cs154: Introduction to Computer Systems
Spring 2020

Homework 8
(Assigned May 25)
Due Jun 1 11:59pm

Submit your work by committing files in the `hw8` sub-directory of your `CNETID-cs154-spr-20` svn repository. The filename should be either `hw8.txt` or `hw8.pdf` for answers in plain ASCII text, or PDF, respectively. PDFs of scanned hand-written pages must not exceed 6 megabytes. Not following directions will result in losing points.

(1) (18 points)

```
1  #include "csapp.h"
2  #define N 2
3  void *thread(void *vargp);
4
5  char **ptr;  /* global variable */
6
7  int main()
8  {
9      int i;
10     pthread_t tid;
11     char *msgs[N] = {
12         "Hello from foo",
13         "Hello from bar"
14     };
15
16     ptr = msgs;
17     for (i = 0; i < N; i++)
18         Pthread_create(& tid, NULL, thread, (void *)&i);
19     Pthread_exit(NULL);
20 }
21
22 void *thread(void *vargp)
23 {
24     int myid = *((int*)vargp);
25     int cnt = 0;
26     printf("[%d]: %s (cnt=%d)\n", myid, ptr[myid], ++cnt);
27     return NULL;
28 }
```

This question refers to the above *modified* version of the code in textbook Figure 12.15. Modifications were made to lines 18, 24, and 25.

Using the analysis from Section 12.4, fill each entry in the following table with "Yes" or "No" for the code above. In the first column, the notation `v.t` denotes an instance of variable `v` residing on the local stack for thread `t`, where `t` is either `m` (main thread), `t0` (peer thread 0), or `t1` (peer thread 1). This is a modification of Practice Problem 12.6.A.

Variable instance	Referenced by main thread?	Referenced by peer thread 0?	Referenced by peer thread 1?
`cnt.t0`	_____	_____	_____
`cnt.t1`	_____	_____	_____
`i.m`	_____	_____	_____
`msgs.m`	_____	_____	_____
`myid.t0`	_____	_____	_____
`myid.t1`	_____	_____	_____

(2) (15 points)

Consider the following use of mutexes in two threads in light of the "Mutex lock ordering rule" from textbook Section 12.7.

```
Initially: a = 1, b = 1, c = 1, d = 1.
Thread 1:          Thread 2:
   P(a);              P(a);
   P(b);              V(a);
   V(b);              P(d);
   P(c);              P(c);
   V(c);              P(b);
   P(d);              V(b);
   V(d);              V(c);
   V(a);              V(d);
```

A. List all pairs of mutexes that Thread 1 can hold, and all pairs of mutexes that Thread 2 can hold.

B. Is there overlap between the sets of pairs of mutexes? If yes, are the mutexes (within the pair) locked in the same order?

C. Given your analysis, is there a potential for deadlock (Yes or No)?