README Report

Homework 0: User Level Threads

By: Augustus Chang & Anushiya Balakrishnan

Introduction:

To approach this project, we identified what user level threads require.

They require:

1. A *scheduler* to manage them
2. A data structure to describe thread attributes (to be used by the thread scheduler)

The thread scheduler is built into the function :

- **mypthread_yield**
- **mypthread_exit**
- **mypthread_join**

It is worth noting that **mypthread_exit** and **mypthread_join** are special cases of **mypthread_yield**.

We need the data structure to manage the threads **mypthread_info_t**

```
typedef struct {
    int pid;                // unique thread id
    state status;           // current status of thread
    ucontext_t mycontext;   // context associated with thread

    // so that if a thread is joinable, its calling thread is unblocked
    int wait_pid;           // waiting (if any) thread : -1 OR pid
    void * retval;          // pointer to return value


} mypthread_info_t;
```

We then declared (and initialized upon the creation of the first thread) a table of **mypthread_info_t** structs.

```
mypthread_info_t table[512];
```

Moving onto the **mypthread_t** itself, it is rather minimalistic. It only requires a thread id (which is incremented for every new thread via a global variable).

```
// Types
    typedef struct {
    // Define any fields you might need inside here.
    int pid;    // unique thread id
} mypthread_t;
```

We established 5 states for a thread. They are detailed below.

```
typedef enum {
    untouched,  // thread has not been created
    ready,      // thread has been created
    running,    // thread is active (only one thread is active at a time)
    blocked,    // thread is blocked and waiting . . .
    zombie,     // thread has finished executing
} state;
```

Some edge cases we considered were

- Using the mypthread.h library when no threads have been created yet created (this was solved by checking the current threads id number)
- A thread joining on itself (this is illegal because the thread will be blocked forever)
- Mutiple joins on the same thread (undefined behavior, we learned from the actual pthread library)

Our main challenges were

- The thread switching policy for joining
- Understanding return values from mypthread_exit() and implementing the return values

Thanks!