CPE 442 : Sobel Filter

Author(s)

- Victor Delaplaine
- Tristin Chutka

Video link(s)

- Video demonstrating the sobel program with pthreads
- Video demonstrating the diffence in time between tutorial 2 and 3

Tutorial about pthreads (#include <pthread.h> and in)

- We chose to have four cores work on each seperate frames in this case
- If you are using pthreads make sure you include: '#include <pthread.h>' and -pthread in your C_FLAGS

Using pthread_create

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine) (void *), void *arg);
```

- This function above is used to create a pthread
- Each thread has its own code of execution aka a function of the format `void *(start_routine)(void *)`
  - This is a function pointer meaning if your thread function has the same format then you can simply pass the name in
- `void *arg` is the argument that you want to pass into your thread_func
  - If you want to pass multiple things into this thread function you should make a structure to hold all the members and pass the address of it to the void *arg.
- `pthread_t *thread` is the address to the thread you want to run
- `const pthread_attr_t *attr` are attributes you want to give to your thread on creation (such as priority or scheduling policy)

Using pthread_join

```
int pthread_join(pthread_t thread, void **retval)
```

- This function above waits blocks until the `thread` given in the parameter is terminated
- `retval` is the return value of you thread function
  - To return a value in the function use `pthread_exit(<address of the value you want to return>)`

What is a mutext

Since threads share some parts of memory, its most like they will try to access the same part of memory(i.e a variable) at the same time. This can lead to a race condition - unkown things can happen to the data. Because of this its very important that you block access to that region of memory, you can do this by using a mutext. In the pthread API there are two important function to be used to keep that data secure from each other thread.

1.)`pthread_mutext_lock(pthread_mutext_t *mux)`

2.)`pthread_mutext_unlock(pthread_mutext_t *mux)`

Before entering a critical region (aka accessing shared memory between thread) call `pthread_mutext_lock`, then do the necessary work and then when your done manupulating the data call `pthread_mutext_unlock`, so other threads can use this data. For other threads trying to accessing the data in the critical region, they will be blocked and woken up when the region is unlocked.