BILKENT UNIVERSITY
SPRING 2018
CS342: OPERATING SYSTEMS
PROJECT 3 REPORT

BERAT BİÇER - 21503050
ECEM İLGÜN – 21502157

GENERAL INFORMATION

This report includes experiment results of project 3. In the experiment, a custom test program is used for measuring stack and heap growth of its PCB. Source code of the test program is located in **app.c** file, while source code of the kernel module is **osp3.c**. Tests are performed by comparing boundaries and size of heap and stack section of PCB of the process, with respect to different parameters as follows:

- For heap, we used a struct consisting a char array of length 1024, an integer and a next pointer of the same type. Heap test depends on the number of structs allocated as nodes of a linked list.
- For stack, we used a recursive function that finds the sum of first S elements where S is the function parameter.
- For page table, we accessed top level pgd directory, and printed the valid entries it has, while mapping each bit to its corresponding content.

We measured heap growth for values of H = {500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000, 20000}, where H represents the total number of nodes allocated in the linked list. It is used to measure the effect of total number of nodes on heap size.

We measured stack growth for values of S ={5000, 7500, 10000, 12500, 15000, 17500, 20000, 30000, 40000, 50000}, where S represents the largest integer included in the sum of S integers from 1 to S. The largest integer has linear correlation with the number of recursive calls made by the test program (app.c). It is used to measure the effect of the number of recursive calls on heap size.

ENVIRONMENT AND SETUP

The experiment in which the experiments run is as follows: Ubuntu 18.04 LTS 64-bit and Windows 10 64-bit Dual Boot, Intel Core i7-6700HQ CPU @2.60 GHz x 8 with 16 GB RAM at GNOME version 3.28.1. Tests are performed using a single test file, with different parameters.

RESULTS AND DISCUSSIONS

We conducted the experiment with the following procedure:
First we execute the program (**app.c**). Program has break points where the user is required to give input from the console. This way, we allow the tester to load the kernel module **osp3.ko**, remove it and see the content of PCB of the process via **dmesg** command. Then, test program reads an input from the console and test is initialized.
First test is heap test where linked list grows from the heap until we read the char '*' from the console. When the input is requested, the tester may repeat the (load kernel module – remove kernel module – **dmesg** for log output) cycle to see the difference from the previous log output.
After heap test is done, we deallocate the linked list and start the stack test. For stack test, tester is given the opportunity to repeat the test cycle before the test and at the largest stack point, which is the base case. Then, recursion is terminated and the test program quits.
Each time the kernel module is loaded and removed for heap/stack testing, we also receive content of pgd.

After each run of the procedure described above, we compared the **dmesg** data we received from the experiments, and prepared the following tables and plots.

We made 3 runs for both heap and stack testing.
We made 5 additional runs for the pgd entries in use during first run of **app.c** in order to inspect pgd indices that kept occuring in different runs.

```
[28477.325626] Part 2b
[28477.325627] -----Heap Start: 94242461421568
[28477.325628] -----Heap End: 94242461556736
[28477.325629] -----Heap Size: 135168
[28477.325630] -----Stack Start: 140720615309312
[28477.325630] -----Stack Current End: 140720615444480
[28477.325631] -----Stack Current Size: 135168
```

Figure 1. Example output of initial heap & stack sizes

We have observed from the outputs, like figure 1, that by default, heap and stack size of the process are equal, 135168 bytes in decimal.

HEAP TESTS:

node count	Test 1 heap growth	Test 2 heap growth	Test 3 heap growth	# of allocation with mallocs
500	405504	405504	405504	520000
1000	946176	946176	946176	1040000
1500	1486848	1486848	1486848	1560000
2000	2027520	2027520	2027520	2080000
2500	2568192	2568192	2568192	2600000
3000	3108864	3108864	3108864	3120000
3500	3649536	3649536	3649536	3640000
4000	4190208	4190208	4190208	4160000
4500	4730880	4730880	4730880	4680000
5000	5271552	5271552	5271552	5200000
6000	6217728	6217728	6217728	6240000
7000	7299072	7299072	7299072	7280000
8000	8380416	8380416	8380416	8320000
9000	9461760	9461760	9461760	9360000
10000	10543104	10543104	10543104	10400000
12000	12570624	12570624	12570624	12480000
14000	14733312	14733312	14733312	14560000
16000	16896000	16896000	16896000	16640000
18000	18923520	18923520	18923520	18720000
20000	21086208	21086208	21086208	20800000

Table 1. Node Count vs Heap Growth (in bytes) vs Amount of Mallocs (in bytes)

Amount of allocation with mallocs were directly proportional with node count, with a ratio of 1 to 1040 (1040 bytes = sizeof(struct test) = 1024 * sizeof(char) + sizeof(int) + sizeof(struct test*)), where 1040 bytes is the amount of memory needed for each node. Therefore, any relation about amount of allocation with mallocs can be inferred from node count. Hence we didn't provide any further plot of Total Amount of Allocation with mallocs.

Heap growth was the same for all test runs, which indicates that heap growth depended solely on the amount of malloc() allocation used, which stays the same for all runs. This is logical since the only time data will be allocated from heap is when we specifically ask the data from the heap, which happens only with malloc() commands within our test program.

It is noticeable that heap growth is not equal to the amount of memory malloc() allocates. The reason might be the following: since each process has a default heap size, kernel first allocates space from this existing heap. When the heap is full, kernel allocates more heap area to the process and user allocation continues from this area. Therefore, this proves heap growth occurs only when existing heap is full.

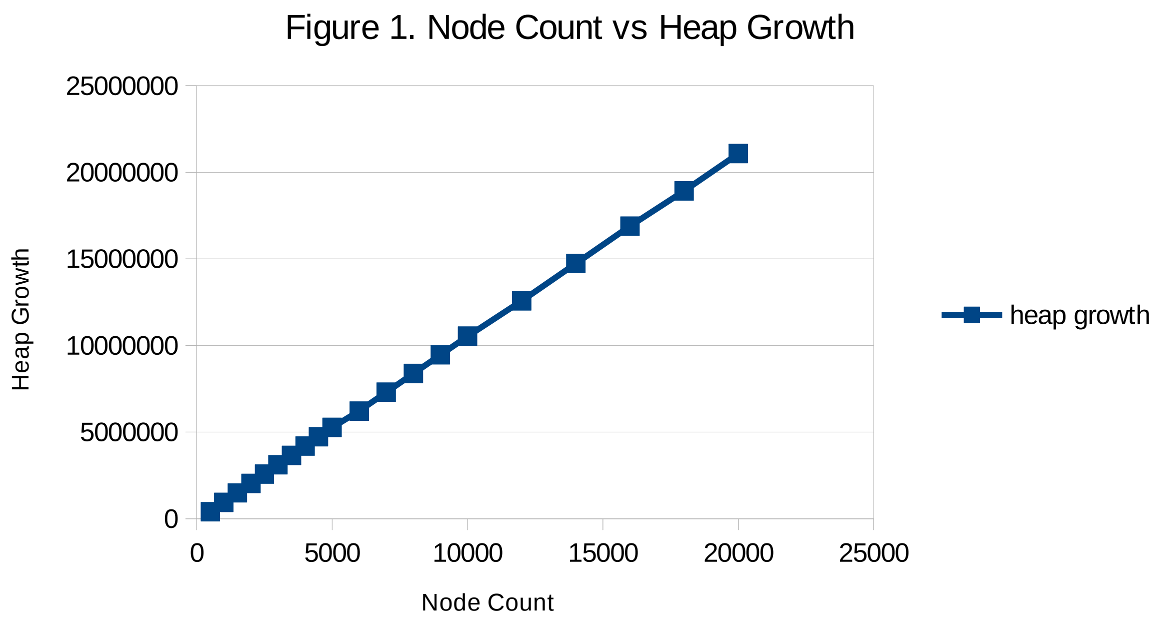

Figure 2. Node Count vs Heap Growth

Figure 2 suggests a linear proportion between node count and heap growth. This proportionality happens since number of nodes directly affect the number of malloc() statements, which in turn directly affects heap allocation.

STACK TESTS:

S	Test 1 stack growth	Test 2 stack growth	Test 3 stack growth
5000	36864	36864	36864
7500	114688	118784	110592
10000	196608	192512	196608
12500	274432	278528	278528
15000	356352	356352	356352
17500	438272	434176	434176
20000	520192	516096	516096
30000	839680	835584	835584
40000	1155072	1159168	1150976
50000	1474560	1474560	1478656

Table 2. Largest integer S vs Stack Growth (in bytes)

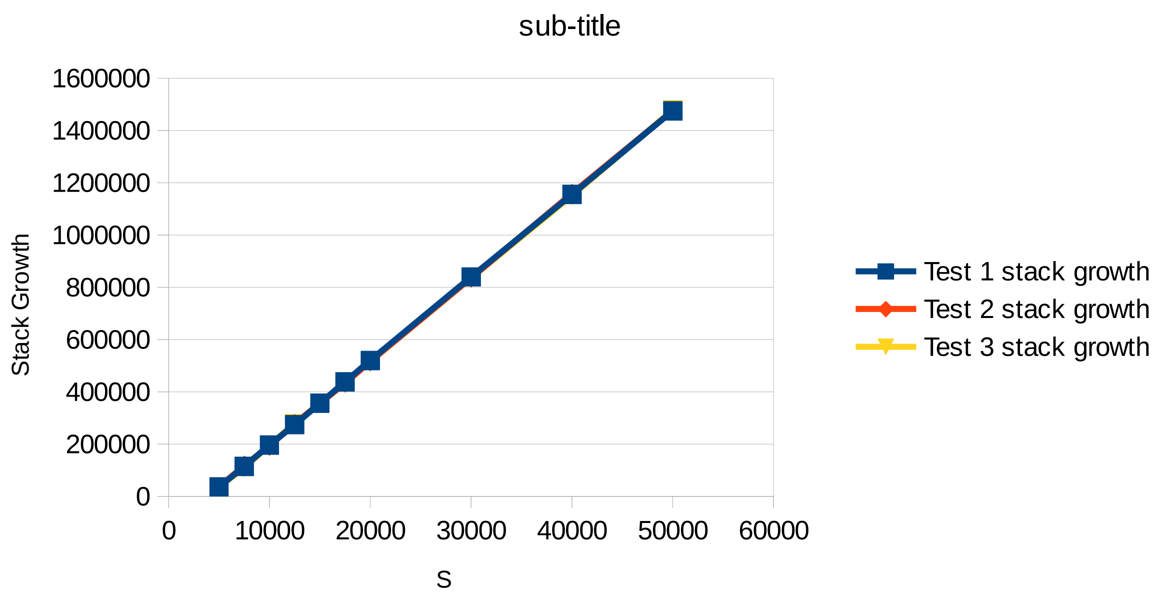

Figure 3. Largest integer S vs Stack Growth (in bytes)

We have observed that stack growth was different for the same parameter S for different test runs. This problem can only be explained only if stack growth implementation of Linux kernel is known. We would have to check each module running at that point to see what might have caused the differences of stack growths for that particular run. However, we have a guess that it might have been due to context switches happening during our calculation, e.g. a context switch might have posed the need to save some variables or such data in stack in order to be able to retrieve them once another context switch returns the cpu to the test process. Therefore, at this point, it is impossible to know for sure.

PAGE TABLE TESTS:

Test 1	Test 2	Test 3	Test 4	Test 5
pgd[172]	pgd[171]	pgd[171]	pgd[0]	pgd[0]
pgd[254]	pgd[255]	pgd[254]	pgd[254]	pgd[255]
pgd[255]	pgd[273]	pgd[255]	pgd[255]	pgd[320]
pgd[273]	pgd[334]	pgd[273]	pgd[320]	pgd[367]
pgd[334]	pgd[398]	pgd[334]	pgd[367]	pgd[431]
pgd[398]	pgd[419]	pgd[398]	pgd[431]	pgd[470]
pgd[419]	pgd[508]	pgd[419]	pgd[470]	pgd[508]
pgd[508]	pgd[510]	pgd[508]	pgd[508]	pgd[510]
pgd[510]	pgd[511]	pgd[510]	pgd[510]	pgd[511]
pgd[511]		pgd[511]	pgd[511]	

Table 3. pgd entries in use, during initiation of app.c in 5 different runs

We have observed from table 3 that some pgd indices kept occuring in different test runs. Therefore we made Table 4, inspecting which test runs shared some nonempty pgd indices with others.

Test 1	Test 2	Test 3	Test 4	Test 5
			0	0
	171	171		
172				
254		254	254	
255	255	255	255	255
273	273	273		
			320	320
334	334	334		
			367	367
398	398	398		
419	419	419		
			431	431
508	508	508	508	508
510	510	510	510	510
511	511	511	511	511

Table 4. pgd entries in use, during initiation of app.c in 5 different runs (version 2)

Table 4 shows that all 5 test runs shared pgd[255], pgd[508], pgd[510], pgd[511] entries. Also, each pgd entry with some index occurred in at least one other run. This might suggest that Unix kernel is using hashing to map process data into page table entries. If we assume that pgd[255], pgd[508], pgd[510], pgd[511] entries might point to pages which all map to the same data across different runs.

Appendix A.

Sample Page Table 1.

[28273.685400] -----pgd[172]
[28273.685400] -----P: 1
[28273.685401] -----R/W: 1
[28273.685402] -----U/S: 1
[28273.685402] -----PWT: 0
[28273.685403] -----PCD: 0
[28273.685404] -----A: 1
[28273.685404] -----PS: 0
[28273.685405] -----PTE bits: 8
[28273.685406] -----PMD bits: 9
[28273.685406] -----PUD bits: 8
[28273.685407] -----PGD bits: 0
[28273.685407] -----XD: 1

[28273.685409] -----pgd[254]
[28273.685410] -----P: 1
[28273.685410] -----R/W: 1
[28273.685411] -----U/S: 1
[28273.685412] -----PWT: 0
[28273.685412] -----PCD: 0
[28273.685413] -----A: 1
[28273.685414] -----PS: 0

[28273.685414] -----PTE bits: 0
[28273.685415] -----PMD bits: 8
[28273.685416] -----PUD bits: 9
[28273.685416] -----PGD bits: 0
[28273.685417] -----XD: 1

[28273.685418] -----pgd[255]
[28273.685419] -----P: 1
[28273.685419] -----R/W: 1
[28273.685420] -----U/S: 1
[28273.685421] -----PWT: 0
[28273.685421] -----PCD: 0
[28273.685422] -----A: 1
[28273.685422] -----PS: 0
[28273.685423] -----PTE bits: 1
[28273.685424] -----PMD bits: 1
[28273.685424] -----PUD bits: 9
[28273.685425] -----PGD bits: 0
[28273.685426] -----XD: 1

[28273.685427] -----pgd[273]
[28273.685427] -----P: 1
[28273.685428] -----R/W: 1
[28273.685429] -----U/S: 1
[28273.685430] -----PWT: 0
[28273.685430] -----PCD: 0
[28273.685431] -----A: 1
[28273.685432] -----PS: 0
[28273.685432] -----PTE bits: 8
[28273.685433] -----PMD bits: 8
[28273.685433] -----PUD bits: 9
[28273.685434] -----PGD bits: 0
[28273.685435] -----XD: 0

[28273.685436] -----pgd[334]
[28273.685437] -----P: 1
[28273.685437] -----R/W: 1
[28273.685438] -----U/S: 1
[28273.685438] -----PWT: 0
[28273.685439] -----PCD: 0
[28273.685439] -----A: 1
[28273.685440] -----PS: 0
[28273.685441] -----PTE bits: 8
[28273.685441] -----PMD bits: 8
[28273.685442] -----PUD bits: 1
[28273.685443] -----PGD bits: 0
[28273.685443] -----XD: 0

[28273.685445] -----pgd[398]
[28273.685445] -----P: 1
[28273.685446] -----R/W: 1
[28273.685446] -----U/S: 1

[28273.685447] -----PWT: 0
[28273.685448] -----PCD: 0
[28273.685448] -----A: 1
[28273.685449] -----PS: 0
[28273.685449] -----PTE bits: 9
[28273.685450] -----PMD bits: 1
[28273.685451] -----PUD bits: 1
[28273.685451] -----PGD bits: 0
[28273.685452] -----XD: 0

[28273.685453] -----pgd[419]
[28273.685453] -----P: 1
[28273.685454] -----R/W: 1
[28273.685454] -----U/S: 1
[28273.685455] -----PWT: 0
[28273.685455] -----PCD: 0
[28273.685456] -----A: 1
[28273.685456] -----PS: 0
[28273.685457] -----PTE bits: 8
[28273.685457] -----PMD bits: 9
[28273.685457] -----PUD bits: 1
[28273.685458] -----PGD bits: 0
[28273.685458] -----XD: 0

[28273.685459] -----pgd[508]
[28273.685460] -----P: 1
[28273.685460] -----R/W: 1
[28273.685461] -----U/S: 1
[28273.685461] -----PWT: 0
[28273.685462] -----PCD: 0
[28273.685462] -----A: 1
[28273.685463] -----PS: 0
[28273.685463] -----PTE bits: 9
[28273.685464] -----PMD bits: 9
[28273.685464] -----PUD bits: 1
[28273.685464] -----PGD bits: 0
[28273.685465] -----XD: 0

[28273.685466] -----pgd[510]
[28273.685466] -----P: 1
[28273.685467] -----R/W: 1
[28273.685467] -----U/S: 1
[28273.685468] -----PWT: 0
[28273.685468] -----PCD: 0
[28273.685468] -----A: 1
[28273.685469] -----PS: 0
[28273.685469] -----PTE bits: 0
[28273.685470] -----PMD bits: 9
[28273.685470] -----PUD bits: 9
[28273.685471] -----PGD bits: 0
[28273.685471] -----XD: 0

[28273.685472] -----pgd[511]
[28273.685472] -----P: 1
[28273.685473] -----R/W: 1
[28273.685473] -----U/S: 1
[28273.685474] -----PWT: 0
[28273.685474] -----PCD: 0
[28273.685475] -----A: 1
[28273.685475] -----PS: 0
[28273.685476] -----PTE bits: 8
[28273.685476] -----PMD bits: 9
[28273.685477] -----PUD bits: 9
[28273.685477] -----PGD bits: 0
[28273.685478] -----XD: 0

Sample Page Table 2.
[29603.541610] Part 2c
[29603.541611] -----pgd[171]
[29603.541612] -----P: 1
[29603.541612] -----R/W: 1
[29603.541613] -----U/S: 1
[29603.541613] -----PWT: 0
[29603.541614] -----PCD: 0
[29603.541614] -----A: 1
[29603.541615] -----PS: 0
[29603.541616] -----PTE bits: 1
[29603.541616] -----PMD bits: 1
[29603.541617] -----PUD bits: 8
[29603.541617] -----PGD bits: 0
[29603.541618] -----XD: 1

[29603.541619] -----pgd[255]
[29603.541619] -----P: 1
[29603.541620] -----R/W: 1
[29603.541620] -----U/S: 1
[29603.541621] -----PWT: 0
[29603.541621] -----PCD: 0
[29603.541622] -----A: 1
[29603.541622] -----PS: 0
[29603.541623] -----PTE bits: 9
[29603.541623] -----PMD bits: 0
[29603.541624] -----PUD bits: 9
[29603.541624] -----PGD bits: 0
[29603.541625] -----XD: 1

[29603.541626] -----pgd[273]
[29603.541626] -----P: 1
[29603.541627] -----R/W: 1
[29603.541627] -----U/S: 1
[29603.541628] -----PWT: 0
[29603.541628] -----PCD: 0
[29603.541629] -----A: 1
[29603.541629] -----PS: 0

[29603.541630] -----PTE bits: 8
[29603.541630] -----PMD bits: 8
[29603.541631] -----PUD bits: 9
[29603.541632] -----PGD bits: 0
[29603.541632] -----XD: 0

[29603.541633] -----pgd[334]
[29603.541634] -----P: 1
[29603.541634] -----R/W: 1
[29603.541635] -----U/S: 1
[29603.541635] -----PWT: 0
[29603.541636] -----PCD: 0
[29603.541636] -----A: 1
[29603.541637] -----PS: 0
[29603.541637] -----PTE bits: 8
[29603.541638] -----PMD bits: 8
[29603.541639] -----PUD bits: 1
[29603.541639] -----PGD bits: 0
[29603.541640] -----XD: 0

[29603.541641] -----pgd[398]
[29603.541641] -----P: 1
[29603.541642] -----R/W: 1
[29603.541642] -----U/S: 1
[29603.541643] -----PWT: 0
[29603.541660] -----PCD: 0
[29603.541665] -----A: 1
[29603.541668] -----PS: 0
[29603.541671] -----PTE bits: 9
[29603.541674] -----PMD bits: 1
[29603.541679] -----PUD bits: 1
[29603.541682] -----PGD bits: 0
[29603.541686] -----XD: 0

[29603.541691] -----pgd[419]
[29603.541693] -----P: 1
[29603.541695] -----R/W: 1
[29603.541697] -----U/S: 1
[29603.541699] -----PWT: 0
[29603.541701] -----PCD: 0
[29603.541703] -----A: 1
[29603.541705] -----PS: 0
[29603.541707] -----PTE bits: 8
[29603.541709] -----PMD bits: 9
[29603.541711] -----PUD bits: 1
[29603.541713] -----PGD bits: 0
[29603.541714] -----XD: 0

[29603.541719] -----pgd[508]
[29603.541721] -----P: 1
[29603.541723] -----R/W: 1
[29603.541725] -----U/S: 1

[29603.541727] -----PWT: 0
[29603.541728] -----PCD: 0
[29603.541729] -----A: 1
[29603.541731] -----PS: 0
[29603.541732] -----PTE bits: 9
[29603.541734] -----PMD bits: 9
[29603.541736] -----PUD bits: 1
[29603.541739] -----PGD bits: 0
[29603.541744] -----XD: 0

[29603.541750] -----pgd[510]
[29603.541752] -----P: 1
[29603.541753] -----R/W: 1
[29603.541755] -----U/S: 1
[29603.541757] -----PWT: 0
[29603.541759] -----PCD: 0
[29603.541760] -----A: 1
[29603.541762] -----PS: 0
[29603.541763] -----PTE bits: 0
[29603.541765] -----PMD bits: 9
[29603.541767] -----PUD bits: 9
[29603.541769] -----PGD bits: 0
[29603.541770] -----XD: 0

[29603.541775] -----pgd[511]
[29603.541778] -----P: 1
[29603.541780] -----R/W: 1
[29603.541782] -----U/S: 1
[29603.541784] -----PWT: 0
[29603.541786] -----PCD: 0
[29603.541787] -----A: 1
[29603.541789] -----PS: 0
[29603.541791] -----PTE bits: 8
[29603.541793] -----PMD bits: 9
[29603.541795] -----PUD bits: 9
[29603.541797] -----PGD bits: 0
[29603.541799] -----XD: 0

Sample Page Table 3.
[29646.732236] Part 2c
[29646.732237] -----pgd[171]
[29646.732238] -----P: 1
[29646.732238] -----R/W: 1
[29646.732239] -----U/S: 1
[29646.732240] -----PWT: 0
[29646.732240] -----PCD: 0
[29646.732240] -----A: 1
[29646.732241] -----PS: 0
[29646.732242] -----PTE bits: 1
[29646.732242] -----PMD bits: 8
[29646.732242] -----PUD bits: 9
[29646.732243] -----PGD bits: 0

[29646.732243] -----XD: 1

[29646.732244] -----pgd[254]
[29646.732245] -----P: 1
[29646.732245] -----R/W: 1
[29646.732246] -----U/S: 1
[29646.732246] -----PWT: 0
[29646.732247] -----PCD: 0
[29646.732247] -----A: 1
[29646.732248] -----PS: 0
[29646.732248] -----PTE bits: 9
[29646.732248] -----PMD bits: 1
[29646.732249] -----PUD bits: 8
[29646.732249] -----PGD bits: 0
[29646.732250] -----XD: 1

[29646.732251] -----pgd[255]
[29646.732251] -----P: 1
[29646.732251] -----R/W: 1
[29646.732252] -----U/S: 1
[29646.732252] -----PWT: 0
[29646.732253] -----PCD: 0
[29646.732253] -----A: 1
[29646.732254] -----PS: 0
[29646.732254] -----PTE bits: 8
[29646.732254] -----PMD bits: 1
[29646.732255] -----PUD bits: 8
[29646.732255] -----PGD bits: 0
[29646.732256] -----XD: 1

[29646.732257] -----pgd[273]
[29646.732257] -----P: 1
[29646.732257] -----R/W: 1
[29646.732258] -----U/S: 1
[29646.732258] -----PWT: 0
[29646.732259] -----PCD: 0
[29646.732259] -----A: 1
[29646.732260] -----PS: 0
[29646.732260] -----PTE bits: 8
[29646.732260] -----PMD bits: 8
[29646.732261] -----PUD bits: 9
[29646.732261] -----PGD bits: 0
[29646.732262] -----XD: 0

[29646.732263] -----pgd[334]
[29646.732263] -----P: 1
[29646.732264] -----R/W: 1
[29646.732264] -----U/S: 1
[29646.732265] -----PWT: 0
[29646.732265] -----PCD: 0
[29646.732266] -----A: 1
[29646.732266] -----PS: 0

[29646.732267] -----PTE bits: 8
[29646.732267] -----PMD bits: 8
[29646.732268] -----PUD bits: 1
[29646.732268] -----PGD bits: 0
[29646.732269] -----XD: 0

[29646.732270] -----pgd[398]
[29646.732270] -----P: 1
[29646.732270] -----R/W: 1
[29646.732271] -----U/S: 1
[29646.732272] -----PWT: 0
[29646.732272] -----PCD: 0
[29646.732273] -----A: 1
[29646.732273] -----PS: 0
[29646.732274] -----PTE bits: 9
[29646.732274] -----PMD bits: 1
[29646.732275] -----PUD bits: 1
[29646.732275] -----PGD bits: 0
[29646.732276] -----XD: 0

[29646.732277] -----pgd[419]
[29646.732277] -----P: 1
[29646.732278] -----R/W: 1
[29646.732278] -----U/S: 1
[29646.732279] -----PWT: 0
[29646.732280] -----PCD: 0
[29646.732280] -----A: 1
[29646.732281] -----PS: 0
[29646.732281] -----PTE bits: 8
[29646.732282] -----PMD bits: 9
[29646.732282] -----PUD bits: 1
[29646.732283] -----PGD bits: 0
[29646.732283] -----XD: 0

[29646.732284] -----pgd[508]
[29646.732285] -----P: 1
[29646.732285] -----R/W: 1
[29646.732286] -----U/S: 1
[29646.732286] -----PWT: 0
[29646.732286] -----PCD: 0
[29646.732287] -----A: 1
[29646.732287] -----PS: 0
[29646.732288] -----PTE bits: 9
[29646.732288] -----PMD bits: 9
[29646.732289] -----PUD bits: 1
[29646.732289] -----PGD bits: 0
[29646.732290] -----XD: 0

[29646.732290] -----pgd[510]
[29646.732291] -----P: 1
[29646.732291] -----R/W: 1
[29646.732292] -----U/S: 1

[29646.732292] -----PWT: 0
[29646.732292] -----PCD: 0
[29646.732293] -----A: 1
[29646.732293] -----PS: 0
[29646.732294] -----PTE bits: 0
[29646.732294] -----PMD bits: 9
[29646.732295] -----PUD bits: 9
[29646.732295] -----PGD bits: 0
[29646.732296] -----XD: 0

[29646.732296] -----pgd[511]
[29646.732297] -----P: 1
[29646.732297] -----R/W: 1
[29646.732298] -----U/S: 1
[29646.732298] -----PWT: 0
[29646.732299] -----PCD: 0
[29646.732299] -----A: 1
[29646.732299] -----PS: 0
[29646.732300] -----PTE bits: 8
[29646.732300] -----PMD bits: 9
[29646.732301] -----PUD bits: 9
[29646.732301] -----PGD bits: 0
[29646.732302] -----XD: 0