9129.239508] paging_vma_close() invoked
[9129.239527] paging_vma_close() invoked
[9129.239568] paging_vma_close() invoked
[9130.318025] paging_mmap() invoked: new VMA for pid 3104 from VA 0x76f3c000 to 0x76f44000
[9130.318060] paging_mmap() invoked: new VMA for pid 3104 from VA 0x76efb000 to 0x76f03000
[9130.318086] paging_mmap() invoked: new VMA for pid 3104 from VA 0x76ef3000 to 0x76efb000
[9130.352557] paging_vma_close() invoked
[9130.352577] paging_vma_close() invoked
[9130.352592] paging_vma_close() invoked
[9131.435725] paging_mmap() invoked: new VMA for pid 3114 from VA 0x76d9d000 to 0x76dbd000
[9131.435806] paging_mmap() invoked: new VMA for pid 3114 from VA 0x76d7d000 to 0x76d9d000
[9131.435873] paging_mmap() invoked: new VMA for pid 3114 from VA 0x76d5d000 to 0x76d7d000
[9131.631149] paging_vma_close() invoked
[9131.631183] paging_vma_close() invoked
[9131.631213] paging_vma_close() invoked
[9132.698692] paging_mmap() invoked: new VMA for pid 3124 from VA 0x76d46000 to 0x76d66000
[9132.698769] paging_mmap() invoked: new VMA for pid 3124 from VA 0x76d26000 to 0x76d46000
[9132.698834] paging_mmap() invoked: new VMA for pid 3124 from VA 0x76d06000 to 0x76d26000
[9132.866839] paging_vma_close() invoked
[9132.866861] paging_vma_close() invoked
[9132.866893] paging_vma_close() invoked
[9133.934521] paging_mmap() invoked: new VMA for pid 3134 from VA 0x76db7000 to 0x76dd7000
[9133.934589] paging_mmap() invoked: new VMA for pid 3134 from VA 0x76d97000 to 0x76db7000
[9133.934659] paging_mmap() invoked: new VMA for pid 3134 from VA 0x76d77000 to 0x76d97000
[9134.105183] paging_vma_close() invoked
[9134.105221] paging_vma_close() invoked
[9134.105252] paging_vma_close() invoked
[9135.171787] paging_mmap() invoked: new VMA for pid 3144 from VA 0x76d51000 to 0x76d71000
[9135.171857] paging_mmap() invoked: new VMA for pid 3144 from VA 0x76d31000 to 0x76d51000
[9135.171926] paging_mmap() invoked: new VMA for pid 3144 from VA 0x76d11000 to 0x76d31000
[9135.350607] paging_vma_close() invoked
[9135.350629] paging_vma_close() invoked
[9135.350646] paging_vma_close() invoked
[9136.421417] paging_mmap() invoked: new VMA for pid 3154 from VA 0x76d99000 to 0x76db9000
[9136.421485] paging_mmap() invoked: new VMA for pid 3154 from VA 0x76d79000 to 0x76d99000
[9136.421544] paging_mmap() invoked: new VMA for pid 3154 from VA 0x76d59000 to 0x76d79000
[9136.623402] paging_vma_close() invoked
[9136.623424] paging_vma_close() invoked
[9136.623441] paging_vma_close() invoked
[9137.690542] paging_mmap() invoked: new VMA for pid 3164 from VA 0x76d51000 to 0x76d71000
[9137.690611] paging_mmap() invoked: new VMA for pid 3164 from VA 0x76d31000 to 0x76d51000
[9137.690669] paging_mmap() invoked: new VMA for pid 3164 from VA 0x76d11000 to 0x76d31000
[9137.877804] paging_vma_close() invoked
[9137.877826] paging_vma_close() invoked
[9137.877843] paging_vma_close() invoked
[9138.943896] paging_mmap() invoked: new VMA for pid 3174 from VA 0x76d53000 to 0x76d73000

[9138.943975] paging_mmap() invoked: new VMA for pid 3174 from VA 0x76d33000 to 0x76d53000
[9138.944039] paging_mmap() invoked: new VMA for pid 3174 from VA 0x76d13000 to 0x76d33000
[9139.114331] paging_vma_close() invoked
[9139.114364] paging_vma_close() invoked
[9139.114392] paging_vma_close() invoked
[9140.185127] paging_mmap() invoked: new VMA for pid 3184 from VA 0x76e12000 to 0x76e32000
[9140.185197] paging_mmap() invoked: new VMA for pid 3184 from VA 0x76df2000 to 0x76e12000
[9140.185268] paging_mmap() invoked: new VMA for pid 3184 from VA 0x76dd2000 to 0x76df2000
[9140.380248] paging_vma_close() invoked
[9140.380284] paging_vma_close() invoked
[9140.380314] paging_vma_close() invoked
[9141.447234] paging_mmap() invoked: new VMA for pid 3194 from VA 0x76ddb000 to 0x76dfb000
[9141.447303] paging_mmap() invoked: new VMA for pid 3194 from VA 0x76dbb000 to 0x76ddb000
[9141.447619] paging_mmap() invoked: new VMA for pid 3194 from VA 0x76d9b000 to 0x76dbb000
[9141.636055] paging_vma_close() invoked
[9141.636091] paging_vma_close() invoked
[9141.636121] paging_vma_close() invoked
[9142.702367] paging_mmap() invoked: new VMA for pid 3204 from VA 0x76df9000 to 0x76e19000
[9142.702450] paging_mmap() invoked: new VMA for pid 3204 from VA 0x76dd9000 to 0x76df9000
[9142.702516] paging_mmap() invoked: new VMA for pid 3204 from VA 0x76db9000 to 0x76dd9000
[9142.873970] paging_vma_close() invoked
[9142.874006] paging_vma_close() invoked
[9142.874036] paging_vma_close() invoked
[9143.940429] paging_mmap() invoked: new VMA for pid 3214 from VA 0x76d49000 to 0x76dc9000
[9143.940649] paging_mmap() invoked: new VMA for pid 3214 from VA 0x76cc9000 to 0x76d49000
[9143.940852] paging_mmap() invoked: new VMA for pid 3214 from VA 0x76c49000 to 0x76cc9000
[9145.376090] paging_vma_close() invoked
[9145.376187] paging_vma_close() invoked
[9145.376283] paging_vma_close() invoked
[9146.447556] paging_mmap() invoked: new VMA for pid 3224 from VA 0x76dcf000 to 0x76e4f000
[9146.447768] paging_mmap() invoked: new VMA for pid 3224 from VA 0x76d4f000 to 0x76dcf000
[9146.447969] paging_mmap() invoked: new VMA for pid 3224 from VA 0x76ccf000 to 0x76d4f000
[9147.916390] paging_vma_close() invoked
[9147.916484] paging_vma_close() invoked
[9147.916573] paging_vma_close() invoked
[9148.987738] paging_mmap() invoked: new VMA for pid 3235 from VA 0x76d36000 to 0x76db6000
[9148.987931] paging_mmap() invoked: new VMA for pid 3235 from VA 0x76cb6000 to 0x76d36000
[9148.988137] paging_mmap() invoked: new VMA for pid 3235 from VA 0x76c36000 to 0x76cb6000
[9150.424310] paging_vma_close() invoked
[9150.424412] paging_vma_close() invoked
[9150.424502] paging_vma_close() invoked
[9151.494717] paging_mmap() invoked: new VMA for pid 3245 from VA 0x76da3000 to 0x76e23000
[9151.494914] paging_mmap() invoked: new VMA for pid 3245 from VA 0x76d23000 to 0x76da3000
[9151.495128] paging_mmap() invoked: new VMA for pid 3245 from VA 0x76ca3000 to 0x76d23000
[9153.033119] paging_vma_close() invoked
[9153.033219] paging_vma_close() invoked
[9153.033310] paging_vma_close() invoked
[9154.101916] paging_mmap() invoked: new VMA for pid 3255 from VA 0x76d89000 to 0x76e09000
[9154.102137] paging_mmap() invoked: new VMA for pid 3255 from VA 0x76d09000 to 0x76d89000

[9154.102341] paging_mmap() invoked: new VMA for pid 3255 from VA 0x76c89000 to 0x76d09000
[9155.730632] paging_vma_close() invoked
[9155.730731] paging_vma_close() invoked
[9155.730832] paging_vma_close() invoked
[9156.802297] paging_mmap() invoked: new VMA for pid 3265 from VA 0x76d34000 to 0x76db4000
[9156.802493] paging_mmap() invoked: new VMA for pid 3265 from VA 0x76cb4000 to 0x76d34000
[9156.802699] paging_mmap() invoked: new VMA for pid 3265 from VA 0x76c34000 to 0x76cb4000
[9158.411045] paging_vma_close() invoked
[9158.411139] paging_vma_close() invoked
[9158.411231] paging_vma_close() invoked
[9159.482878] paging_mmap() invoked: new VMA for pid 3275 from VA 0x76d40000 to 0x76dc0000
[9159.483097] paging_mmap() invoked: new VMA for pid 3275 from VA 0x76cc0000 to 0x76d40000
[9159.483305] paging_mmap() invoked: new VMA for pid 3275 from VA 0x76c40000 to 0x76cc0000
[9160.935753] paging_vma_close() invoked
[9160.935846] paging_vma_close() invoked
[9160.935939] paging_vma_close() invoked
[9162.008242] paging_mmap() invoked: new VMA for pid 3285 from VA 0x76dbe000 to 0x76e3e000
[9162.008451] paging_mmap() invoked: new VMA for pid 3285 from VA 0x76d3e000 to 0x76dbe000
[9162.008662] paging_mmap() invoked: new VMA for pid 3285 from VA 0x76cbe000 to 0x76d3e000
[9163.471130] paging_vma_close() invoked
[9163.471225] paging_vma_close() invoked
[9163.471324] paging_vma_close() invoked
[9164.542707] paging_mmap() invoked: new VMA for pid 3295 from VA 0x76d57000 to 0x76dd7000
[9164.542924] paging_mmap() invoked: new VMA for pid 3295 from VA 0x76cd7000 to 0x76d57000
[9164.543128] paging_mmap() invoked: new VMA for pid 3295 from VA 0x76c57000 to 0x76cd7000
[9165.883739] paging_vma_close() invoked
[9165.883835] paging_vma_close() invoked
[9165.883930] paging_vma_close() invoked
[9166.954017] paging_mmap() invoked: new VMA for pid 3305 from VA 0x76cf0000 to 0x76d70000
[9166.954221] paging_mmap() invoked: new VMA for pid 3305 from VA 0x76c70000 to 0x76cf0000
[9166.954427] paging_mmap() invoked: new VMA for pid 3305 from VA 0x76bf0000 to 0x76c70000
[9168.392591] paging_vma_close() invoked
[9168.392686] paging_vma_close() invoked
[9168.392778] paging_vma_close() invoked
[9169.461750] paging_mmap() invoked: new VMA for pid 3315 from VA 0x76c35000 to 0x76e35000
[9169.462527] paging_mmap() invoked: new VMA for pid 3315 from VA 0x76a35000 to 0x76c35000
[9169.463311] paging_mmap() invoked: new VMA for pid 3315 from VA 0x76835000 to 0x76a35000
[9194.500339] paging_vma_close() invoked
[9194.500644] paging_vma_close() invoked
[9194.500961] paging_vma_close() invoked
[9195.568833] paging_mmap() invoked: new VMA for pid 3325 from VA 0x76c60000 to 0x76e60000
[9195.569612] paging_mmap() invoked: new VMA for pid 3325 from VA 0x76a60000 to 0x76c60000
[9195.570375] paging_mmap() invoked: new VMA for pid 3325 from VA 0x76860000 to 0x76a60000
[9220.573360] paging_vma_close() invoked
[9220.573653] paging_vma_close() invoked
[9220.573987] paging_vma_close() invoked
[9221.640551] paging_mmap() invoked: new VMA for pid 3335 from VA 0x76bca000 to 0x76dca000
[9221.641319] paging_mmap() invoked: new VMA for pid 3335 from VA 0x769ca000 to 0x76bca000
[9221.642082] paging_mmap() invoked: new VMA for pid 3335 from VA 0x767ca000 to 0x769ca000

[9246.642765] paging_vma_close() invoked
[9246.643073] paging_vma_close() invoked
[9246.643395] paging_vma_close() invoked
[9247.711042] paging_mmap() invoked: new VMA for pid 3345 from VA 0x76bff000 to 0x76dff000
[9247.711813] paging_mmap() invoked: new VMA for pid 3345 from VA 0x769ff000 to 0x76bff000
[9247.712584] paging_mmap() invoked: new VMA for pid 3345 from VA 0x767ff000 to 0x769ff000
[9272.727055] paging_vma_close() invoked
[9272.727350] paging_vma_close() invoked
[9272.727684] paging_vma_close() invoked
[9273.797021] paging_mmap() invoked: new VMA for pid 3355 from VA 0x76b8f000 to 0x76d8f000
[9273.797813] paging_mmap() invoked: new VMA for pid 3355 from VA 0x7698f000 to 0x76b8f000
[9273.798671] paging_mmap() invoked: new VMA for pid 3355 from VA 0x7678f000 to 0x7698f000
[9298.913933] paging_vma_close() invoked
[9298.914230] paging_vma_close() invoked
[9298.914573] paging_vma_close() invoked
[9299.984930] paging_mmap() invoked: new VMA for pid 3366 from VA 0x76c50000 to 0x76e50000
[9299.985708] paging_mmap() invoked: new VMA for pid 3366 from VA 0x76a50000 to 0x76c50000
[9299.986472] paging_mmap() invoked: new VMA for pid 3366 from VA 0x76850000 to 0x76a50000
[9325.012065] paging_vma_close() invoked
[9325.012360] paging_vma_close() invoked
[9325.012699] paging_vma_close() invoked
[9326.080941] paging_mmap() invoked: new VMA for pid 3376 from VA 0x76c14000 to 0x76e14000
[9326.081713] paging_mmap() invoked: new VMA for pid 3376 from VA 0x76a14000 to 0x76c14000
[9326.082483] paging_mmap() invoked: new VMA for pid 3376 from VA 0x76814000 to 0x76a14000
[9351.076349] paging_vma_close() invoked
[9351.076646] paging_vma_close() invoked
[9351.076985] paging_vma_close() invoked
[9352.146096] paging_mmap() invoked: new VMA for pid 3386 from VA 0x76c16000 to 0x76e16000
[9352.146876] paging_mmap() invoked: new VMA for pid 3386 from VA 0x76a16000 to 0x76c16000
[9352.147641] paging_mmap() invoked: new VMA for pid 3386 from VA 0x76816000 to 0x76a16000
[9377.163422] paging_vma_close() invoked
[9377.163725] paging_vma_close() invoked
[9377.164060] paging_vma_close() invoked
[9378.233567] paging_mmap() invoked: new VMA for pid 3396 from VA 0x76c08000 to 0x76e08000
[9378.234340] paging_mmap() invoked: new VMA for pid 3396 from VA 0x76a08000 to 0x76c08000
[9378.235102] paging_mmap() invoked: new VMA for pid 3396 from VA 0x76808000 to 0x76a08000
[9403.234947] paging_vma_close() invoked
[9403.235253] paging_vma_close() invoked
[9403.235580] paging_vma_close() invoked
[9404.303718] paging_mmap() invoked: new VMA for pid 3406 from VA 0x76c2e000 to 0x76e2e000
[9404.304493] paging_mmap() invoked: new VMA for pid 3406 from VA 0x76a2e000 to 0x76c2e000
[9404.305263] paging_mmap() invoked: new VMA for pid 3406 from VA 0x7682e000 to 0x76a2e000
[9429.322305] paging_vma_close() invoked
[9429.322610] paging_vma_close() invoked
[9429.322942] paging_vma_close() invoked