PSET 5

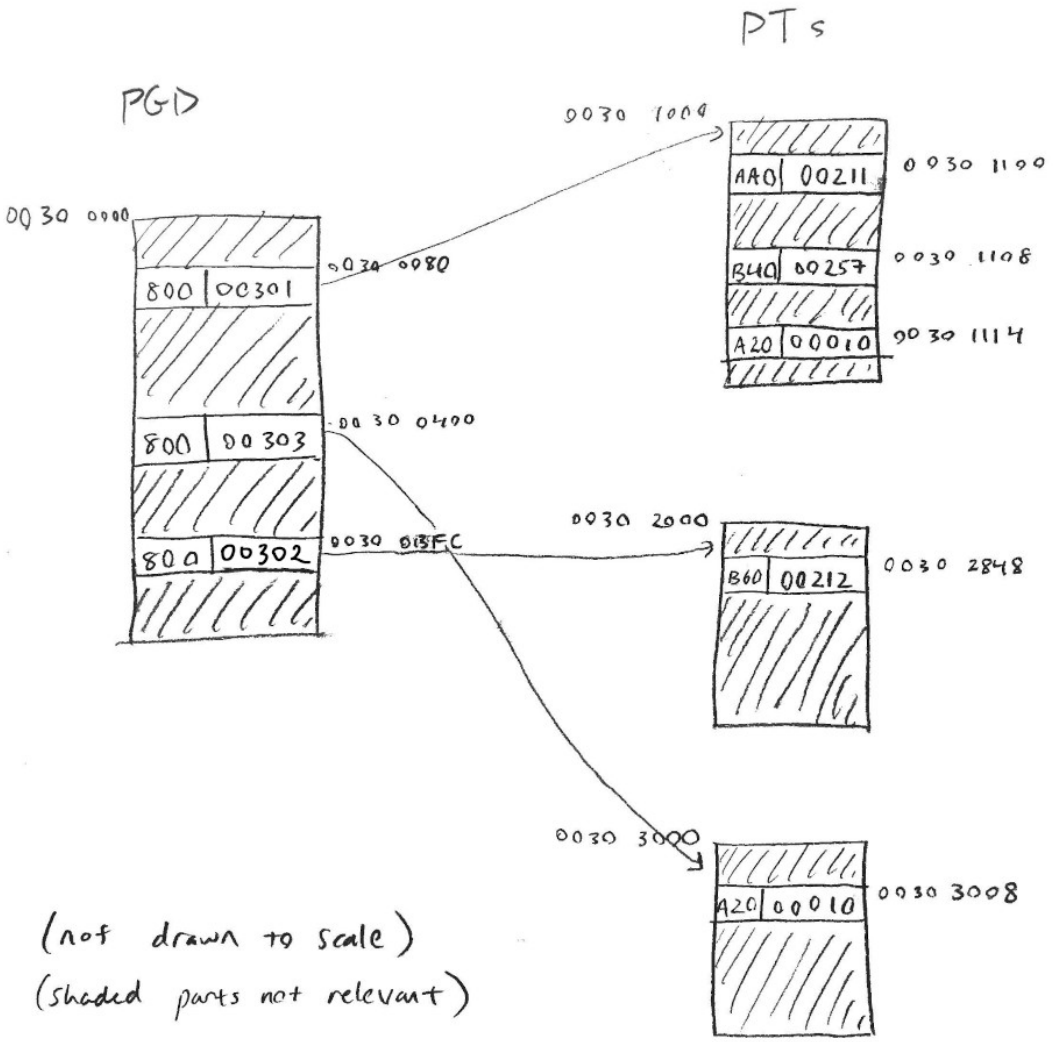
Jonathan Lam
Prof. Hakner
ECE 357
Computer OS
11/20/19

Problem 1 – Page Tables

a) Identfy the type of each memory region, based on the clues.

1) text (clues: map r/x/private, contains opcodes)
2) data (clues: map r/w/private, not empty)
3) bss (clues: map r/w/private, zero-filled)
4) bss – mmapped in (clues: map r/w/shared, zero-filled)
5) stack (clues: high address, map r/w/private/anon/growsdown, filled at end)

b) Draw out the PGD and any PTs that the kernel would have allocated.



c) Compose a brief program to reach the above state.

Note: From the diagram, we only know that these PTEs are marked not present, which can mean either that they have never been accessed (read from/written to/executed) or were paged in at some point and are currently paged out. This program (and the following answers) assumes that none of the non-present, mmap-ed in virtual pages (i.e., 0x40001000, 0x40003000, and 0x08044000) have never been accessed.

```
1      // assume compiler doesn't optimize out useless statements/functions
2
3      // .data
4      unsigned long i = 1, j = 1;
5
6      // two-page .bss
7      char l[8192];
8
9      int main(void) {
10       // keep accessing this to keep .stack accessed
11       int k;
12       char *m;
13
14       // create mmap-ed region
15       // assume 0x40001000 was chosen by mmap, not chosen explicitly
16       m = (char *) mmap(NULL, 12288, PROT_READ|PROT_WRITE,
17                         MAP_ANONYMOUS|MAP_SHARED, -1, 0);
18
19       // dirty .data; assume little-endian (shows correctly in core dump)
20       i = 0x45334543, j = 0x00003537;
21
22       // rapidly access both .bss and .stack regions so that PFRA doesn't
23       // try to reclaim it and access bit stays set
24       while(1) {
25         l[4096];
26         m[4096];
27         k;
28       }
29     }
```

d) Explain each page fault incurred by the program and resolution.

First, there is the minor page fault to create space for the PGD. Next, there is the mmap of the .text, .data, .bss, and .stack regions. These mmaps set up the valid memory regions but do not incur page faults.

Program execution begins in the .text region, so this region gets read, leading into the minor page fault to create the PT (0x00301000) and a major page fault (reading from a.out mapped to VA 0x08040000). The .text region has an instruction to add main to the stack, which causes a minor page fault for the PT (0x00302000) and a minor page fault in the stack, writing to a virgin anonymous page mapped to VA 0xBFFFF000.

In the program in part (c) above, next is a mmap to region 4 (no page fault). After that, there is a major page fault after writing to the data region (reading from a.out mapped to VA 0x08042000). On accessing (reading) the bss region, a minor page fault occurs mapping VA 0x08045FFF to the dedicated kernel page of all zeros (for COW optimization). For the mmap-ed region, a minor page fault occurs for the PT (0x00303000) and a minor page fault to read the data with VA 0x40002000 (similar to above, mapped to dedicated kernel page of all zeros).

This is a total of 2 major and 7 minor page faults.

e) What evidence from the diagram above shows that the PFRA has been recently active? Which page frame(s) might be a candidate for paging-out and reclamation?

Region 2 has been written to (dirty) and is not shared, but not accessed, which means that something (which can only be the PFRA) cleared the accessed bit. The PFRA must have run recently since it would claim pages for paging out if they have the accessed bit left unchecked for too long. Because this page is the only one with the accessed bit unset, it is the most likely candidate for being paged out.

There are no clear candidates for page reclamation just based on this diagram alone. However, if the valid memory regions that are not mapped to any page table have been swapped out (contrary to the assumption made in part (c)), these may be candidates for reclamation.

Problem 2 – mmap test programs

Test program (mtest.c)

```
 1      #include <errno.h>
 2      #include <fcntl.h>
 3      #include <signal.h>
 4      #include <stdio.h>
 5      #include <stdlib.h>
 6      #include <string.h>
 7      #include <sys/mman.h>
 8      #include <unistd.h>
 9
10      // test program vars; chosen arbitrarily and tweakable
11      #define TMP_PATH "/tmp/mtest.tmp"
12      #define TST_IND 3
13      #define TST_LEN 324
14      #define TST_CHR 'A'
15
16      // less strict error reporting, since these errors shouldn't happen
17      int err_chk(int res, char *cmd_name) {
18        if(res<0) {
19          dprintf(2, "./mtest: ERROR: failed on %s: %s.\n",
20                  cmd_name, strerror(errno));
21          exit(EXIT_FAILURE);
```

```
22        }
23        dprintf(2, "./mtest: Performing %s.\n", cmd_name);
24        return res;
25      }
26
27      // auxiliary fns
28      void sig_trap(int sig_num) {
29        dprintf(2, "./mtest: Received signal %d: %s. Exiting.\n",
30                sig_num, strsignal(sig_num));
31        exit(sig_num);
32      }
33      void trap_all_sigs() {
34        for(int i=1; i<=31; i++)
35          signal(i, sig_trap);
36      }
37      int create_tmp_file() {
38        int fd;
39        char buf[] = "Hello, world!";
40
41        // create and fill with some (known) bytes
42        fd = err_chk(open(TMP_PATH, O_CREAT|O_RDWR|O_TRUNC), "creat tmpfile");
43        err_chk(write(fd, buf, sizeof buf), "fill some data to tmpfile");
44        err_chk(unlink(TMP_PATH), "unlink tmpfile");
45
46        return fd;
47      }
48
49      // test 1: PROT_READ violation
50      void test_1() {
51        char *p, prev, post;
52
53        trap_all_sigs();
54        p = (char *) mmap(NULL, 4096, PROT_READ, MAP_ANON|MAP_SHARED, -1, 0);
55        err_chk(-(p==MAP_FAILED), "mmap tmpfile with MAP_ANON|MAP_SHARED");
56
57        prev = p[TST_IND];
58        dprintf(2, "./mtest: p[TST_IND]==%x.\n", prev);
59        dprintf(2, "./mtest: attempting write p[TST_IND]=%x.\n", prev+1);
60        post = ++p[TST_IND];
61
62        // diagnostic message and cleanup
63        err_chk(munmap(p, 4096), "munmap tmpfile");
64        dprintf(2, "./mtest: test byte: pre-write: %x post-write %x.\n", prev, post);
65        exit(-(prev==post));
66      }
67
68      // tests 2/3: writing to MAP_SHARED/MAP_PRIVATE
69      void test_23(int is_shared) {
70        int fd;
71        char *p, prev, post;
72
73        fd = create_tmp_file();
```

```
74        p = (char *) mmap(NULL, 4096, PROT_READ|PROT_WRITE,
75                          MAP_FILE|(is_shared?MAP_SHARED:MAP_PRIVATE), fd, 0);
76        err_chk(-(p==MAP_FAILED), "mmap tmpfile with MAP_SHARED or MAP_PRIVATE");
77
78        prev = p[TST_IND];
79        post = ++p[TST_IND];
80
81        // see if regular read call works
82        err_chk(lseek(fd, TST_IND, SEEK_SET), "lseek in tmpfile");
83        err_chk(read(fd, &post, 1), "read in tmpfile");
84
85        // diagnostic messae and cleanup
86        err_chk(munmap(p, 4096), "munmap tmpfile");
87        err_chk(close(fd), "close tmpfile");
88        dprintf(2, "./mtest: test byte: pre-write: %x post-write %x.\n", prev, post);
89        exit(prev==post);
90      }
91
92      // test 4: writing into a hole
93      void test_4() {
94        int fd;
95        char *p, post;
96
97        fd = create_tmp_file();
98        p = (char *) mmap(NULL, TST_LEN, PROT_READ|PROT_WRITE, MAP_FILE|MAP_SHARED,
99                          fd, 0);
100       err_chk(-(p==MAP_FAILED), "mmap tmpfile with MAP_SHARED");
101
102       p[TST_LEN] = TST_CHR;
103       err_chk(lseek(fd, TST_LEN+16, SEEK_SET), "lseek to TST_LEN+16 in tmpfile");
104       err_chk(write(fd, " ", 1), "write to TST_LEN+16 in tmpfile");
105       err_chk(lseek(fd, TST_LEN, SEEK_SET), "lseek to TST_LEN in tmpfile");
106       err_chk(read(fd, &post, 1), "read in tmpfile");
107
108       // diagnostic messae and cleanup
109       err_chk(munmap(p, TST_LEN), "munmap tmpfile");
110       err_chk(close(fd), "close tmpfile");
111       dprintf(2,"./mtest:test byte: %x; byte at pos TST_LEN+1:%x.\n",TST_CHR,post);
112       exit(1-(TST_CHR==post));
113     }
114
115     // usage: ./a.out [TEST_NUM]
116     int main(int argc, char **argv) {
117       if(argc > 1)
118         switch(**(argv+1)) {
119         case '1': test_1(); break;
120         case '2': test_23(1); break;
121         case '3': test_23(0); break;
122         case '4': test_4(); break;
123         }
124       exit(EXIT_FAILURE);
125     }
```

Sample Output

```
(base) [jon@archijon prog5]$ ./mtest 1
./mtest: Performing mmap tmpfile with MAP_ANON|MAP_SHARED.
./mtest: p[TST_IND]==0.
./mtest: attempting write p[TST_IND]=1.
./mtest: Received signal 11: Segmentation fault. Exiting.
(base) [jon@archijon prog5]$ echo $?
11
(base) [jon@archijon prog5]$ ./mtest 2
./mtest: Performing creat tmpfile.
./mtest: Performing fill some data to tmpfile.
./mtest: Performing unlink tmpfile.
./mtest: Performing mmap tmpfile with MAP_SHARED or MAP_PRIVATE.
./mtest: Performing lseek in tmpfile.
./mtest: Performing read in tmpfile.
./mtest: Performing munmap tmpfile.
./mtest: Performing close tmpfile.
./mtest: test byte: pre-write: 6c post-write 6d.
(base) [jon@archijon prog5]$ echo $?
0
(base) [jon@archijon prog5]$ ./mtest 3
./mtest: Performing creat tmpfile.
./mtest: Performing fill some data to tmpfile.
./mtest: Performing unlink tmpfile.
./mtest: Performing mmap tmpfile with MAP_SHARED or MAP_PRIVATE.
./mtest: Performing lseek in tmpfile.
./mtest: Performing read in tmpfile.
./mtest: Performing munmap tmpfile.
./mtest: Performing close tmpfile.
./mtest: test byte: pre-write: 6c post-write 6c.
(base) [jon@archijon prog5]$ echo $?
1
(base) [jon@archijon prog5]$ ./mtest 4
./mtest: Performing creat tmpfile.
./mtest: Performing fill some data to tmpfile.
./mtest: Performing unlink tmpfile.
./mtest: Performing mmap tmpfile with MAP_SHARED.
./mtest: Performing lseek to TST_LEN+16 in tmpfile.
./mtest: Performing write to TST_LEN+16 in tmpfile.
./mtest: Performing lseek to TST_LEN in tmpfile.
./mtest: Performing read in tmpfile.
./mtest: Performing munmap tmpfile.
./mtest: Performing close tmpfile.
./mtest: test byte: 41; byte at pos TST_LEN+1: 41.
(base) [jon@archijon prog5]$ echo $?
0
```