Lecture Notes 3

Synchronization

HW Assignment 1 Overview diagram

In last lecture we covered Mutex (Mutual Exclusion).
The idea is to avoid two different threads accessing
the same variables (memory locations) at the same time.

- We talked about different ways of creating
  mutexes. In particular, we talked about

  - disable interrupts
  - spinlock

So we use these to create the mutex system.
Spinlock - inefficient but guaranteed to work

pti to control block

queue of pointers to control block

mutex:

\[
\begin{array}{c}
\text{lock} \\
\text{owner} \\
\text{queue}
\end{array}
\]

spinlock pti to thread

queue of threads

m.lock()

spin-lock m.lock

if owner == null

  owner = me

else

  add me to queue

  tmp = owner

spin-unlock m.lock

if tmp != me

  sleep

else

Unlock:

spin-lock m.lock

if q is empty

  owner = null

else

  owner = pop(queue)

  wake owner

spin-unlock (m.lock)
Classes of problems in Synchronization

Bounded buffer problem:

\[
\begin{align*}
\text{writer(s)} & \quad \text{put}(\cdot) \rightarrow \text{buffer} \rightarrow \text{get}(\cdot) \quad \text{reader(s)\
}\end{align*}
\]

Circular buffer

write modifies head pointer
reader modifies tail pointer

Implementation

\[
\begin{align*}
\text{int head, tail} \\
\text{item buf[N]} \\
\text{put() \quad while (head + 1) mod N = tail} \quad \text{do nothing} \\
& \quad \text{buffer[head]} = \text{item} \\
& \quad \text{head} = (h + 1) \mod N
\end{align*}
\]

Inefficient for multiple readers and writers
Solution for the problem using Semaphores.

Semaphore has

- Value
  wait () if there are other names / P() / lock / down
  signal () /

Semaphore s = new sem (3)

(Mutex is a special case of this where the value is 1)
  (binary semaphore)
Mutex m
Semaphore spaces (N), values (0)

get: wait(values)
    lock (m)
    val = buf[tail]
    tail = tail + 1 mod N
    unlock (m)
    Signal (spaces)

If value is available
    take it and signals
    a space.

Values = 0   space = 3

get():
    wait(values)
    {
    {
    {
        Signal (space)

put():
    wait(spaces)
    {
    {
        Signal (values)

In most cases we end up using a mutex
and two semaphores.
Reader / Writer problem:

We want shared read but exclusive write

An object that implements

read-lock / unlock
write lock / unlock

implementation

mutex m, int count, sem, wlock(1)

RL: lock(m)
if count = 0
wait (wlock)
count++
unlock(m)

RUL: lock(m)
count--
if count = 0
signal (wlock)
unlock(m)
WL: wait (wlock) \{ \text{need not block on mutex} \}
WL = signal (wlock)

Rules

1) Don’t read shared variables outside of a mutex. Use a local copy instead.
2) Rank locks to avoid deadlock
   (sometimes...)
3) Don’t sleep holding a mutex.