CS 251 / EE 255 REAL-TIME EMBEDDED SYSTEMS

HW Project #3: Task Allocation and Virtual Memory Management

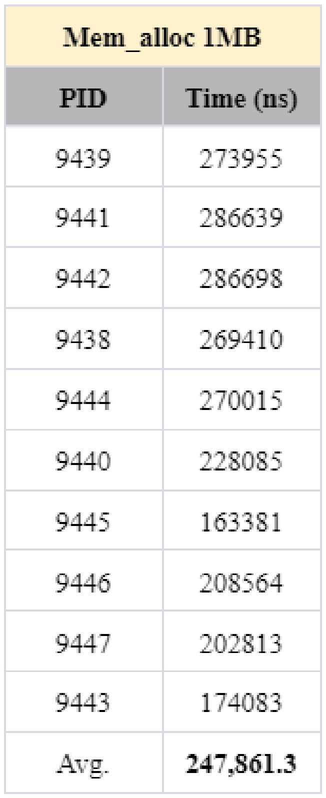
Group16

Group members:

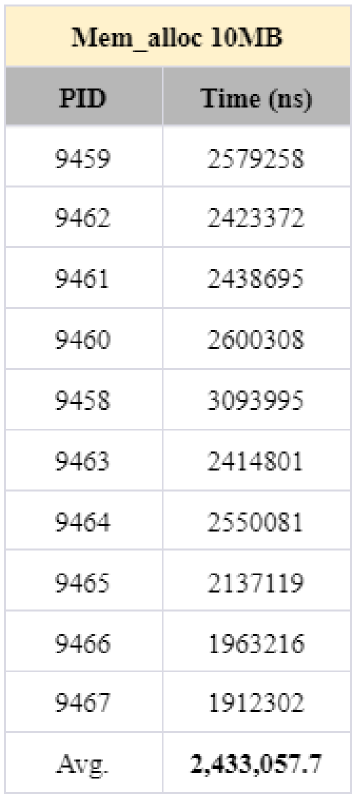
Hengshuo Zhang	Zhaoze Sun
hzhan402@ucr.edu	zsun114@ucr.edu

Writeup (Bonus Implemented 4.3.3)

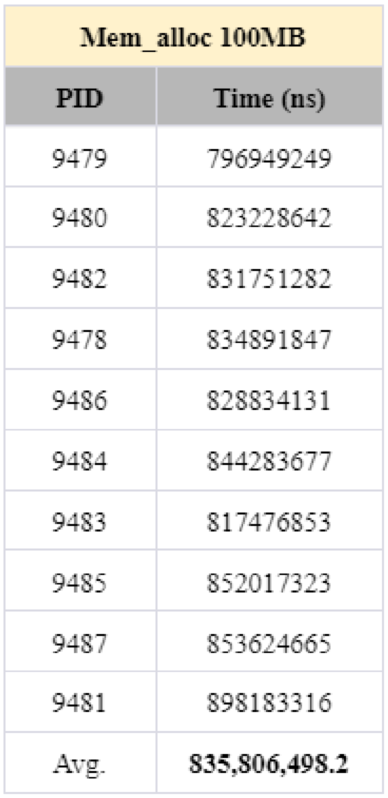
1. Report the memory access time of mem_alloc and mem_alloc_lock (assignment 4.2.1 & 4.2.2) with the memory size of 1 MB, 10 MB, and 100MB. For each case, run at least 10 times and provide average memory access. Give a brief discussion comparing the results.



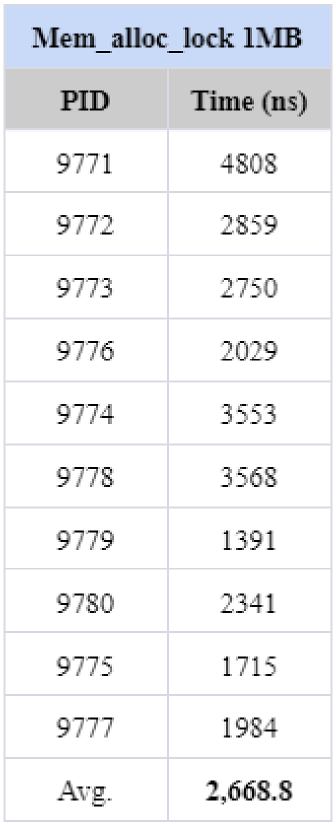
Mem_alloc 1MB	
PID	Time (ns)
9439	273955
9441	286639
9442	286698
9438	269410
9444	270015
9440	228085
9445	163381
9446	208564
9447	202813
9443	174083
Avg.	**247,861.3**



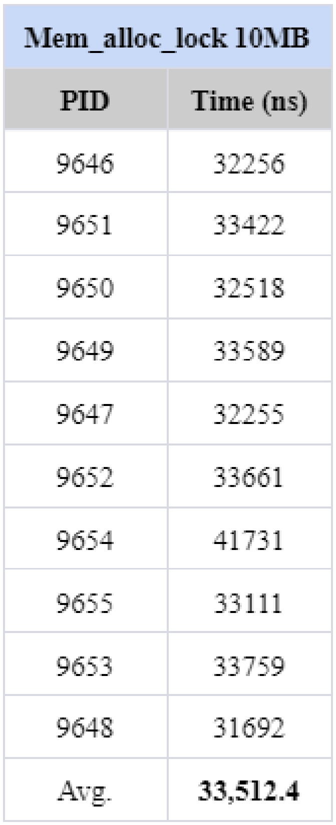
Mem_alloc 10MB	
PID	Time (ns)
9459	2579258
9462	2423372
9461	2438695
9460	2600308
9458	3093995
9463	2414801
9464	2550081
9465	2137119
9466	1963216
9467	1912302
Avg.	**2,433,057.7**



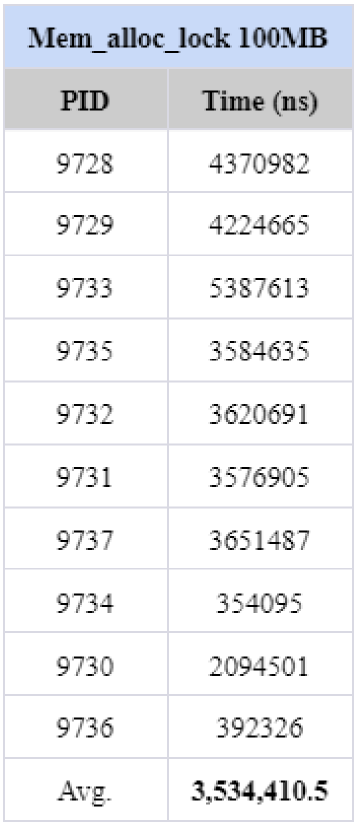
Mem_alloc 100MB	
PID	Time (ns)
9479	796949249
9480	823228642
9482	831751282
9478	834891847
9486	828834131
9484	844283677
9483	817476853
9485	852017323
9487	853624665
9481	898183316
Avg.	**835,806,498.2**



Mem_alloc_lock 1MB	
PID	Time (ns)
9771	4808
9772	2859
9773	2750
9776	2029
9774	3553
9778	3568
9779	1391
9780	2341
9775	1715
9777	1984
Avg.	**2,668.8**



Mem_alloc_lock 10MB	
PID	Time (ns)
9646	32256
9651	33422
9650	32518
9649	33589
9647	32255
9652	33661
9654	41731
9655	33111
9653	33759
9648	31692
Avg.	**33,512.4**



Mem_alloc_lock 100MB	
PID	Time (ns)
9728	4370982
9729	4224665
9733	5387613
9735	3584635
9732	3620691
9731	3576905
9737	3651487
9734	354095
9730	2094501
9736	392326
Avg.	**3,534,410.5**

By comparing the three memory size tests, we can clearly observe that the difference in memory access time between ***mem_alloc*** and ***mem_alloc_lock*** becomes more pronounced as the memory size increases.

Memory Size (bytes)	mem_alloc (ns)	mem_alloc_lock (ns)	Performance Improvement
1,000,000	247,861.3	2,668.8	92.81x
10,000,000	2,433,057.7	33,512.4	72.65x
100,000,000	835,806,498.2	3,534,410.5	236.46x

By comparing the results, ***mem_alloc_lock*** has better performance and also has better memory access performance for large memory handling. So for some memory access programs with larger requirements, we will choose ***mem_alloc_lock*** to improve the efficiency.

Based on the average memory access times reported by the program, we can see that ***mem_alloc_lock*** typically has shorter memory access times compared to ***mem_alloc*** for all the memory sizes tested. This is expected because ***mem_alloc_lock*** uses *mlock*, which reduces a lot of page faults and paging operations; both of which may cause memory access to take longer.

While memory locking brings significant performance, it also has some drawbacks. First of all, we need higher privileges to run the program (sudo), which does not necessarily apply to all software. Also memory locking will consume a lot of physical memory, which may affect system performance.

2. Report the kernel logs of /dev/segment_info (assignment 4.3.1) for mem_alloc_lock (assignment 4.2.2) with the memory size of 1 KB and 100MB.
3. Report the kernel logs of /dev/vm_areas (assignment 4.3.2) for mem_alloc_lock (assignment 4.2.2) with the memory size of 1 KB and 100MB.
Note: To make it easier to see the results of the data more intuitively, I will use mem_alloc_lock once to get a pid and then test both questions 2 and 3. For question 3 implements 4.3.3

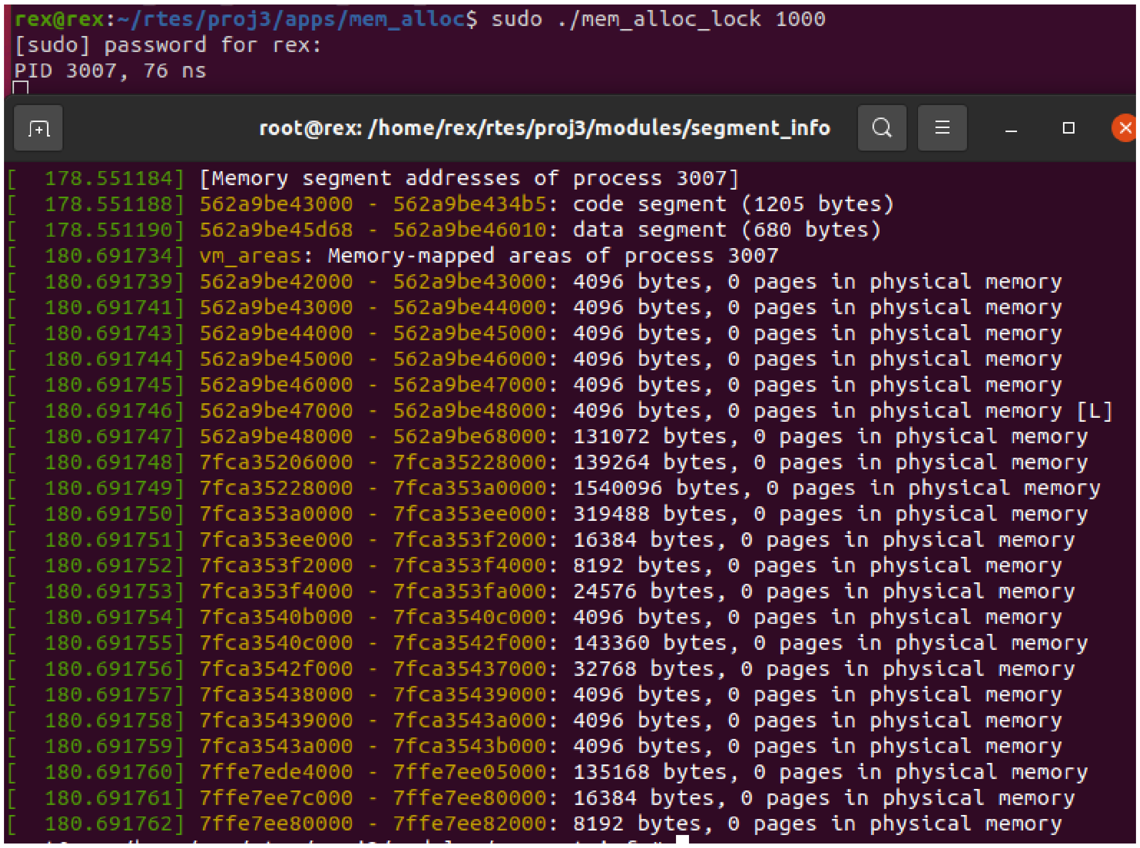
Code implementation steps:

```
> sudo ./mem_alloc_lock 1000       // 1KB
   PID 3007, 76 ns
> echo 3007 > /dev/segment_info
> echo 3007 > /dev/vm_areas

> sudo ./mem_alloc_lock 100000000  // 100MB
   PID 3037, 280562 ns
> echo 3037 > /dev/segment_info
> echo 3037 > /dev/vm_areas
```

Here is a screenshot after the test:

Case 1KB:

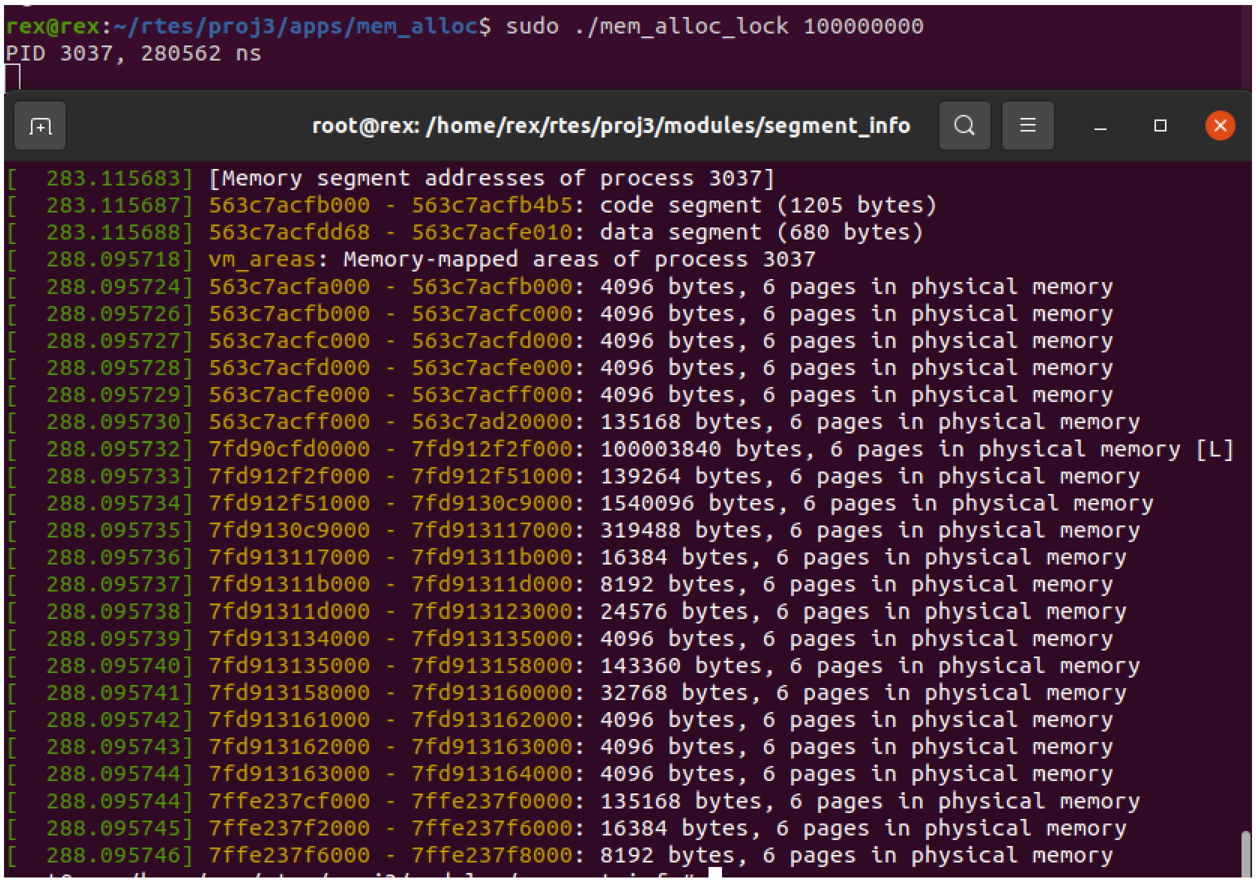


```
rex@rex:~/rtes/proj3/apps/mem_alloc$ sudo ./mem_alloc_lock 1000
[sudo] password for rex:
PID 3007, 76 ns
```

root@rex: /home/rex/rtes/proj3/modules/segment_info

```
[  178.551184] [Memory segment addresses of process 3007]
[  178.551188] 562a9be43000 - 562a9be434b5: code segment (1205 bytes)
[  178.551190] 562a9be45d68 - 562a9be46010: data segment (680 bytes)
[  180.691734] vm_areas: Memory-mapped areas of process 3007
[  180.691739] 562a9be42000 - 562a9be43000: 4096 bytes, 0 pages in physical memory
[  180.691741] 562a9be43000 - 562a9be44000: 4096 bytes, 0 pages in physical memory
[  180.691743] 562a9be44000 - 562a9be45000: 4096 bytes, 0 pages in physical memory
[  180.691744] 562a9be45000 - 562a9be46000: 4096 bytes, 0 pages in physical memory
[  180.691745] 562a9be46000 - 562a9be47000: 4096 bytes, 0 pages in physical memory
[  180.691746] 562a9be47000 - 562a9be48000: 4096 bytes, 0 pages in physical memory [L]
[  180.691747] 562a9be48000 - 562a9be68000: 131072 bytes, 0 pages in physical memory
[  180.691748] 7fca35206000 - 7fca35228000: 139264 bytes, 0 pages in physical memory
[  180.691749] 7fca35228000 - 7fca353a0000: 1540096 bytes, 0 pages in physical memory
[  180.691750] 7fca353a0000 - 7fca353ee000: 319488 bytes, 0 pages in physical memory
[  180.691751] 7fca353ee000 - 7fca353f2000: 16384 bytes, 0 pages in physical memory
[  180.691752] 7fca353f2000 - 7fca353f4000: 8192 bytes, 0 pages in physical memory
[  180.691753] 7fca353f4000 - 7fca353fa000: 24576 bytes, 0 pages in physical memory
[  180.691754] 7fca3540b000 - 7fca3540c000: 4096 bytes, 0 pages in physical memory
[  180.691755] 7fca3540c000 - 7fca3542f000: 143360 bytes, 0 pages in physical memory
[  180.691756] 7fca3542f000 - 7fca35437000: 32768 bytes, 0 pages in physical memory
[  180.691757] 7fca35438000 - 7fca35439000: 4096 bytes, 0 pages in physical memory
[  180.691758] 7fca35439000 - 7fca3543a000: 4096 bytes, 0 pages in physical memory
[  180.691759] 7fca3543a000 - 7fca3543b000: 4096 bytes, 0 pages in physical memory
[  180.691760] 7ffe7ede4000 - 7ffe7ee05000: 135168 bytes, 0 pages in physical memory
[  180.691761] 7ffe7ee7c000 - 7ffe7ee80000: 16384 bytes, 0 pages in physical memory
[  180.691762] 7ffe7ee80000 - 7ffe7ee82000: 8192 bytes, 0 pages in physical memory
```

Case 100 MB:



```
rex@rex:~/rtes/proj3/apps/mem_alloc$ sudo ./mem_alloc_lock 100000000
PID 3037, 280562 ns
```

```
[  283.115683] [Memory segment addresses of process 3037]
[  283.115687] 563c7acfb000 - 563c7acfb4b5: code segment (1205 bytes)
[  283.115688] 563c7acfdd68 - 563c7acfe010: data segment (680 bytes)
[  288.095718] vm_areas: Memory-mapped areas of process 3037
[  288.095724] 563c7acfa000 - 563c7acfb000: 4096 bytes, 6 pages in physical memory
[  288.095726] 563c7acfb000 - 563c7acfc000: 4096 bytes, 6 pages in physical memory
[  288.095727] 563c7acfc000 - 563c7acfd000: 4096 bytes, 6 pages in physical memory
[  288.095728] 563c7acfd000 - 563c7acfe000: 4096 bytes, 6 pages in physical memory
[  288.095729] 563c7acfe000 - 563c7acff000: 4096 bytes, 6 pages in physical memory
[  288.095730] 563c7acff000 - 563c7ad20000: 135168 bytes, 6 pages in physical memory
[  288.095732] 7fd90cfd0000 - 7fd912f2f000: 100003840 bytes, 6 pages in physical memory [L]
[  288.095733] 7fd912f2f000 - 7fd912f51000: 139264 bytes, 6 pages in physical memory
[  288.095734] 7fd912f51000 - 7fd9130c9000: 1540096 bytes, 6 pages in physical memory
[  288.095735] 7fd9130c9000 - 7fd913117000: 319488 bytes, 6 pages in physical memory
[  288.095736] 7fd913117000 - 7fd91311b000: 16384 bytes, 6 pages in physical memory
[  288.095737] 7fd91311b000 - 7fd91311d000: 8192 bytes, 6 pages in physical memory
[  288.095738] 7fd91311d000 - 7fd913123000: 24576 bytes, 6 pages in physical memory
[  288.095739] 7fd913134000 - 7fd913135000: 4096 bytes, 6 pages in physical memory
[  288.095740] 7fd913135000 - 7fd913158000: 143360 bytes, 6 pages in physical memory
[  288.095741] 7fd913158000 - 7fd913160000: 32768 bytes, 6 pages in physical memory
[  288.095742] 7fd913161000 - 7fd913162000: 4096 bytes, 6 pages in physical memory
[  288.095743] 7fd913162000 - 7fd913163000: 4096 bytes, 6 pages in physical memory
[  288.095744] 7fd913163000 - 7fd913164000: 4096 bytes, 6 pages in physical memory
[  288.095744] 7ffe237cf000 - 7ffe237f0000: 135168 bytes, 6 pages in physical memory
[  288.095745] 7ffe237f2000 - 7ffe237f6000: 16384 bytes, 6 pages in physical memory
[  288.095746] 7ffe237f6000 - 7ffe237f8000: 8192 bytes, 6 pages in physical memory
```

4. If the OS kernel uses virtual memory with demand paging but does not provide mlock-like functions, then what would be a workaround that a user-level program can do to mitigate unpredictable memory access delay at runtime? Recall lecture slides.

With demand paging, only the in-memory pages needed by the program are loaded into physical memory. When a program tries to access a page that is not currently in physical memory, the operating system generates a page error and loads the required page from disk into memory. Paging to disk can be much slower than accessing physical memory, and excessive paging can cause significant performance degradation.

To mitigate demand paging and copy-on-write, touch allocated memory in the initialization phase.

The touch method may not be very suitable for programs with large memory requirements, as it wastes a lot of system resources back. Based on the results tested in the first question, we can see that the difference in memory access time between the two is relatively small in the case of smaller memory access requirements. However, for programs with large memory access requirements, having lock will improve the efficiency a lot.

Summary of The Contributions

(Note: Bonus Implemented 4.3.3)

Overall:

Hengshuo Zhang	50%
Zhaoze Sun	50%

Breakdown to each Assignment

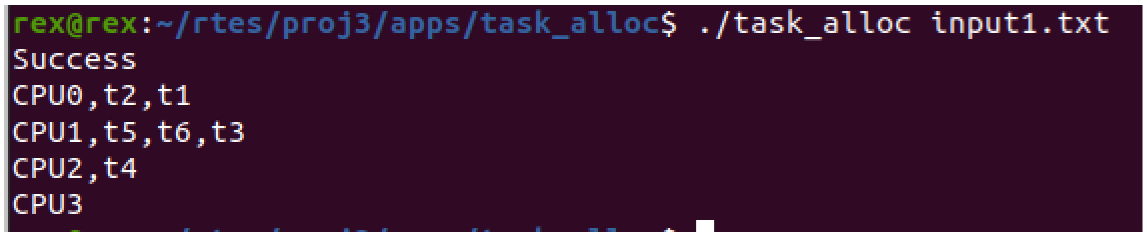
4.1	Multiprocessor task allocation	Hengshuo/Zhaoze
4.2	Memory access time	
4.2.1	User-level memory access time check	Hengshuo/Zhaoze
4.2.2	Memory locking	Hengshuo/Zhaoze
4.3	Process memory viewer	
4.3.1	Process memory segment	Hengshuo/Zhaoze
4.3.2	Memory map of a process	Hengshuo/Zhaoze
4.3.3	Resident pages in memory areas (bonus)	Hengshuo/Zhaoze
4.6	Writeup	Hengshuo/Zhaoze

Demo test for each question in the next page.

Demo Test

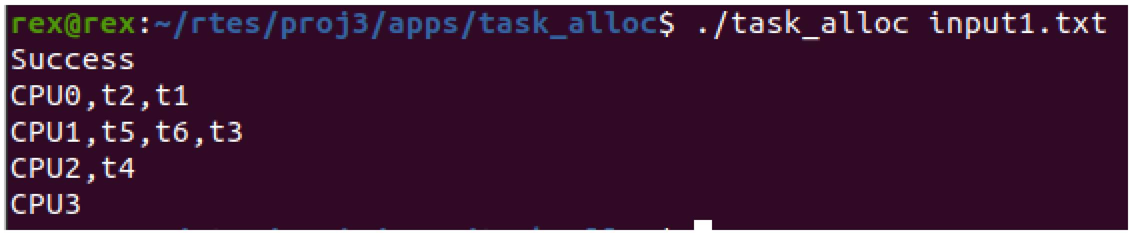
4.1 Multiprocessor task allocation

FFD



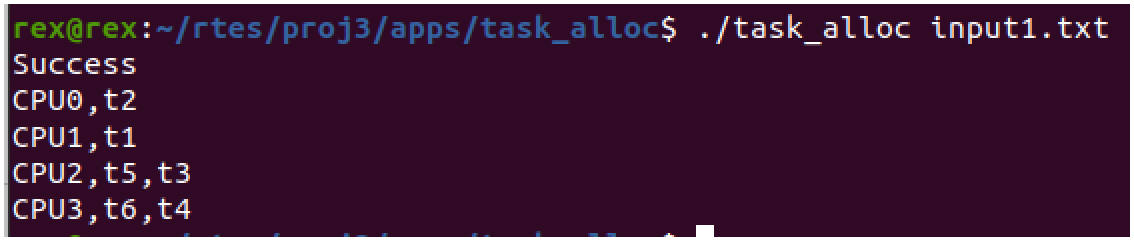
```
rex@rex:~/rtes/proj3/apps/task_alloc$ ./task_alloc input1.txt
Success
CPU0,t2,t1
CPU1,t5,t6,t3
CPU2,t4
CPU3
```

BFD



```
rex@rex:~/rtes/proj3/apps/task_alloc$ ./task_alloc input1.txt
Success
CPU0,t2,t1
CPU1,t5,t6,t3
CPU2,t4
CPU3
```

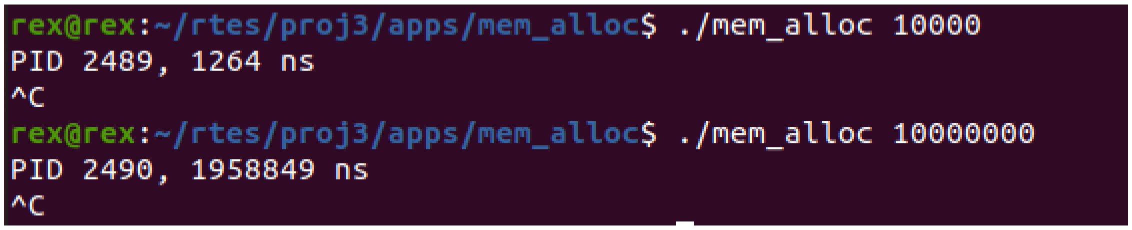
WFD



```
rex@rex:~/rtes/proj3/apps/task_alloc$ ./task_alloc input1.txt
Success
CPU0,t2
CPU1,t1
CPU2,t5,t3
CPU3,t6,t4
```

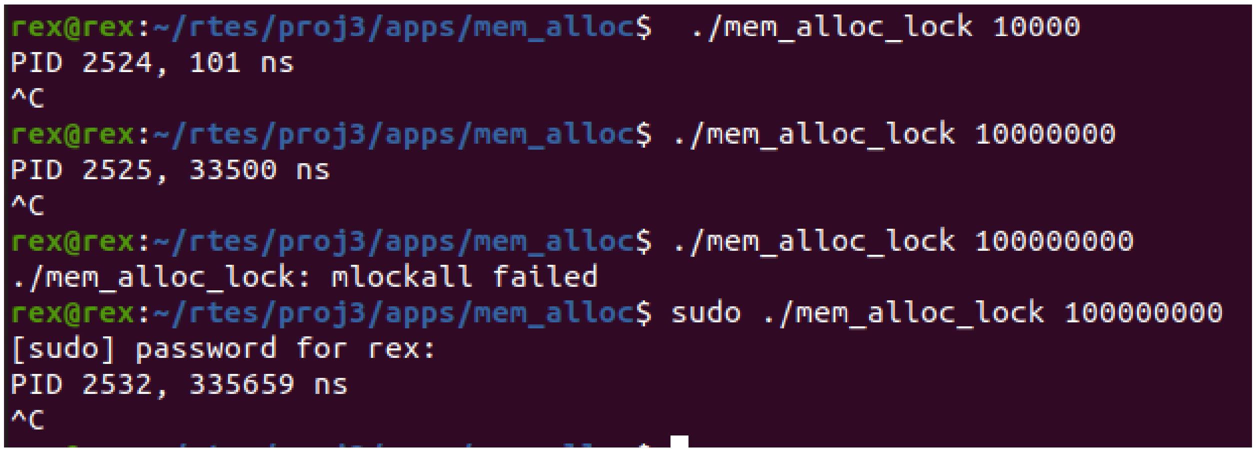
4.2 Memory access time

4.2.1 User-level memory access time check



```
rex@rex:~/rtes/proj3/apps/mem_alloc$ ./mem_alloc 10000
PID 2489, 1264 ns
^C
rex@rex:~/rtes/proj3/apps/mem_alloc$ ./mem_alloc 10000000
PID 2490, 1958849 ns
^C
```

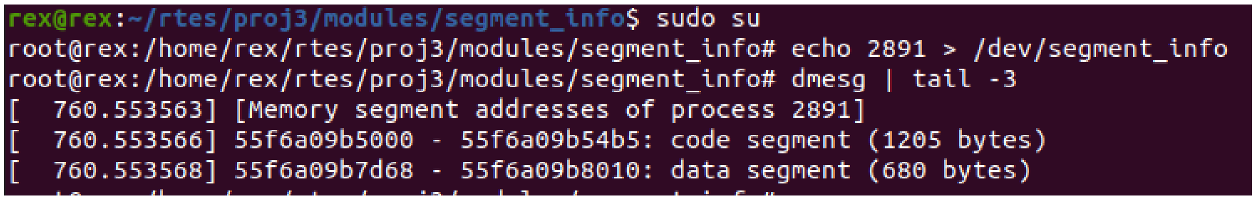
4.2.2 Memory locking



```
rex@rex:~/rtes/proj3/apps/mem_alloc$  ./mem_alloc_lock 10000
PID 2524, 101 ns
^C
rex@rex:~/rtes/proj3/apps/mem_alloc$ ./mem_alloc_lock 10000000
PID 2525, 33500 ns
^C
rex@rex:~/rtes/proj3/apps/mem_alloc$ ./mem_alloc_lock 100000000
./mem_alloc_lock: mlockall failed
rex@rex:~/rtes/proj3/apps/mem_alloc$ sudo ./mem_alloc_lock 100000000
[sudo] password for rex:
PID 2532, 335659 ns
^C
```

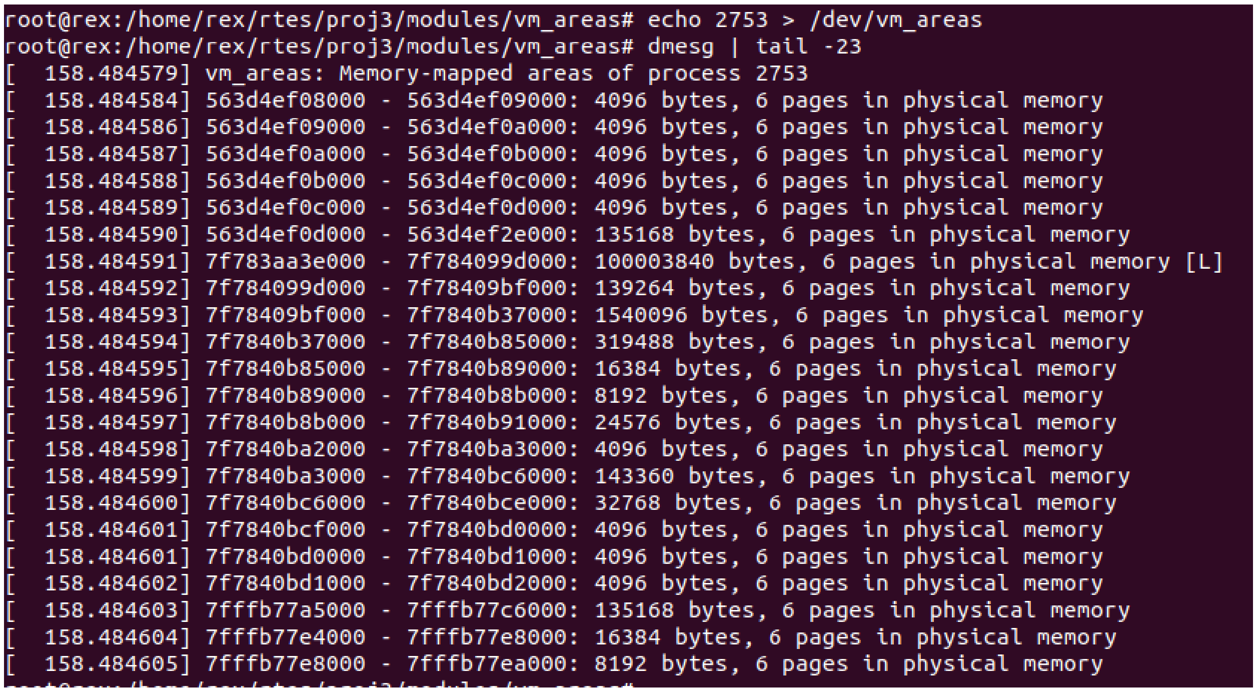
4.3 Process memory viewer

4.3.1 Process memory segment



```
rex@rex:~/rtes/proj3/modules/segment_info$ sudo su
root@rex:/home/rex/rtes/proj3/modules/segment_info# echo 2891 > /dev/segment_info
root@rex:/home/rex/rtes/proj3/modules/segment_info# dmesg | tail -3
[  760.553563] [Memory segment addresses of process 2891]
[  760.553566] 55f6a09b5000 - 55f6a09b54b5: code segment (1205 bytes)
[  760.553568] 55f6a09b7d68 - 55f6a09b8010: data segment (680 bytes)
```

4.3.2 Memory map of a process & 4.3.3 Resident pages in memory areas



```
root@rex:/home/rex/rtes/proj3/modules/vm_areas# echo 2753 > /dev/vm_areas
root@rex:/home/rex/rtes/proj3/modules/vm_areas# dmesg | tail -23
[  158.484579] vm_areas: Memory-mapped areas of process 2753
[  158.484584] 563d4ef08000 - 563d4ef09000: 4096 bytes, 6 pages in physical memory
[  158.484586] 563d4ef09000 - 563d4ef0a000: 4096 bytes, 6 pages in physical memory
[  158.484587] 563d4ef0a000 - 563d4ef0b000: 4096 bytes, 6 pages in physical memory
[  158.484588] 563d4ef0b000 - 563d4ef0c000: 4096 bytes, 6 pages in physical memory
[  158.484589] 563d4ef0c000 - 563d4ef0d000: 4096 bytes, 6 pages in physical memory
[  158.484590] 563d4ef0d000 - 563d4ef2e000: 135168 bytes, 6 pages in physical memory
[  158.484591] 7f783aa3e000 - 7f784099d000: 100003840 bytes, 6 pages in physical memory [L]
[  158.484592] 7f784099d000 - 7f78409bf000: 139264 bytes, 6 pages in physical memory
[  158.484593] 7f78409bf000 - 7f7840b37000: 1540096 bytes, 6 pages in physical memory
[  158.484594] 7f7840b37000 - 7f7840b85000: 319488 bytes, 6 pages in physical memory
[  158.484595] 7f7840b85000 - 7f7840b89000: 16384 bytes, 6 pages in physical memory
[  158.484596] 7f7840b89000 - 7f7840b8b000: 8192 bytes, 6 pages in physical memory
[  158.484597] 7f7840b8b000 - 7f7840b91000: 24576 bytes, 6 pages in physical memory
[  158.484598] 7f7840ba2000 - 7f7840ba3000: 4096 bytes, 6 pages in physical memory
[  158.484599] 7f7840ba3000 - 7f7840bc6000: 143360 bytes, 6 pages in physical memory
[  158.484600] 7f7840bc6000 - 7f7840bce000: 32768 bytes, 6 pages in physical memory
[  158.484601] 7f7840bcf000 - 7f7840bd0000: 4096 bytes, 6 pages in physical memory
[  158.484601] 7f7840bd0000 - 7f7840bd1000: 4096 bytes, 6 pages in physical memory
[  158.484602] 7f7840bd1000 - 7f7840bd2000: 4096 bytes, 6 pages in physical memory
[  158.484603] 7fffb77a5000 - 7fffb77c6000: 135168 bytes, 6 pages in physical memory
[  158.484604] 7fffb77e4000 - 7fffb77e8000: 16384 bytes, 6 pages in physical memory
[  158.484605] 7fffb77e8000 - 7fffb77ea000: 8192 bytes, 6 pages in physical memory
```