**EECS 338**

***Today...***

1. **OpenMP “critical”**
2. **Peterson’s Solution (Ch 5)**
3. **Semaphores (Ch 5)**

***Next week: Classical synchronization problems (Ch 5)***

**OpenMP Review**

* POSIX Threads
  + <https://en.wikipedia.org/wiki/POSIX_Threads>
* OpenMP (Open Multi-Processing)
  + <https://en.wikipedia.org/wiki/OpenMP>

 vs. 

OpenMP POSIX

**OpenMP Review**

* Creating threads (two ways)
  + “#pragma omp parallel” before a block using {}
  + “#pragma omp parallel for” before a for-loop
* Variables: shared vs. private
  + Do not need global variables!
  + Variables declared before “#pragma omp” are shared
  + Variables declared after “#pragma omp” are private to each thread (same name but independent variable)

**OpenMP Review**

* Revised: openmp1.c, openmp2.c
  + What’s different:
    - declared “tid” inside the block or loop
    - removed the “private” clause
* *Also revised openmp2.c to only print # of threads only once*
  + Purpose: Uses “#pragma omp parallel **for**” to divide “i” ranges automatically!
  + Change in new version: if (tid == 0 **&& i == 1**)

**Review of Critical Region**

* Problems:
  + multiple threads/processes trying to use a shared variable
  + operations might not be atomic
    - Example: x = x + 1
* Example: Bounded buffer for producer + consumer
  + Section 3.4.1 has one example
  + Section 5.1 has a similar example
* Solutions: OpenMP, Peterson’s solution, semaphores

**Bounded buffer (Section 3.4.1)**

**Producer**

item next\_produced;

while (true) {

/\* produce an item in next produced \*/

while (((in + 1) % BUFFER\_SIZE) == out)

; /\* do nothing \*/

buffer[in] = next\_produced;

in = (in + 1) % BUFFER\_SIZE;

}

**Consumer**

item next\_consumed;

while (true) {  
 while (in == out)

; /\* do nothing \*/  
 next\_consumed = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

/\* consume the item in next consumed \*/

}

**Bounded buffer (Section 5.1)**

**Producer**

while (true) {  
 /\* produce an item in next produced \*/

while (counter == BUFFER\_SIZE) ;

/\* do nothing \*/

buffer[in] = next\_produced;

in = (in + 1) % BUFFER\_SIZE;

counter++;

}

**Consumer**

while (true) {

while (counter == 0)

; /\* do nothing \*/

next\_consumed = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

counter--;

/\* consume the item in next consumed \*/

}

**Exercise:** Write down the values in counter, buffer, in, and out with some values for next\_produced

**Solution #1: OpenMP “critical” clause**

* Problem: multiple threads trying to use a shared variable
  + Variables declared before “#pragma omp” are shared
  + Example: critical\_problem.c
  + Testing a few times may not reveal the danger!
* Solution: use “#pragma omp critical” inside block or loop
  + Example: critical\_solution.c

**Solution #2: Peterson’s Solution**



* petersons\_problem.c
* petersons\_solution.c

**Solution #3: Semaphores**

* semaphore = flag
  + <https://en.wikipedia.org/wiki/Flag_semaphore>
* semaphore1.c (from textbook)
  + Compile with -lpthread option on eecslinab server
  + sem\_init()
    - 2nd argument: 0=multithreaded, 1=multiprocess
    - <http://man7.org/linux/man-pages/man3/sem_init.3.html>
  + sem\_wait(), sem\_post()
  + sem\_destroy()
    - “On some implementations, omitting calls to sem\_destroy() doesn’t cause problems. On others, it can result in resource leaks.” (The Linux Programming Interface By Michael Kerrisk)

**Solution #3: Semaphores**

* Semaphores with fork() + shared memory
  + semaphore2\_problem.c, semaphore2\_solution.c
  + gcc -o semaphore2\_solution semaphore2\_solution.c -lpthread –lrt

**Semaphores (another examaple)**

* semaphore3\_problem.c, semaphore3\_solution.c
  + Uses multithreading
  + gcc -o semaphore3\_solution semaphore3\_solution.c -lpthread –lrt
* Semaphores with POSIX threads
  + semaphore3\_problem.c, semaphore3\_solution.c