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?

pthread_join ( tid1, &result)
               tid 2
```

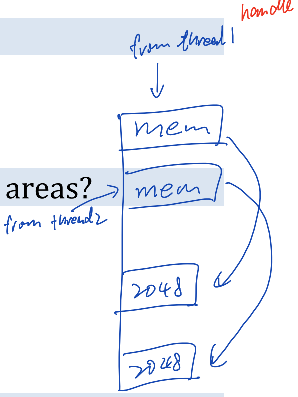
→ exit status

2a. Can you call malloc from two threads?

Yes because it is "thread safe"

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

mem is in different threads' stack, so safe



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?

starting_value [10]

```
void* myfunc(void*ptr) {
printf("My thread id is %p
 and I'm starting at %d\n",
 (void*) pthread_self() , *((int*) ptr) );
return NULL;
}
int main() {
// Each thread needs a different value of i
pthread_t tid[10];
for(int i =0; i < 10; i++) { starting_value[i] = 100+i
  pthread_create(& tid[i], 0, myfunc, &i);
} pthread_join(&tid[i], NULL);
```

returns the ID of the calling thread

directly reading this i in real time.

Not thread safe

change to & starting_value [i] → works!

solution (1)

solution (2)

wait until "myfunc" finish

solution (3) → in every loop,

4. What is a critical section?

segment of code where only 1 thread can be running (for expected behavior)

5. What is a mutex?

A duck! Before we access any data / do any change, we have to grab it first

It is a mutual exclusion. It has not access to the code

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

p_mutex_t duck = PTHREAD MUTEX INITIALIZER

more like a file handle

p_m_init(& duck)

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

P_m_lock (& duck)

P_m_unlock (& duck)

other threads got time frozen

6c. When can you destroy a mutex?

P_m_destroy(& duck)

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

```
int sharedcounter; → globle variable

void*myfunc2(void*param) {
 int i=0; // stack variable   pthred_mutex-lock(&dluk)
 for(; i < 1000000;i++) sharedcounter ++;
 return NULL;   pthred_mutex_unlock(&dluk)
}
int main() {
 pthread_create(&tid1, 0, myfunc2, NULL);
 pthread_create(&tid~~1,~~2 0, myfunc2, NULL);
 pthread_join(tid1,NULL);
 pthread_join(tid2,NULL);
 printf("%d\n", ~~counter~~ );
                sharedcounter
}
```

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);
}
```