Condition variables revisited

Pamela Delgado

April 3, 2019

based on:
- W. Zwaenepoel slides
- Arpaci-Dusseau book

Multithreaded Web Server
Working solution

```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        avail++
        Pthread_cond_signal( notempty )
        Pthread_mutex_unlock( queuelock )
    } }
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        while( avail <= 0 ) Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        avail--
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    } }
```

Recap Pthreads: Condition Variables

- Pthread_cond_wait(cond, mutex)

- Pthread_cond_signal(cond, ~~mutex~~)
- Pthread_cond_broadcast(cond , ~~mutex~~)
  - Mutex not really needed, easier to explain

- Must hold mutex when calling any of these!
  - Not strictly needed for signal/broadcast, but safe

Wait: particularities of implementation

- assumes that mutex is locked (to caller thread) when its called

a. (atomically*) release lock and put caller thread to sleep;
b. when signaled, re-acquire lock and return to caller thread

* It means no possible interleaving

Signal: particularities of implementation

- Thread might **not immediately** acquire lock
- More (not necessary for this class)
  - unblocks *at least* one of the threads that are blocked on the specified *cond**
  - scheduling policy determines the order in which threads are unblocked*

* source: pthreads man page

Multithreaded Web Server
Broken solution 1

```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        Pthread_cond_signal( notempty )
        Pthread_mutex_unlock( queuelock )
    }
}
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    }
}
```

Broken solution 1 – problem

1. All worker threads busy (none waiting)
2. Listener does a signal
3. No thread waiting: signal is no-op
4. Worker thread finishes what it was doing
   - Will do a wait
   - Although request is waiting in queue

Thread trace – broken solution 1
Workers wait after Listener signals

Listener	Waiting for request	Add request X to queue	**SIGNAL** waiting thread	Waiting for request
Queue	Empty	Full	Full	Full
Worker 1	Processing request A	Processing request A	Processing request A	**Waiting on signal**
Worker2	Processing request B	Processing request B	Processing request B	**Waiting on signal**

time

No op ✗

In General

- Signals have no memory
- Forgotten if no thread waiting
- So need an extra variable to remember them

Multithreaded Web Server
Broken solution 2

```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        avail++
        Pthread_cond_signal( notempty, queuelock)
        Pthread_mutex_unlock( queuelock )
    } }
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        if( avail <= 0 ) Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        avail--
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    } }
```

Thread trace – broken solution 2
Worker wont wait after Listener signals

Listener	Waiting for request	Add request X to queue & avail++	**SIGNAL** waiting thread	Waiting for request
Queue	Empty	Full	Full	**Full**
Avail	0	1	1	1
Worker 1	Processing request A	Processing request A	Processing request A	**Check avail, wont wait**

time

Note

- Should now be clear why mutex must be held
- Avail is a shared data item
- Without mutex could have data race

Multithreaded Web Server
No locks

```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        avail++
        Pthread_cond_signal( notempty, queuelock)
        Pthread_mutex_unlock( queuelock )
    } }
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        if( avail <= 0 ) Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        avail--
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    } }
```

No locks solution - problem

1. Worker checks avail and finds it to be 0
2. Worker interrupted **(by OS)** and listener runs
3. Listener sets avail to 1 and signals
4. No thread is waiting, so signal is no-op
5. Listener interrupted **(by OS)** and worker runs
6. Worker does a wait

Incorrect: worker waits with request in queue

Thread trace – No locks solution

Listener	Waiting for request	**SIGNAL** waiting thread	Waiting for request
Queue Avail	Empty 0	Full 1	Full 1
Worker 1	Check avail, **Interrupted before wait**	Interrupted before wait	**Continue to wait**

No op ✗

time

Back to Solution With Locks

```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        avail++
        Pthread_cond_signal( notempty )
        Pthread_mutex_unlock( queuelock )
    } }
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        if( avail <= 0 ) Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        avail--
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    } }
```

Still not quite correct

1. Queue is empty, Worker1 waits
2. Listener puts request in queue
   – Sets avail to 1
   – Signals
   – Worker1 is unblocked
3. Worker2 runs, takes something out of queue
   – Sets avail to 0
4. Now Worker1 runs
   – It must check the value of avail

Thread trace – broken solution 2
Worker wants to process empty queue

Listener	Waiting for request	**SIGNAL** waiting thread	Waiting for request	Waiting for request
Queue	Empty	Full	Full	**Empty**
Avail	0	1	1	**0**
Worker 1	Wait	Unblocked	Unblocked	**Take from queue?!**
Worker2			**Check avail, process request**	Waiting for signal

time

Multithreaded Web Server
Working solution

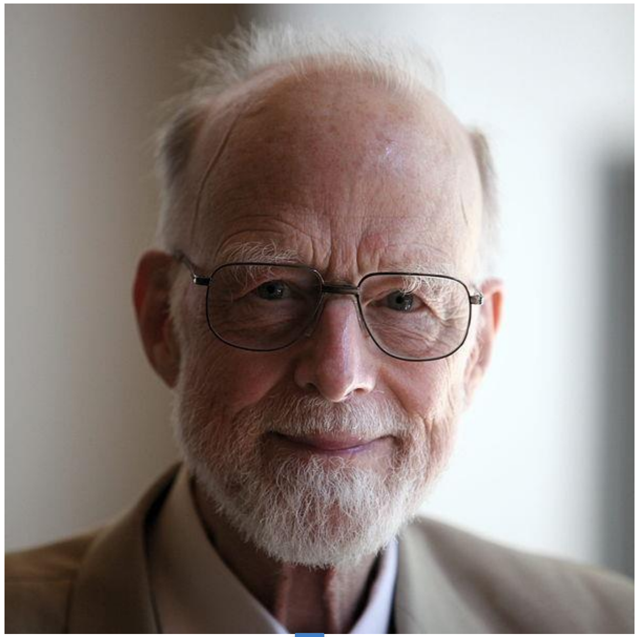
```
ListenerThread {
    for( i=0; i<MAX_THREADS; i++ ) thread[i] = Pthread_create(…)
    forever {
        Receive( request )
        Pthread_mutex_lock( queuelock )
        put request in queue
        avail++
        Pthread_cond_signal( notempty )
        Pthread_mutex_unlock( queuelock )
    } }
WorkerThread {
    forever {
        Pthread_mutex_lock( queuelock )
        while( avail <= 0 ) Pthread_cond_wait( notempty, queuelock )
        take request out of queue
        avail--
        Pthread_mutex_unlock( queuelock )
        read file from disk
        Send( reply )
    } }
```

Mesa semantics

Signaling only wakes a thread up
NO GUARANTEE that when the thread runs the state will be the same

Want stronger guarantees?

Hoare semantics (older than Mesa semantics)



Also author of Quicksort!