Chapter 19 – Parallel Processing Part III

CPSC 240-09
John Overton

Review

Parallel Processing - pthreads

- pThreads is short for POSIX Threads
- pThreads is a thread library that works with the operating system to create and manage threads
- We will use pthread_create(), pthread_join() in our project from the C library
- pthread_create() is a function that will create a new thread and pass a parameter to the new thread. It will return the thread id of the newly created thread
- pthread_join() is a function to wait until a specific thread has ended. Can be used by the main thread to wait on a created thread

Race conditions

- When multiple threads simultaneously write to the same location at the same time
- NOT GOOD!
- Example from book:

$$\sum_{i=0}^{MAX-1} myValue = \left(\frac{myValue}{X}\right) + Y$$

```
for (int i=0; i < MAX; i++)
    myValue = (myValue / X) + Y;
```

Race conditions – Example from Book

```
MAX     equ     1000000000

; Perform MAX / 2 iterations to update myValue
        mov     rcx, MAX
        shr     rcx, 1                  ; divide by 2
        mov     r10, qword [x]
        mov     r11, qword [y]

incLoop0:     ; myValue = (myValue / x) + y
        mov     rax, qword [myValue]
        cqo
        div     r10
        add     rax, r11
        mov     qword [myValue], rax
        loop    incLoop0
        ret

        section .data
myValue dq      0
x       dq      1
y       dq      1
```

Race conditions – Example from Book

Since each thread is independent of other threads, chances are they may be doing different things at different times and could save a value to a variable at an inconvenient time

Step	Code: Core 0, Thread 0	Code: Core 1, Thread 1
1	`mov rax, qword [myValue]`	
2	`cqo`	`mov rax, qword [myValue]`
3	`div qword [x]`	`cqo`
4	`add rax, qword [y]`	`div qword [x]`
5	`mov qword [myValue], rax`	`add rax, qword [y]`
6		`mov qword [myValue], rax`

Ways to safeguard resources : Mutex lock and unlock

- A mutex (mutual exclusion) is a way to synchronize access to a resource (a variable or a data structure)
- A mutex is also called a lock: A thread can lock and unlock – when locked, it means one thread has obtained the right to access the resource
- Lock contention is when one or more threads request (attempt) to acquire a lock but another thread has already acquired the lock
- A deadlock is a case where thread A has acquired lock 1 and wants to acquire lock 2, which is already acquired by thread B, and thread B wants to acquire lock 1, already acquired by thread A. Both tasks are waiting for the other task's owned resource
- pthread_mutex_lock()
- Pthread_mutex_unlock()

Ways to safeguard resources : semaphore

- A semaphore is an integer whose value is never allowed to fall below zero.
- A semaphore is set to some initial number to indicate the number of resources it is to protect
- Then, two operations can be used by threads that need access to one or more of the set of resources
  - sem_wait() is called by a thread to request access to resources. If the semaphore is already zero, the thread waits until it is not zero
  - sem_post() is called by a thread to release access to the resources. It increments the semaphore so that other threads can use the resources. If there was a waiting thread, then it woken up, the semaphore is decremented and the thread has access to the resources
- A semaphore can be initialized to 1 – the semaphore would work like a mutex

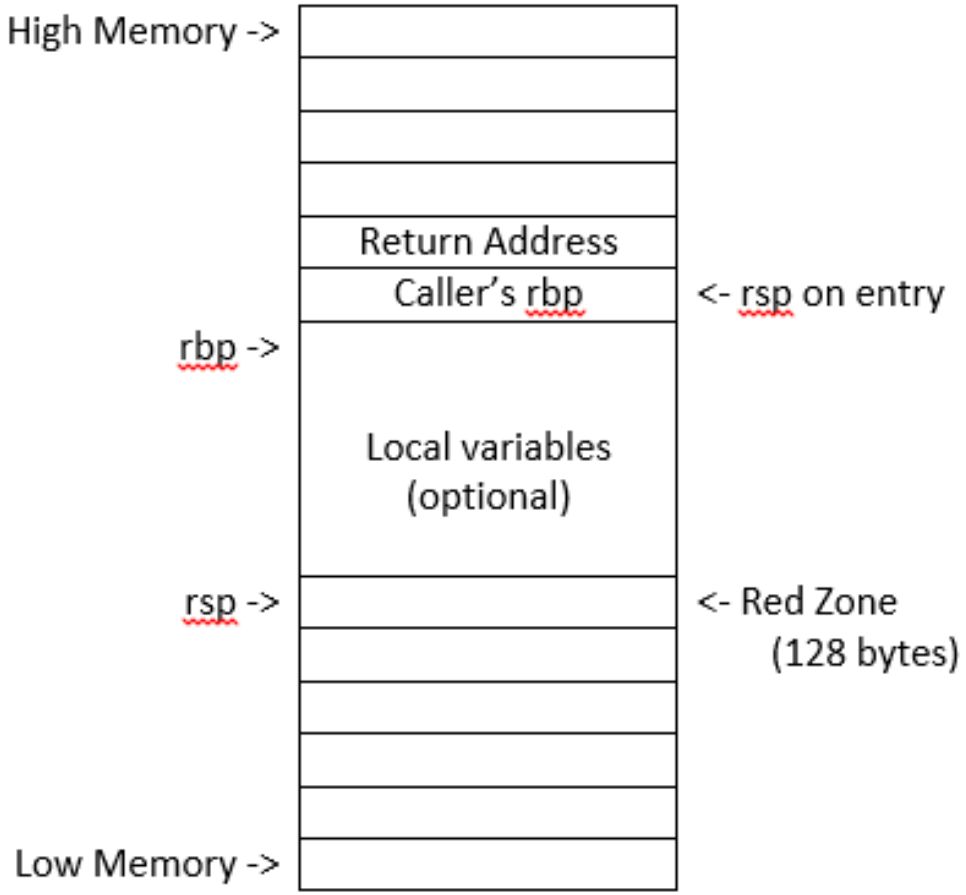
Ways to safeguard resources – Underlying architecture

- We need an atomic operation
  - An atomic operation means, do not interrupt me while I do this (sequence of) operation(s)
  - Or, if you need to interrupt me, act as if it never got started
- Two major architectural solutions: compare-and-swap and test-and-set
  - Compare-and-swap compares a value in memory and if it is equal, swap it with another value
  - Test-and-set makes a note of the address of a variable, then does a “test” on the value of the variable and if true, sets the variable to a new value. The note of the address is used to cause any other processor to wait if they were also trying to test-and-set using the address of the variable
- What does the x86 have? xchg and the lock prefix

Parallel Processing – Various things

More Parallel Processing…

- The Red Zone: An area of 128 bytes that is before the rsp pointer (remember, the rsp grows down)




```
_start:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 0x10 ; allocate local space
        …
```

```
;  Allocating local variables: 2 qword sized variables
var1        equ         -0x8
var2        equ         -0x10

; using local variables:
            mov         qword [rbp+var1], rax  ; save rax
            mov         rbx, qword [rbp+var2]
```

Assignment – Phase 3

In Phase 2, you created multiple threads to print “Hello World” and you could adjust the number of threads by changing your equate for MAX_THREADS

In Phase 3, you will replace your Hello World part of your threads with an equation to factor a number. That is, given a number, count the number of numbers that can be wholly divided into that number:

```
NUMBER_TO_FACTOR    equ  10000000000
RANGE_EACH_THREAD   equ  NUMBER_TO_FACTOR / MAX_THREADS

from = (threadIndex * RANGE_EACH_THREAD) + 1
to = from + RANGE_EACH_THREAD

for( i = from; i < to; i++ ) {
       if ( number % i == 0 )
              nbrFactors++
}

Then save nbrFactors in an array for the main thread to add all thread totals
together:  threadTotals[threadIndex] = nbrFactors

Use registers for all of the variables in this code since each thread gets its own
registers.  threadTotals would be an array in memory.
```

```
for ( int i = 0; i < MAX_THREADS; i++ ) {
        statement(s)
}
```

```
MAX_THREADS   equ   10

        mov   qword [i], 0
loop:
        cmp    qword [i], MAX_THREADS
        jge    loop_end
        …
        statement(s)
        …
        inc     qword [i]
        jmp     loop
loop_end:
        section .data
i       dq      0
```