ECE357 Computer Operating Systems
Fall 2019
Instructor: Prof. Jeff Hakner
Di Mei

Problem Set #5

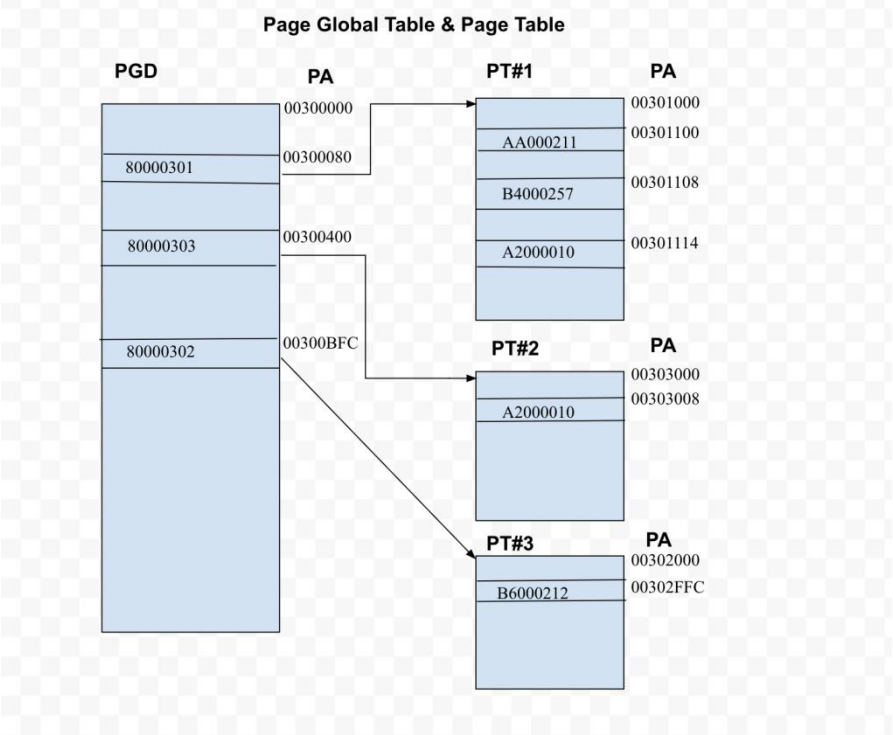
Problem 1:

A)

1st region: text, 2nd region: data, 3rd region: BSS, 4th region: BSS, 5th region: stack

B)

PGD: (PA is obtained based on 1st half of 20 bits)
PA = 00300080, PDE: 80000301
PA = 00300080, PDE: 80000301
PA = 00300080, PDE: 80000301
PA = 00300400, PDE: 80000303
PA = 00300BFC, PDE: 80000302 (stack is allocated earlier than BSS)
PTE: (PA is obtained based on 2nd half of 20 bits)
PA: 00301100, PTE: AA000211 (10101010000000000000001000010001)
PA: 00301108, PTE: B4000257 (10110100000000000000001001010111)
PA: 00301114, PTE: A2000010 (10100010000000000000000000010000)
PA: 00303008, PTE: A2000010 (10100010000000000000000000010000)
PA: 00302FFC, PTE: B6000212 (10110110000000000000001000010010)



C）

```
// x1 in data region: initialized global variable
char x1[] = {0x45, 0x43, 0x45, 0x33, 0x35, 0x37};
// x2 in BSS region: uninitialized global variable
char x2[2*4096];
// main() function is in stack
int main(){
       x1[0] = 0x45; // make x1 dirty
       char v = x2[4096]; // make x2 read and accessed
       // map in BSS region: heap
      char *map = mmap((void*)0x40001000, 3*4096, PROT_READ|PROT_WRITE,
MAP_SHARED|MAP_ANONYMOUS, -1, 0);
      fprintf(stderr, "%c\n", *(map+4096)); // make map read and accessed
      return 0;
}
```

D)

After running the program, the first page fault occurs when the virtual page of text, which is file-mapped with address 0x08040000, is read and executed. This page fault is resolved as a major fault by demanding paging an executable file from disk to the physical address 0x00211000.

The second page fault occurs when the virtual page of stack, which is anonymous, is written by the program. This situation is resolvable as a minor fault: the kernel finds a free page from the page frame pool and also 0-fills that page before the virtual page can be written with any content.

The function main() is now in the stack. Now, we have two “virgin” virtual anonymous pages with virtual addresses 0x08044000 and 0x40001000 in BSS region. Page fault occurs when these two regions are read by the program. This situation is also resolvable as a minor fault: for a read access, the kernel optimizes by mapping both virtual pages to a single, shared virtual page filled with 0s.

The page fault also occurs when the virtual page of data, which is file-mapped with address 0x08042000, is written with 0x45 by the program. This page fault is resolved as a major fault by demanding paging an executable file from disk to the physical address 0x00257000.

E)

The Page Frame Reclamation Algorithm (PFGA) has been active and used in the page frame corresponding to the data region. When the data region is written, the

ACCESSED bit in data region's page is set and then it is cleared. Based on the bit-wise flag PG_referenced, the PFGA scan this page for two times successively. Then, the PFGA moves this page to the tail of the inactive list because it has not been used in two successive scans. For each page in the inactive list, its ACCESSED bit is checked again (3rd time scan) and data region's page is still not used. Thus, the data region's page is reclaimed and ACCESSED bit of that page is off in the diagram shown above.

Problem 2:
Sample Output:

Test#1:
I run this program in MacOS, and the signal received is SIG_BUS (10). When I run it in Linux, the signal received is SIG_SEGV (11).

```
[Davids-MacBook-Pro:program_1 davidmei$ ./p1
Executing Test #1 (write to r/o mmap):
map[3]=='A'
Writing a 'B'
signal 10 received
[Davids-MacBook-Pro:program_1 davidmei$ echo $?
10
```

Test#2:

```
[Davids-MacBook-Pro:program_23 davidmei$ ./p2 2
Executing Test #2 (write to a MAP_SHARED region):
Writing a "ABC" to mapped memory of an empty file
[Davids-MacBook-Pro:program_23 davidmei$ echo $?
0
```

Test#3:

```
[Davids-MacBook-Pro:program_23 davidmei$ ./p2 3
Executing Test #3 (write to a MAP_PRIVATE region):
Writing a "ABC" to mapped memory of an empty file
File's bytes don't change
[Davids-MacBook-Pro:program_23 davidmei$ echo $?
1
```

Test#4:

```
[Davids-MacBook-Pro:program_4 davidmei$ ./p4
Executing Test #4 (write into a hole):
Writing byte "X" one byte beyond 'test_file' mapped memory
Extending 'test_file' by 16 bytes
X byte is visible now
[Davids-MacBook-Pro:program_4 davidmei$ echo $?
0
```