CS241 Lecture 13
Working With pThreads. Introducing mutex locks

```
01  // downloads a web resource in the background
02  void* download(void*url) {
03    void* mem = malloc(2048);
04
05    ... cs241 network magic to download file
06
07   FILE* file = fopen(shortname,"w");
08   if(file&&bytes) fwrite(mem, bytes,1, file);
09   fclose(file);
10   free(mem);
11   return NULL;
12  }
13
14  int main() {
15   pthread_t tid1,tid2;
16   pthread_create(&tid1, NULL, download,
  "https://en.wikipedia.org/wiki/Spanish_dollar");
17    pthread_create(&tid2, NULL, download,
  "…1888_México_8_Reals_Trade_Coin_Silver.jpg");
18  // 2 ways to wait for threads to complete?
19
20
21
22
23
24
```

1. You call malloc from two threads?

Yes because it is "_____"

2. Why is it that *mem* will point to two different heap areas?

3. Which one of these is also safe to be used by two threads?
char *strerror(int errnum);
int strerror_r(int errnum, char *strerrbuf, size_t buflen);

4. Complete this code to print the thread id and an initial starting value. What does this code actually print? Why?

```
01  void* myfunc(void*ptr) {
02  printf("My thread id is %ld
            and I'm starting at %d\n",

  _____, _____);
03  return NULL;
04  }
05  int main() {
06  // Each thread needs a different value of i
07  pthread_t tid[10];
08  for(int i =0; i < 10; i++) {
09    pthread_create(& tid[i], 0, myfunc, &i);
10  }
11
12
```

5 What is a critical section?

6 What is a mutex?

7 What are the two ways to create a mutex?

8 How do you lock and unlock a mutex?

9 When can you destroy a mutex?

10. What does this code print? Will it always print the same output?

```
01  int counter;
02  void*myfunc2(void*param) {
03   int i=0; // stack variable
04   for(; i < 1000000;i++) counter ++;
05   return NULL;
06  }
07  int main() {
08   pthread_create(&tid1, 0, myfunc2, NULL);
09   pthread_create(&tid1, 0, myfunc2, NULL);
10   pthread_join(tid1,NULL);
11   pthread_join(tid2,NULL);
12   printf("%d\n", counter );
13  }
```

11 Common pattern: Use heap memory to pass starting information to each thread.
Example: Create two threads. Each thread will do half the work. The first thread will process 0..numitems/2 in the array. The second thread will process the remaining items. Any gotchas?

```
01  typedef struct work_ {
02
03
04  } work_t;

05  void calc (int * data, size_t nitems) {
06    size_t half = numitems/2;
07
08
09
10
11
12
13
14
15
16    pthread_create(&tid1, 0, imagecalc,____);
17  }
18  // Gotchas odd number of numitems. Memory leak?
```