Pthread Synchronization Patterns

Alexandre David
1.2.05

Creating + Running a Thread

Main program	Thread function
pthread_create	
…	…
pthread_join	pthread_exit

Mutex-Lock

Main program	Thread
pthread_mutex_init	pthread_mutex_lock
…	<critical section
	as small as possible>
pthread_mutex_destroy	pthread_mutex_unlock

Mutex-Try-Lock

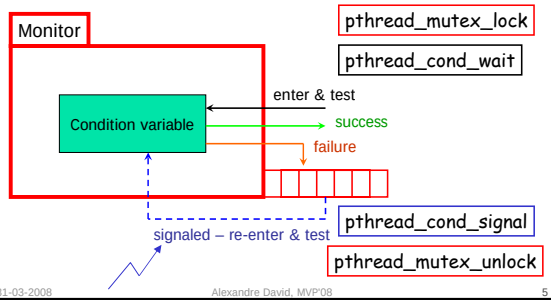
Main program

```
pthread_mutex_init
...
pthread_mutex_destroy
```

Thread

```
if pthread_mutex_trylock
  <pospone work
  do something else>
else
  <critical section
  as small as possible>
  pthread_mutex_unlock
end
```


Condition Variables & Monitors



Condition Variables

Main program

```
pthread_mutex_init
pthread_cond_init
...
pthread_mutex_destroy
pthread_cond_destroy
```

Thread: Wait for a condition.

```
pthread_mutex_lock
while !condition do
  pthread_cond_wait
done
<critical section
as small as possible>
pthread_mutex_unlock
```

Thread:
Make a condition become true.

```
pthread_mutex_lock
<critical section
as small as possible>
pthread_cond_signal
pthread_mutex_unlock
```

Spin-Locks

- Mutex:
  - Threads block until the lock is acquired.
  - Blocked threads are idle and need to wake up.
- Spin-locks:
  - Threads spin until the lock is acquired.
  - Blocked threads are not idle!
  - Better for quick access of **small** critical sections with low contention.

Pthread Spin Locks

- Calls:
  - pthread_spin_init(pthread_spinlock_t*, int)
    pthread_spin_destroy(pthread_spinlock_t*)
  - pthread_spin_lock(pthread_spinlock_t*)
    pthread_spin_trylock(pthread_spinlock_t*)
    pthread_spin_unlock(pthead_spinlock_t*)
- **Not** related to condition variables because threads do not wait and are not woken up!

Semaphores (see PSS)

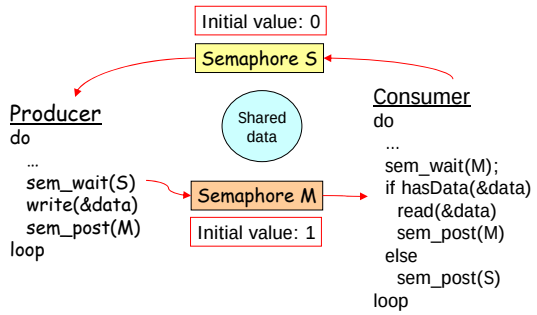
- Special counter
  - inc & dec atomic
  - no access to its value
  - wait for counter > 0 & dec
  - inc & signal a blocked thread/process.
- Initial counter
  - if == 0, useful for synchronizing.
  - if == *n (> 0)*, useful for allowing at most *n* threads/processes in a critical section.

Semaphores

#include <semaphore.h>

- Calls
  - sem_init(semt_t*, int, unsigned int value)
    sem_destroy(sem_t*)
  - sem_wait(sem_t*)
    sem_trywait(sem_t*)
    sem_timedwait(sem_t*, const struct timespec*)
  - sem_post(sem_t*)

Producer-Consumers



Application to our Model-Checker

- Spin-locks on queues.
- Semaphores for synchronization:
  - producer-consumer scheme!
  - Careful with races to detect termination.
    - Atomic: modify the queue & keep track of blocked threads with a flag.

Advanced: Futex

- Futex: Fast userspace locking system call.
  - Wait for a value at a given address to change.
  - Wake up anyone waiting on an address.
  - Low-level call usually used to implement locks.
  - Minix specific, available under Linux.