OSTEP Chapter 26

ECE 3600, Fall 2022

Table of Contents

1. Concurrency and Address Spaces

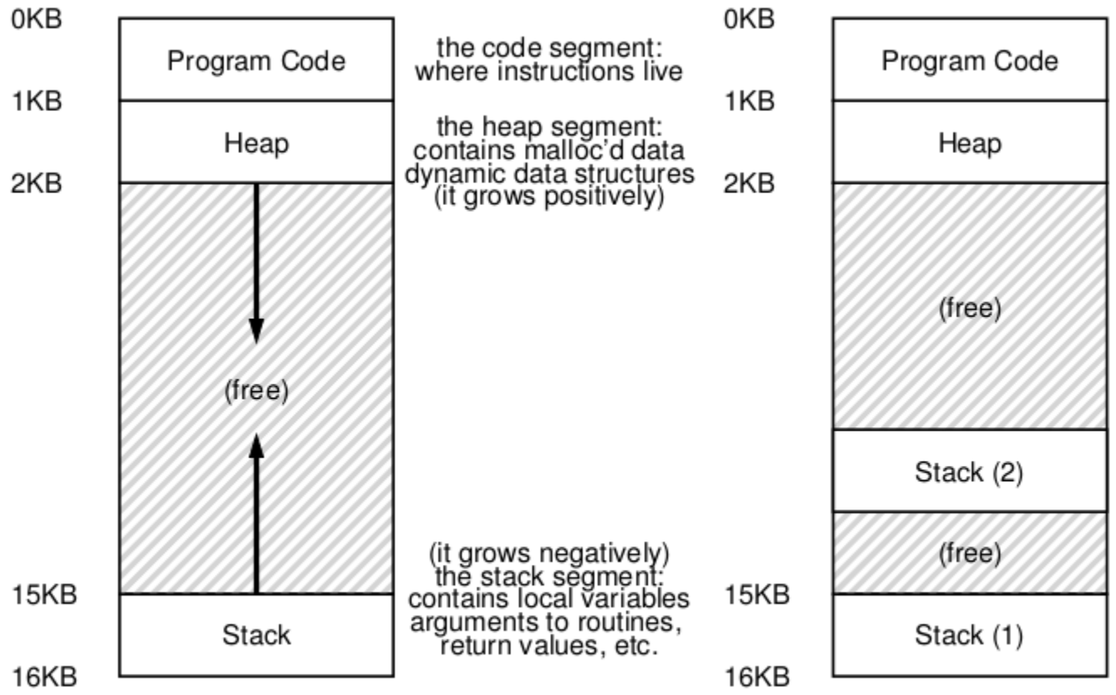

Figure 26.1: **Single-Threaded And Multi-Threaded Address Spaces**

2. Thread Scheduling

t0.c (updated)

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <pthread.h>
 4
 5  #include "common.h"
 6  #include "common_threads.h"
 7
 8  void *mythread(void *arg) {
 9      printf("%s\n", (char *) arg);
10      return NULL;
11  }
12
13  int main(int argc, char *argv[]) {
14      if (argc != 1) {
15          fprintf(stderr, "usage: main\n");
16          exit(1);
17      }
18
19      pthread_t p1, p2;
20      printf("main: begin\n");
21      printf( "main: creating T1\n");
22      Pthread_create(&p1, NULL, mythread, "A");
23      printf( "main: creating T2\n");
24      Pthread_create(&p2, NULL, mythread, "B");
25      // join waits for the threads to finish
26      printf( "main: waiting for T1\n");
27      Pthread_join(p1, NULL);
28      printf( "main: waiting for T2\n");
29      Pthread_join(p2, NULL);
30      printf("main: end\n");
31      return 0;
32  }
```

```
$ ./t0
main: begin
main: creating T1
main: creating T2
main: waiting for T1
A
B
main: waiting for T2
main: end

$ ./t0
main: begin
main: creating T1
main: creating T2
A
main: waiting for T1
main: waiting for T2
B
main: end

$ ./t0
main: begin
main: creating T1
main: creating T2
A
main: waiting for T1
B
main: waiting for T2
main: end
```

Figure 26.2: **Simple Thread Creation Code (t0.c)** (updated)

3. Thread Traces

main	Thread 1	Thread2
starts running		
prints “main: begin”		
creates Thread 1		
creates Thread 2		
waits for T1		
	runs	
	prints “A”	
	returns	
waits for T2		
		runs
		prints “B”
		returns
prints “main: end”		

Figure 26.3: **Thread Trace (1)**

main	Thread 1	Thread2
starts running		
prints “main: begin”		
creates Thread 1		
	runs	
	prints “A”	
	returns	
creates Thread 2		
		runs
		prints “B”
		returns
waits for T1		
returns immediately; T1 is done		
waits for T2		
returns immediately; T2 is done		
prints “main: end”		

Figure 26.4: **Thread Trace (2)**

main	Thread 1	Thread2
starts running		
prints “main: begin”		
creates Thread 1		
creates Thread 2		
		runs
		prints “B”
		returns
waits for T1		
	runs	
	prints “A”	
	returns	
waits for T2		
returns immediately; T2 is done		
prints “main: end”		

Figure 26.5: **Thread Trace (3)**

4. Updating Shared Data

t1.c (updated)

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#include "common.h"
#include "common_threads.h"

int max;
volatile int counter = 0; // shared global variable

void *mythread(void *arg) {
    char *letter = arg;
    int i; // stack (private per thread)
    printf("%s: begin [addr of i: %p]\n", letter, &i);
    for (i = 0; i < max; i++) {
        counter = counter + 1; // shared: only one
    }
    printf("%s: done\n", letter);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: main-first <loopcount>\n");
        exit(1);
    }
    max = atoi(argv[1]);

    pthread_t p1, p2;
    printf("main: begin [counter = %d] [%p]\n", counter,
           &counter);
    Pthread_create(&p1, NULL, mythread, "A");
    Pthread_create(&p2, NULL, mythread, "B");
    // join waits for the threads to finish
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("main: done\n [counter: %d]\n [should: %d]\n",
           counter, max*2);
    return 0;
}
```

```
$ ./t1 1000
main: begin [counter = 0] [0x5595c870602c]
A: begin [addr of i: 0x7f1d4b103edc]
A: done
B: begin [addr of i: 0x7f1d4a902edc]
B: done
main: done
 [counter: 2000]
 [should: 2000]
$ ./t1 10000
main: begin [counter = 0] [0x5614fcebb02c]
A: begin [addr of i: 0x7f2d6429eedc]
B: begin [addr of i: 0x7f2d63a9dedc]
A: done
B: done
main: done
 [counter: 16198]
 [should: 20000]
$ ./t1 10000
main: begin [counter = 0] [0x55e639cc602c]
A: begin [addr of i: 0x7fe89de71edc]
B: begin [addr of i: 0x7fe89d670edc]
A: done
B: done
main: done
 [counter: 13548]
 [should: 20000]
$ ./t1 10000
main: begin [counter = 0] [0x55bc1502402c]
A: begin [addr of i: 0x7f12e8365edc]
A: done
B: begin [addr of i: 0x7f12e7b64edc]
B: done
main: done
 [counter: 20000]
 [should: 20000]
$ ./t1 100000
main: begin [counter = 0] [0x563ec42ec02c]
A: begin [addr of i: 0x7ff3d02b7edc]
B: begin [addr of i: 0x7ff3cfab6edc]
A: done
B: done
main: done
 [counter: 102446]
 [should: 200000]
```

Figure 26.6: **Sharing Data: Uh Oh (t1.c)** (updated)

5. Data Race

$ **objdump -d t1 > t1.disasm.txt** # (note on %rip addressing)

```
...
 a4a:   8b 05 dc 15 20 00       mov    0x2015dc(%rip),%eax        # 20202c <counter>
 a50:   83 c0 01                add    $0x1,%eax
 a53:   89 05 d3 15 20 00       mov    %eax,0x2015d3(%rip)        # 20202c <counter>
...
```

Textbook example:

```
mov 0x8049a1c, %eax
add $0x1, %eax
mov %eax, 0x8049a1c
```

			(after instruction)		
OS	**Thread 1**	**Thread 2**	**PC**	**eax**	**counter**
	before critical section		100	0	50
	`mov 8049a1c,%eax`		105	**50**	50
	`add $0x1,%eax`		108	**51**	50
interrupt					
save T1					
restore T2			100	0	50
		`mov 8049a1c,%eax`	105	**50**	50
		`add $0x1,%eax`	108	**51**	50
		`mov %eax,8049a1c`	113	51	**51**
interrupt					
save T2					
restore T1			108	51	51
	`mov %eax,8049a1c`		113	51	**51**

Figure 26.7: **The Problem: Up Close and Personal**

Need mutual exclusion or atomic operations

6. Exercises

See the book for exercises using x86.py:

```
$ cat loop.s
.main
.top
sub  $1,%dx
test $0,%dx
jgte .top
halt

$ python ./x86.py -p loop.s -t 1 -i 100 -R dx

   dx          Thread 0
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt

$ python ./x86.py -p loop.s -t 1 -i 100 -R dx -c

   dx          Thread 0
    0
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
```

7. Q2

```
$ python ./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx -c

   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
    3   ----- Halt;Switch -----  ----- Halt;Switch -----
    2                            1000 sub  $1,%dx
    2                            1001 test $0,%dx
    2                            1002 jgte .top
    1                            1000 sub  $1,%dx
    1                            1001 test $0,%dx
    1                            1002 jgte .top
    0                            1000 sub  $1,%dx
    0                            1001 test $0,%dx
    0                            1002 jgte .top
   -1                            1000 sub  $1,%dx
   -1                            1001 test $0,%dx
   -1                            1002 jgte .top
   -1                            1003 halt
```

8. Q3

```
$ python ./x86.py -p loop.s -t 2 -a dx=3,dx=3 -R dx -c -i 5

   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    3   ------ Interrupt ------  ------ Interrupt ------
    2                           1000 sub  $1,%dx
    2                           1001 test $0,%dx
    2                           1002 jgte .top
    1                           1000 sub  $1,%dx
    1                           1001 test $0,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1                           1002 jgte .top
    0                           1000 sub  $1,%dx
    0                           1001 test $0,%dx
    0                           1002 jgte .top
   -1                           1000 sub  $1,%dx
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
   -1   ----- Halt;Switch -----  ----- Halt;Switch -----
   -1                           1001 test $0,%dx
   -1                           1002 jgte .top
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1                           1003 halt
```

9. Q4

```
$ cat looping-race-nolock.s
# assumes %bx has loop count in it

.main
.top
# critical section
mov 2000, %ax  # get 'value' at address 2000
add $1, %ax    # increment it
mov %ax, 2000  # store it back

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt

$ python ./x86.py -p looping-race-nolock.s -t 1 -a bx=1 -M 2000 -c

 2000          Thread 0
    0
    0   1000 mov 2000, %ax
    0   1001 add $1, %ax
    1   1002 mov %ax, 2000
    1   1003 sub  $1, %bx
    1   1004 test $0, %bx
    1   1005 jgt .top
    1   1006 halt

$ python ./x86.py -p looping-race-nolock.s -t 2 -a bx=1 -M 2000 -R bx -c

 2000      bx          Thread 0                Thread 1
    0       1
    0       1   1000 mov 2000, %ax
    0       1   1001 add $1, %ax
    1       1   1002 mov %ax, 2000
    1       0   1003 sub  $1, %bx
    1       0   1004 test $0, %bx
    1       0   1005 jgt .top
    1       0   1006 halt
    1       1   ----- Halt;Switch -----  ----- Halt;Switch -----
    1       1                           1000 mov 2000, %ax
    1       1                           1001 add $1, %ax
    2       1                           1002 mov %ax, 2000
    2       0                           1003 sub  $1, %bx
    2       0                           1004 test $0, %bx
    2       0                           1005 jgt .top
    2       0                           1006 halt
```

10. Q7

```
$ python ./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -R bx -c -i 1

 2000      bx          Thread 0                Thread 1

    0       1
    0       1   1000 mov 2000, %ax
    0       1   ------ Interrupt ------  ------ Interrupt ------
    0       1                           1000 mov 2000, %ax
    0       1   ------ Interrupt ------  ------ Interrupt ------
    0       1   1001 add $1, %ax
    0       1   ------ Interrupt ------  ------ Interrupt ------
    0       1                           1001 add $1, %ax
    0       1   ------ Interrupt ------  ------ Interrupt ------
    1       1   1002 mov %ax, 2000
    1       1   ------ Interrupt ------  ------ Interrupt ------
    1       1                           1002 mov %ax, 2000
    1       1   ------ Interrupt ------  ------ Interrupt ------
    1       0   1003 sub  $1, %bx
    1       1   ------ Interrupt ------  ------ Interrupt ------
    1       0                           1003 sub  $1, %bx
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0   1004 test $0, %bx
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0                           1004 test $0, %bx
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0   1005 jgt .top
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0                           1005 jgt .top
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0   1006 halt
    1       0   ----- Halt;Switch -----  ----- Halt;Switch -----
    1       0   ------ Interrupt ------  ------ Interrupt ------
    1       0                           1006 halt
```