ECE 574 – Cluster Computing
Lecture 8

Vince Weaver

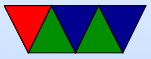
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

14 February 2019

Announcements

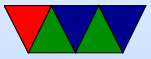
- Homework #4 (pthreads) will be posted
- HW#1 + HW#2 labs were sent out



Pthread Programming

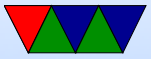
Useful links:

- `https://computing.llnl.gov/tutorials/pthreads/`
- `http://www.cs.cf.ac.uk/Dave/C/node31.html`



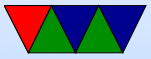
Creating Threads

- Your initial process, as per normal, only includes one thread
- `pthread_create()` creates a new thread
- You can call it anywhere, as many times as you want
- pthread_create (thread,attr,start_routine,arg)
- You pass is a pointer to a thread object (pthread_t) which is opaque, an attr object (which can be NULL), a



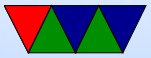
start_routine which is a C function called when it starts, an an arg argument to pass to the routine.

- Only can pass one argument. How can you pass more? pointer to a structure.
- With attributes you can set things like scheduling policies
- No routines for binding threads to specific cores, but some implementations include optional non-portable way. Also Linux has sched_setaffinity routine.



Terminating Threads

- pthread_exit()
- Returns normally from its starting routine
- another thread uses pthread_cancel() in it
- The entire process is terminated (by ending, or calling exit(), etc)

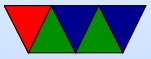


Thread Management

- pthread_join() lets a thread block until another one finishes

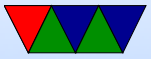
  So master can join all the children and wait until they are done before continuing.

- Argument to a join is a specific thread to wait on (so if waiting on four, have to have four calls to `pthread_join()`



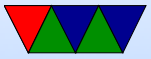
Stack Management

- Manage your own stack? Can get and set size. Be careful allocating too much on stack.



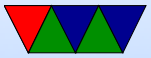
Mutexes

- Type of lock, only one thread can own it at a time. Can be used to avoid race conditions.



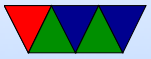
Condition Variables

- A way to avoid spinning on a mutex



Example code

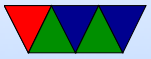
example code is posted on course website.



Simple Pthread Example

See `pthread_simple.c`

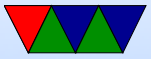
- Hardcodes 5 threads
- Do they run in any specific order?



Simple Init Example

See `pthread_init.c`.

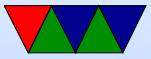
- Initializes 256MB of data. Number of threads from command line.
  Is this the most efficient way to init memory?
- Why do we have the sleep call? Note: you'd never want to write a real program using a sleep like that.
- Why errors if run on odd number?
  Be sure when splitting up problem handle remainders.



Simple Join Example

Can use join to make the master thread wait for the others to finish.

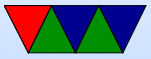
See `pthread_join.c`



Stack Example

How to see how much stack is available, and how to change it if not enough.

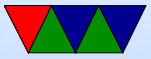
See `pthread_stack.c`



Mutex Example

See `pthread_mutex.c` for code w/o mutex (run with a num greater than 1)
Then see `pthread_mutex2.c` for core w mutex

Creates a "thread pool" and the threads can request more work when they finish.

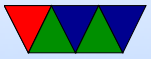


Mutex Info

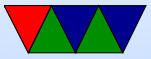
- Can create mutexes two ways,
  - Statically, when declared

```
pthread_mutex_t our_mutex = PTHREAD_MUTEX_INITIALIZER;
```

  - Dynamically with `pthread_mutex_init()` which allows setting mutex object attributes, attr.
- The mutex is initially unlocked.
- Can specify protocol, priority ceiling, and if it's shared/private.
- lock, unlock, trylock. Lock will spin until available,



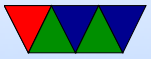
trylock is non-blocking.



Deadlock

When you have more than one lock, it is possible to end up nesting locks in ways that lockup a program with both threads getting stuck.

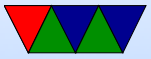
Thread 1	Thread 2
pthread_mutex_lock(&mutex1);	pthread_mutex_lock(&mutex2);
pthread_mutex_lock(&mutex2);	pthread_mutex_lock(&mutex1);



Condition Variable Example?

See `pthread_mutex.c`

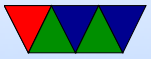
- Can have a thread start up sleeping on a lock, and wake up when signaled by another thread.



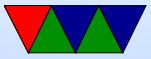
PAPI Example

See `pthread_papi.c`

- Initialize with:
  `PAPI_library_init(PAPI_VER_CURRENT);`
- You can/should check all functions to see if return `PAPI_OK`
- If using pthreads need to do:
  `PAPI_thread_init(pthread_self);`

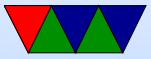


- Eventsets are just integers
  `int eventset=PAPI_NULL;`
- Gathered results are typically 64-bit integers
  `long long values[1];`
- Create an eventset:
  `PAPI_create_eventset(&eventset);`
- Add an event. Available events can be seen with the `papi_avail` and `papi_native_avail` commands.
- `PAPI_add_named_event(eventset,"PAPI_TOT_INS");`

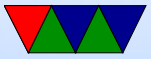


- Before the code of interest do a
  `PAPI_start(eventset);`

- Afterward do a
  `PAPI_stop(eventset,values);`
  and you can print the value or save it for later.



Debugging



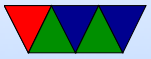
Race Conditions

```
x=0; // times we've run

x=x+1;          x=x+1;

ldr r0,x        ldr r0,x
add r0,#1       add r0,#1
str r0,x        str r0,x
```

- Shared counter address
  RMW on ARM
  Thread A reads value into reg
  Context switch happens



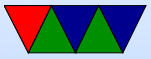
Thread B reads value into reg, increments, writes out

Context switch back to A

increments value, writes out

What happened?

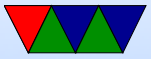
What should value be?



Critical Sections

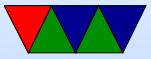
- Want mutual exclusion, only one can access structure at once

1. no two processes can be inside critical section at once
2. no assumption can be made about speed of CPU
3. no process not in critical section may block other processes
4. no process should wait forever



How to avoid

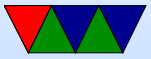
- Disable interrupts. Heavy handed, only works on single-core machines.
- Locks/mutex/semaphore



Mutex

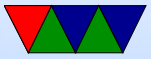
- mutex_lock: if unlocked (0), then it sets lock and returns
  if locked, returns 1, does not enter.
  what do we do if locked? Busy wait? (spinlock) re-schedule (yield)?

- mutex_unlock: sets variable to zero



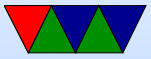
Semaphore

- Up/Down
- Wait in queue
- Blocking
- As lock frees, the job waiting is woken up



Locking Primitives

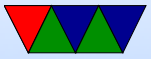
- fetch and add (bus lock for multiple cores), xadd (x86)
- test and set (atomically test value and set to 1)
- test and test and set
- compare-and-swap – Atomic swap instruction SWP (ARM before v6, deprecated)
  x86 CMPXCHG
  Does both load and store in one instruction!



Why bad? Longer interrupt latency (can't interrupt atomic op)
Especially bad in multi-core

- load-link/store conditional
  Load a value from memory
  Later store instruction to same memory address. Only succeeds if no other stores to that memory location in interim.
  ldrex/strex (ARMv6 and later)

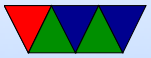
- Transactional Memory



Locking Primitives

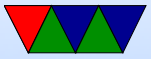
- can be shown to be equivalent

- how swap works:
  lock is 0 (free). r1=1; swap r1,lock
  now r1=0 (was free), lock=1 (in use)
  lock is 1 (not-free). r1=1, swap r1,lock
  now r1=1 (not-free), lock still==1 (in use)

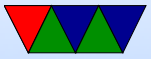


Memory Barriers

- Not a lock, but might be needed when doing locking
- Modern out-of-order processors can execute loads or stores out-of-order
- What happens a load or store bypasses a lock instruction?
- Processor Memory Ordering Models, not fun
- Technically on BCM2835 we need a memory barrier any time we switch between I/O blocks (i.e. from serial

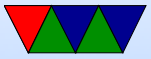


to GPIO, etc.) according to documentation, otherwise loads could return out of order



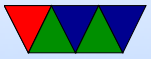
Deadlock

- Two processes both waiting for the other to finish, get stuck
- One possibility is a bad combination of locks, program gets stuck
- P1 takes Lock A. P2 takes Lock B. P1 then tries to take lock B and P2 tries to take Lock A.



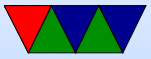
Livelock

- Processes change state, but still no forward progress.
- Two people trying to avoid each other in a hall.
- Can be harder to detect



Starvation

- Not really a deadlock, but if there's a minor amount of unfairness in the locking mechanism one process might get "starved" (i.e. never get a chance to run) even though the other processes are properly taking and freeing the locks.



How to avoid Deadlock

- Don't write buggy code
- Pre-emption (let one of the stuck processes run anyway)
- Rollback (checkpoint occasionally)
- What to do if it happens?
  - Reboot the system
  - Kill off stuck processes