pthreads

Eric Missimer

What are threads?

- A thread is a flow of execution within an address space
- When a process starts it already has one thread
- We can create more threads
- Use the `clone` system call (remember that it's what `fork` actually calls)

Creating a thread

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)`
- thread – will be initialized by the call to `pthread_create`
- attr – can control various thread attributes, e.g. stack size
- start_routine – function pointer to the "main" function for the thread
- arg – argument passed to "main" function

When the thread is done

- `void pthread_exit(void *retval)`
    - terminates the thread
    - retval – passed to any other thread that calls `pthread_join`
- `int pthread_join(pthread_t thread, void **retval)`
    - Waits for the thread specified to `thread` to terminate
    - If `retval` is not NULL, then the exit value for the thread will be copied to `*retval`

When the thread is done

- `void pthread_exit(void *retval)`
  - terminates the thread
  - retval – passed to any other thread that calls `pthread_join`
- `int pthread_join(pthread_t thread, void **retval)`
  - Waits for the thread specified to `thread` to terminate
  - If `retval` is not NULL, then the exit value for the thread will be copied to `*retval`
- What are the process equivalent functions?

When the thread is done

- `void pthread_exit(void *retval)`
    - terminates the thread
    - retval – passed to any other thread that calls `pthread_join`
- `int pthread_join(pthread_t thread, void **retval)`
    - Waits for the thread specified to `thread` to terminate
    - If `retval` is not NULL, then the exit value for the thread will be copied to `*retval`
- What are the process equivalent functions?
- `pthread_exit` – `exit`
- `pthead_join` – `waitpid`
- `pthread_simple` example

Locking

- `pthread_race_condition` example

Locking

- `pthread_race_condition` example
- What's going on?

Locking

- `pthread_race_condition` example
- What's going on?
- Locks can help use solve the problem
- `int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr)`
- `int pthread_mutex_lock(pthread_mutex_t *mutex)`
- `int pthread_mutex_trylock(pthread_mutex_t *mutex)`
- `int pthread_mutex_unlock(pthread_mutex_t *mutex)`

Locking cont.

- When a lock is held by one thread we are guaranteed no other thread holds the lock

Locking cont.

- When a lock is held by one thread we are guaranteed no other thread holds the lock
- Locks give us more than that in terms of synchronization
- Compiler barrier
- Memory barrier

Waiting for something

- Lets say you have a lock but some condition to move forward isn't true, e.g. the buffer is item so no item to grab
- What should you do?

Waiting for something

- Lets say you have a lock but some condition to move forward isn't true, e.g. the buffer is item so no item to grab
- What should you do?
- We want to release the lock, be notified when the condition is now true and when we wake up we have the lock

Waiting for something

- Lets say you have a lock but some condition to move forward isn't true, e.g. the buffer is item so no item to grab
- What should you do?
- We want to release the lock, be notified when the condition is now true and when we wake up we have the lock
- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)`

Waiting for something

- Lets say you have a lock but some condition to move forward isn't true, e.g. the buffer is item so no item to grab
- What should you do?
- We want to release the lock, be notified when the condition is now true and when we wake up we have the lock
- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)`
- `int pthread_cond_signal(pthread_cond_t *cond)` – unblocks at least one
- `int pthread_cond_broadcast(pthread_cond_t *cond)` – unblocks all