CS341 Lecture 13
Thread safety. Introducing mutex locks

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

```
// downloads a web resource in the background
void* download(void*url) {
  void* mem = malloc(2048);
  size_t bytes = 0; // actual file size
  ... cs341 network magic to download file

 FILE* file = fopen(shortname,"w");
 if(file&&bytes) fwrite(mem, bytes,1, file);
 fclose(file);
 return mem; // OR pthread_exit(mem);
}

int main() {
 pthread_t tid1,tid2;
 pthread_create(&tid1, NULL, download,
"https://en.wikipedia.org/wiki/Spanish_dollar");
 pthread_create(&tid2, NULL, download,
"…1888_México_8_Reals_Trade_Coin_Silver.jpg");
// 2 ways to wait for threads to complete?
```

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?

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

 (void*)___________________, ___________________);
return NULL;
}
int main() {
// Each thread needs a different value of i
pthread_t tid[10];
for(int i =0; i < 10; i++) {
  pthread_create(& tid[i], 0, myfunc, &i);
}
```

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?

```
int sharedcounter;

void*myfunc2(void*param) {
 int i=0; // stack variable
 for(; i < 1000000;i++) sharedcounter ++;
 return NULL;
}
int main() {
 pthread_create(&tid1, 0, myfunc2, NULL);
 pthread_create(&tid1, 0, myfunc2, NULL);
 pthread_join(tid1,NULL);
 pthread_join(tid2,NULL);
 printf("%d\n", counter );
}
```

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?

```
typedef struct task_ {


} task_t;

void calc(int* data, size_t nitems) {
   size_t half = numitems/2;








  pthread_create(&tid1, 0, imagecalc,____);
}
// Gotchas: odd number of numitems. 2. Memory leak?
```

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

```
static char message[200];
// char message[200];          // Option 2
int pleaseStop;

char* toTextMessage(char*to, char* from, int val) {
// static  char message[200]; // Option 3
// char message[200];          // Option 4

   sprintf(message,"To:%s From:%s:%d",to,from,val);
   return message;
}

void* runner1(void* ptr) {
  int count = 0;
  while(!pleaseStop) {
     char* mesg=toTextMessage("angrave","illinois",1);
     printf("%d Sending %s\n", count ++, mesg);
  }
}

void* runner2(void* ptr) {
  while(!pleaseStop)
    char* m=toTextMessage("Jurassic","Dinosaur",999);
}

int main() {
  pthread_t tid1, tid2;
  pthread_create(&tid1, 0, runner1, NULL);
  sleep(2);
  pthread_create(&tid2, 0, runner2, NULL);
  sleep(5);
  pleaseStop = 1;
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
}
```