ICS Homework 14

June 3, 2021

1 Organization

1.1

If we only use 0x400000-0x800000, compare the memory usage of page table between single level and 4 level.

Single: $\frac{2^{48}B}{4KB} * 8B = 2^{39}B = 512GB$

4 level: (2 L4 page table+3 L1/L2/L3 page table) * 4KB = 20KB

1.2

If we have enough physical memory and use the whole 2^{48}B virtual space, compare the memory usage again.

Single: $(2^{48}B/4KB) * 8B = 2^{39}B = 512GB$

4 level: $(1 + 512 + 512^2 + 512^3) * 4KB \approx 513GB$

1.3

Given the following page table and cr3=0x1000, please translate the virtual addresses to physical addresses. Assume the machine is x86_64, which uses 4 level page table.

Physical Address of PTE	Address Part in PTE
0x1000	0x4000
0x1008	0x4000
0x4000	0x5000
0x5000	0x7000
0x5008	0x8000
0x5010	0x9000
0x5018	0x3000
0x5020	0x5000
0x7008	0x8000
0x7010	0x10000

Physical Address	Virtual Address
0x1234	0x8234
0x2234	0x10234
0x8000802135	0x9135

1.4

Assume on a x86_64 machine without cache, we have an int `a[70]` at virtual address `0x8000344f00`, a program access `a[0], a[1], ..., a[69]` one by one. Before the first access, we have only an empty L1 page table.

1.4.1

How many physical page is used after the program access `a[0], a[64], a[69]`? (Remember the page table is stored in physical page too)

after access `a[0]`: 4 page table pages + 1 content pages
after access `a[64]`: 4 + 2
after access `a[69]`: 4 + 2

1.4.2

How many memory accesses and page fault happened in the program if we have no TLB?

(4+1)*70+1+4=355, 4+1 for each successful array access.

First access to `a[0]` need one access to L1 PTE to trigger page fault. First access to `a[64]` need four access to L1, L2, L3 and L4 PTE to trigger page fault. 2 page faults in total.

1.4.3

How many memory accesses and page fault happened in the program if we have a full associative infinite TLB?

70+4*2+1+4=83. 1 for each successful array access. Accesses to `a[0], a[64]` will trigger two page table walk and fill TLB.

2 page faults in total.

2 System Software

Consider a simple concurrent program with the following specification: The main thread creates two peer threads, passing each peer thread a unique integer *thread ID* (either 0 or 1), and then waits for each thread to terminate. Each peer thread prints its thread ID and then terminates.

Each of the following programs attempts to implement this specification. However, some are incorrect because they contain a race on the value of *myid* that makes it possible for one or more peer threads to print an incorrect thread ID. Except for the race, each program is otherwise correct.

You are to indicate whether or not each of the following programs contains such a race on the value of *myid*.

A. No, each thread has its own heap variable for myid.
B. Yes, both threads can point to myid.
C. No, myid is passed in on the stack.
D. Yes, the mutex doesn't protect myid.
E. No, the mutex protects the assignment of myid.

(A) Does the following program contain a race on the value of myid? (Yes/No)

```
1  void *foo(void *vargp) {
2    int myid;
3    myid = *((int *)vargp);
4    Free(vargp);
5    printf("Thread %d\n", myid);
6  }
7
8  int main() {
9    pthread_t tid[2];
10   int i, *ptr;
11   for (i = 0; i < 2; i++) {
12     ptr = Malloc(sizeof(int));
13     *ptr = i;
14     Pthread_create(&tid[i], 0, foo, ptr);
15   }
16   Pthread_join(tid[0], 0);
17   Pthread_join(tid[1], 0);
18 }
```

(B) Does the following program contain a race on the value of myid? (Yes/No)

```
1  void *foo(void *vargp) {
2    int myid;
3    myid = *((int *)vargp);
4    printf("Thread %d\n", myid);
5  }
6
7  int main() {
8    pthread_t tid[2];
9    int i;
10   for (i = 0; i < 2; i++)
11     Pthread_create(&tid[i], NULL, foo, &i);
12   Pthread_join(tid[0], NULL);
13   Pthread_join(tid[1], NULL);
14 }
```

(C) Does the following program contain a race on the value of myid? (Yes/No)

```
1  void *foo(void *vargp) {
2    int myid;
3    myid = (int)vargp;
4    printf("Thread %d\n", myid);
5  }
6
7  int main() {
8    pthread_t tid[2];
9    int i;
10   for (i = 0; i < 2; i++)
11     Pthread_create(&tid[i], 0, foo, i);
12   Pthread_join(tid[0], 0);
13   Pthread_join(tid[1], 0);
14 }
```

(D) Does the following program contain a race on the value of myid? (Yes/No)

```
1  sem_t s; /* semaphore s */
2  void *foo(void *vargp) {
3    int myid;
4    P(&s);
5    myid = *((int *)vargp);
6    V(&s);
7    printf("Thread %d\n", myid);
8  }
9
10 int main() {
11   pthread_t tid[2];
12   int i;
13   sem_init(&s, 0, 1); /* S=1 INITIALLY */
14   for (i = 0; i < 2; i++) {
15     Pthread_create(&tid[i], 0, foo, &i);
16   }
17   Pthread_join(tid[0], 0);
18   Pthread_join(tid[1], 0);
19 }
```

(E) Does the following program contain a race on the value of myid? (Yes/No)

```
1  sem_t s; /* semaphore s */
2  void *foo(void *vargp) {
```

```
3     int myid;
4     myid = *((int *)vargp);
5     V(&s);
6     printf("Thread %d\n", myid);
7   }
8
9   int main() {
10    pthread_t tid[2];
11    int i;
12    sem_init(&s, 0, 0); /* S=0 INITIALLY */
13    for (i = 0; i < 2; i++) {
14      Pthread_create(&tid[i], 0, foo, &i);
15      P(&s);
16    }
17    Pthread_join(tid[0], 0);
18    Pthread_join(tid[1], 0);
19  }
```