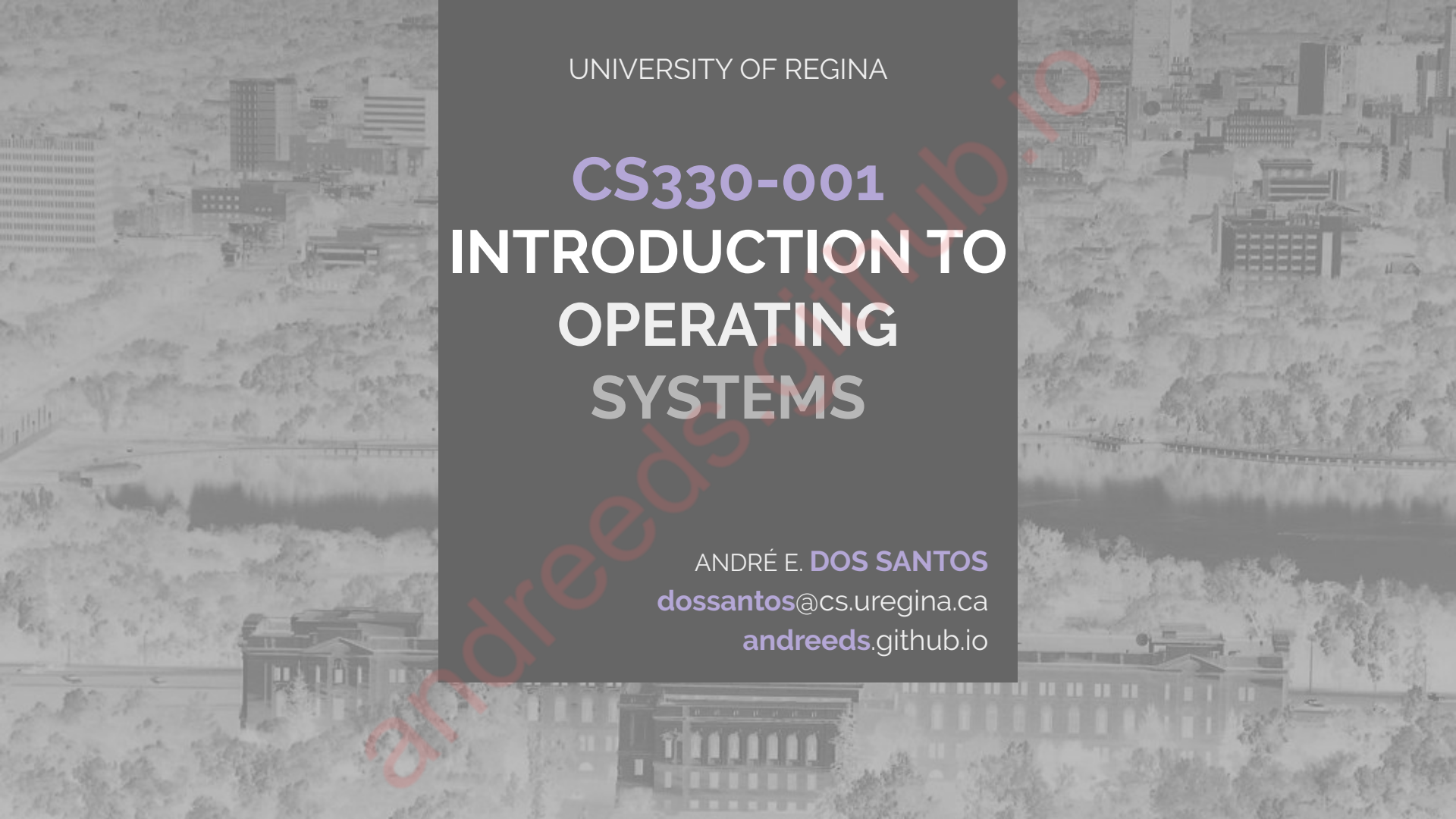

UNIVERSITY OF REGINA

CS330-001
INTRODUCTION TO OPERATING SYSTEMS

ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreeds.github.io

CS330-001
INTRODUCTION TO
OPERATING SYSTEMS

PROGRAMMING THREADS
C

ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreeds.github.io

PROGRAMMING THREADS

- All of the thread system calls require a thread variable as one of the arguments

```
#include <pthread.h>
pthread_t *tid;
```

ELEMENTARY THREAD SYSTEM CALLS

The `pthread_create` system call creates a new thread

```
#include <pthread.h>
int pthread_create( pthread_t *tid,
                    pthread_attr_t * attr_t,
                    void *(* start_routine) (void *),
                    void *arg );
```

- A newly created thread is immediately runnable
- `tid` points to the **thread ID** of the newly created thread
- `attr_t` represents an object that encapsulates the **attributes of a thread**
  - If `NULL`, the new thread is described by **default attribute values** regarding the placement and size of the thread stack and thread scheduling level
- `start_routine` is the **name of a function** that the thread will call when it begins executing.
  - It returns a pointer to void
- `arg` is the single parameter to `start_routine` and is a pointer to void

- If **successful**, `pthread_create` returns **0**
- If **unsuccessful**, returns a **non-zero error code**

ELEMENTARY THREAD SYSTEM CALLS

The `pthread_join` system call causes the creating program to wait for the specified thread to exit

```
#include <pthread.h>
int pthread_join (pthread_t thread, void **value_ptr);
```

- `thread` is the **thread** that will be **joined** and is called the **target thread**
- A call to `pthread_join` causes the calling thread to block until the target thread terminates.
- `value_ptr`, is a pointer to the location for the return status of the target thread
  - If `NULL`, the calling thread does not retrieve the return status of the target thread

- If **successful**, `pthread_join` returns **0**
- If **unsuccessful**, `pthread_join` returns a **non-zero error code**

ELEMENTARY THREAD SYSTEM CALLS

pthread1.c

URCourses

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int pthread_join (pthread_t thread, void
**value_ptr);

void *print_message_function( void *ptr
);

main()
{
    pthread_t thread1, thread2;
```

THREAD SYNCHRONIZATION

- The threads library provides three synchronization mechanisms:
  - **mutexes** - Mutual exclusion lock
    - Block access to variables by other threads
    - This enforces exclusive access by a thread to a variable or set of variables
  - **joins** - Make a thread wait till others are complete (terminated)
  - **condition variables** - data type `pthread_cond_t`

MUTEXES

- Mutexes are used **to prevent data inconsistencies** due to race conditions
- A race condition often occurs when **two or more threads need to perform operations on the same memory area**, but the results of computations depends on the order in which these operations are performed
- Mutexes are used for serializing shared resources
- Mutexes can be applied only to threads in a single process and do not work between processes as do **semaphores**

Without Mutex

```
int counter=0;

/* Function C */
void functionC()
{

  counter++

}
```

Thread 1	Thread 2
counter = 0	counter = 0
counter = 1	counter = 1

With Mutex

```
/* Note scope of variable and mutex are the
same */
pthread_mutex_t mutex1 =
PTHREAD_MUTEX_INITIALIZER;
int counter=0;

/* Function C */
void functionC()
{
  pthread_mutex_lock( &mutex1 );
  counter++
  pthread_mutex_unlock( &mutex1 );
}
```

Thread 1	Thread 2
counter = 0	counter = 0
counter = 1	Thread 2 locked out. Thread 1 has exclusive use of variable counter
	counter = 2

ELEMENTARY THREAD SYSTEM CALLS

mutex1.c

URCourses

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *functionC();
pthread_mutex_t mutex1 =
PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;
```

```
ad2;
```

When a mutex lock is attempted against a mutex which is held by another thread, the thread is blocked until the mutex is unlocked. When a thread terminates, the mutex does not unless explicitly unlocked.
Nothing happens by default.

ELEMENTARY THREAD SYSTEM CALLS

join1.c

URCourses

```
#include <stdio.h>
#include <pthread.h>

#define NTHREADS 10
void *thread_function(void *);
pthread_mutex_t mutex1 =
PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;

main()
{
   pthread_t thread_id[NTHREADS];
   int i, j;
```

CONDITION VARIABLES

- A condition variable is a variable of type `pthread_cond_t` and is used with the appropriate functions for waiting and later, process continuation
- The condition variable mechanism allows threads to suspend execution and relinquish the processor until some condition is true
- **A condition variable must always be associated with a mutex** to avoid a race condition created by one thread preparing to wait and another thread which may signal the condition before the first thread actually waits on it resulting in a deadlock
- The thread will be perpetually waiting for a signal that is never sent
- Any mutex can be used, there is no explicit link between the mutex and the condition variable

CONDITION VARIABLES

Creating/Destroying:

- `pthread_cond_init`
- `pthread_cond_t cond = PTHREAD_COND_INITIALIZER;`
- `Pthread_cond_destroy`

Waiting on condition:

- `pthread_cond_wait`
- `pthread_cond_timedwait` - place limit on how long it will block.

Waking thread based on condition:

- `pthread_cond_signal`
- `pthread_cond_broadcast` - wake up all threads blocked by the specified condition variable

cond1.c

URCourses

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>


/* Compile like this:
 gcc --std=c99 -lpthread cond.c -o cond
*/

const size_t NUMTHREADS = 20;
```

CONDITION VARIABLES

- An example of using `pthread`'s condition variables, showing how to block a main thread while waiting for worker threads to finish their work, without joining the threads
- This could be useful if you want to loop the threads, for example