CS241 #14
Working With threads and locks

0. Would you expect the following to work on your 64 bit VM?
(How about a 32bit machine?)

```
01  int bad = (int) "Hello";
02  puts( (char*) bad);
```

1 Which of the following calls will block?

pthread_mutex_init
pthread_mutex_lock
pthread_mutex_unlock
pthread_mutex_destroy

2 You call to *pthread_mutex_X* (what is X?) blocks. When will it return i.e. when will it unblock?

3 Why might pthread_mutex_X not block?

4. Where are the *critical sections* in the following two code examples?
Fix any errors you notice.
Modify the code to be thread safe

```
01  link_t* head;
02
03  void* list_prepend(int v) {
04    link_t* link = malloc( sizeof(link_t*));
05    link -> value = v;
06    link -> next = head;
07    head = link;
08  }
09
10  int list_remove_front() {
11     link_t link = head;
12     int v = link ->value;
13     head = link->next;
14     free(link);
15     return v;
16  }
```

4b. Meanwhile the code continued… (check for errors)

```
01  size_t capacity = 64;
02  size_t size = 0;
03  char** data = malloc(capacity);
04  void push(char*value) {
05    if(size == capacity) {
06      capacity *= 2;
07      realloc(data,capacity);
08    }
09    data[size++] = value;
10  }
11  char* pop() {
12     char* result = data[--size];
13     return result;
14  }
```

5. Lock Contention and likelihood of discovering race conditios
A thread at a random time executes for 1ms code inside an unprotected critical section with 1s total running time. If there are now 2 threads that run for 1second each, estimate the probability of both threads in the critical section at the same time.

6. Remember me?Notice any mistakes? What will happen exactly?

```
01  pthread_t tid1,tid2;
02  pthread_mutex_t m;
03
04  int counter;
05
06  void* myfunc2(void*param) {
07   int i = 0; // stack variable
08   for(; i < 1000000;i++) {
09     pthread_mutex_lock( &m );
10     counter ++;
11   }
12   return NULL;
13  }
14  int main() {
15   pthread_create(&tid1, 0, myfunc2, NULL);
16   pthread_create(&tid2, 0, myfunc2, NULL);
17   pthread_join(tid1,NULL);
18   pthread_join(tid2,NULL);
19   printf("%d\n", counter );
20  }
```

7. Case study1: Critical Sections and functions that are not thread safe

```
01  static FILE* file;
02
03  void logerror(int errnum, char*mesg) {
04    char* error = strerror(errnum);
05    if(!file) {
06      file = fopen("errorlog.txt","a+");
07    }
08    fprintf(file,"%s:%s", mesg, error);
09    fflush(file)
10  }
```

8. Meet your next *Synchronization Primitive*: What is a *Counting Semaphore*?

9. Case study2: Parallelize *AngraveCoin* miner for fun and profit!

```
void search(long start, long end) {
  printf("Searching from 0x%lx to 0x%lx\n", start , end);
  for(long i = start; i < end; i++) {
    char message[100];
    sprintf(message,"AngraveCoin:%lx", i);

    unsigned char *hash; // 256 bit result ( = 32 bytes )

    hash = SHA256(message, strlen(message), NULL);

    int iscoin; // first three bytes must be zero
    iscoin =(hash[0]==0)&&(hash[1]==0)&&(hash[2]==0);

    if(iscoin)
        printf("%lx %02x %02x %02x '%s'\n",
                i, hash[0],hash[1],hash[2] , message);
  }
  printf("Finished %lx to %lx\n", start, end);
}

// I want to speed up search of 2^33 possible coins
long array[] = {0L, 1L <<25, 1L <<27, 1L <<33};
int main() {
  search(array[0], array[1]);
  search(array[1], array[2]);
  search(array[2], array[3]);
  return 0;
}
```