

## CS241 Lecture 13

### Thread safety. Introducing mutex locks

#### 1. How do this code work? Finish *main()*

```
01 // downloads a web resource in the background
02 void* download(void*url) {
03     void* mem = malloc(2048);
04     size_t bytes = 0; // actual file size
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     return mem; // OR pthread_exit(mem);
11 }
12
13 int main() {
14     pthread_t tid1,tid2;
15     pthread_create(&tid1, NULL, download,
16 "https://en.wikipedia.org/wiki/Spanish_dollar");
17     pthread_create(&tid2, NULL, download,
18 "...1888_México_8_Reals_Trade_Coin_Silver.jpg");
19 // 2 ways to wait for threads to complete?
20
21
22
23
```

#### 2a. Can you call malloc from two threads?

Yes because it is "\_\_\_\_\_"

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

#### 2c Your question about threads?

#### 3. 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 %p
           and I'm starting at %d\n",
           (void*)_____, _____);
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
```

#### 4. What is a critical section?

#### 5. What is a mutex?

#### 6a. What are the two ways to create a pthread mutex?

#### 6b. How do you lock and unlock a mutex?

#### 6c. When can you destroy a mutex?

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

```
01  int sharedcounter;

02  void*myfunc2(void*param) {
03      int i=0; // stack variable
04      for(; i < 1000000;i++) sharedcounter ++;
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  }
```

8. 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 task_ {
02
03
04  } task_t;

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

9. Add mutex locks so *toTextMessage* can be called concurrently from two threads

```
01  static char message[200];
02  // char message[200];           // Option 2
03  int pleaseStop;
04
05  char* toTextMessage(char*to, char* from, int val) {
06  // static char message[200]; // Option 3
07  // char message[200];         // Option 4
08
09      sprintf(message,"To:%s From:%s:%d",to,from,val);
10      return message;
11  }
12
13  void* runner1(void* ptr) {
14      int count = 0;
15      while(!pleaseStop) {
16          char* mesg=toTextMessage("angrave","illinois",1);
17          printf("%d Sending %s\n", count ++, mesg);
18      }
19  }
20
21  void* runner2(void* ptr) {
22      while(!pleaseStop)
23          char* m=toTextMessage("Jurassic","Dinosaur",999);
24  }
25
26  int main() {
27      pthread_t tid1, tid2;
28      pthread_create(&tid1, 0, runner1, NULL);
29      sleep(2);
30      pthread_create(&tid2, 0, runner2, NULL);
31      sleep(5);
32      pleaseStop = 1;
33      pthread_join(tid1, NULL);
34      pthread_join(tid2, NULL);
35  }
```