Project 2A

Alexandre Tiard

10/26/18

What is this project about?

Update a shared variable

- A simple integer in part 1
- A list in part 2

To demonstrate the existence of race conditions

...and how to avoid them

What is this project about?

Update a shared variable

- A simple integer in part 1
- A list in part 2

To demonstrate the existence of race conditions

...and how to avoid them

```
void add(long long *pointer, long long value) {
        long long sum = *pointer + value;
        *pointer = sum;
}
```

Multi threaded applications

Initially, your main() program comprises a single thread

- All other threads will have to be explicitly created
  - Done using pthread_create()
- Upon exit, wait on all other threads to complete
  - Using pthread_join
- Time each run for each thread
  - Using clock_gettime()
- Optionally providing protection from race conditions
  - Mutexes, spin locks, compare and swap adds
- Optionally creating conflicts
  - When adding, or when modifying the linked list

Measuring run times

Success: 0 // Error: errno

int clock_gettime(clockid_t *clk_id*, struct timespec **tp*);

The specified clock:

- **CLOCK_REALTIME**
- **CLOCK_MONOTONIC**
- **CLOCK_PROCESS_CPUTIME_ID**
- **CLOCK_THREAD_CPUTIME_ID**

```
struct timespec {
        time_t   tv_sec;        /* seconds */
        long     tv_nsec;       /* nanoseconds */
};
```

Rmk: A clock may be system wide or per-process/per thread. Which do we want?
-> A system wide clock

pthread_create

Goal: Create a thread

Attribute object, set to 'NULL' for default values. These include:

- Detached or joinable state
- Scheduling inheritance
- Scheduling policy
- Scheduling parameters
- Scheduling contention scope
- Stack size
- Stack address
- Stack guard (overflow) size

Success: 0 // Error : errno, *thread left undefined

Opaque identifier returned by the routine

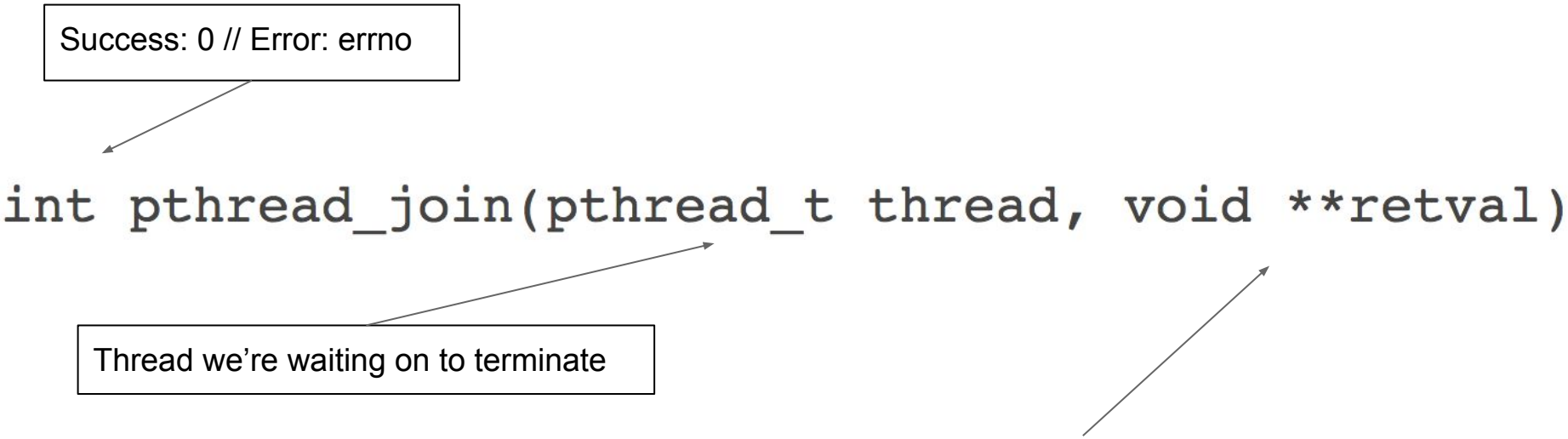
```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

The C routine to be executed on creation

A single argument to be passed to start routine:
Passed by reference as a pointer cast of type void.
Can be set to ''NULL' .

pthread_join()

Goal: Join with a terminated thread



Initializing a mutex

2 ways:

- Statically , when declared :

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
```

- Dynamically, which permits to set attributes:

Success: 0 // Error : errno

```
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
const pthread_mutexattr_t *restrict attr);
```

Mutex object

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Rmk: Attempting to destroy a locked mutex results in a ‘busy’ error

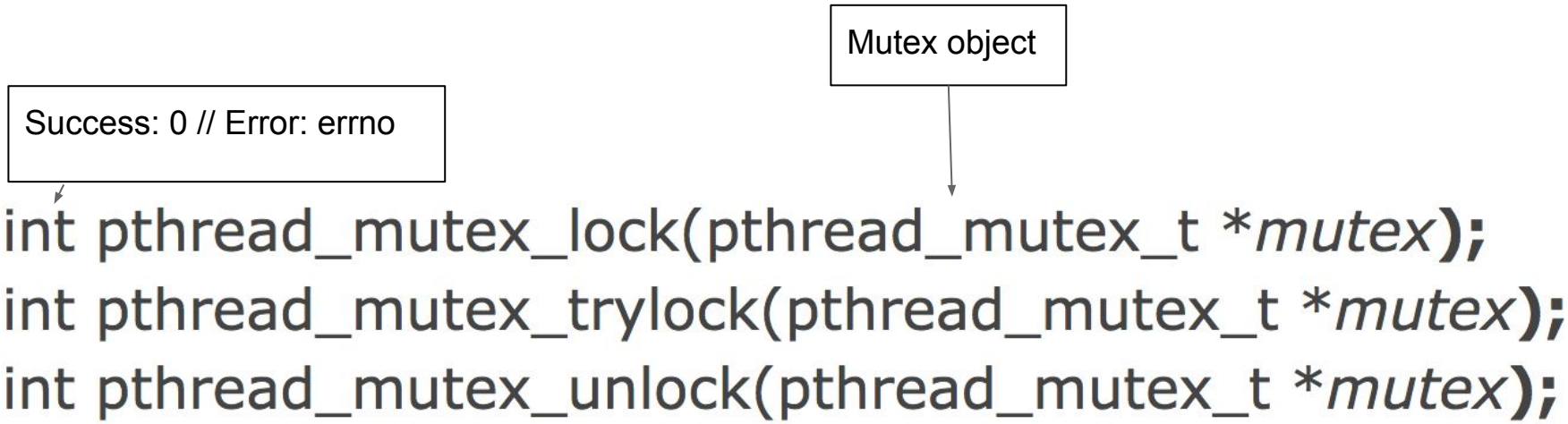
Setting attributes

Mutex attributes include:

- The type (deadlocking, deadlock-detecting, recursive...)
- The robustness (if you acquire a mutex and original owner died while processing it)
- Process-shared attributes (sharing a mutex across process boundaries)
- Protocol (how a thread behaves when higher priority thread wants the mutex)
- Priority ceiling (of the critical section, can prevent priority inversion)

Rmk: you need to call init/destroy to create the ‘attributes’ object

Locking and Unlocking a Mutex



Sync lock test and set

Goal: Implement a spin lock

Writes ‘value’ into *ptr, and returns the previous contents of *ptr

```
type __sync_lock_test_and_set (type *ptr, type value, ...)
```

```
__sync_lock_release (type *ptr, ...)
```

Releases lock pointed at by ptr

Compare and swap

Goal: Atomic compare and swap, if the current value of *ptr is oldval, then write newval into *ptr

True if comparison is successful and newval is written

```
bool __sync_bool_compare_and_swap (type *ptr, type oldval type newval, ...)
type __sync_val_compare_and_swap (type *ptr, type oldval type newval, ...)
```

Contents of *ptr before the operation

Variable to modify