Semaphores (con't) (9/19/2019 CS4410 F Schneider) ①

Synchronization

- atomic load & store (only)
  - Peterson's mutual exclusion
- interlock instructions
  - TS (test & set)
- semaphores
  - general semaphores $0 \leq sem$
  - binary semaphore $0 \leq sem \leq 1$

P(sem): ⟨await sem > 0 then sem := sem − 1⟩

V(sem): ⟨sem := sem + 1⟩

- locks vs binary semaphores
  - acquire (l) vs P_b(sem)
  - release (l) vs V_b(sem)

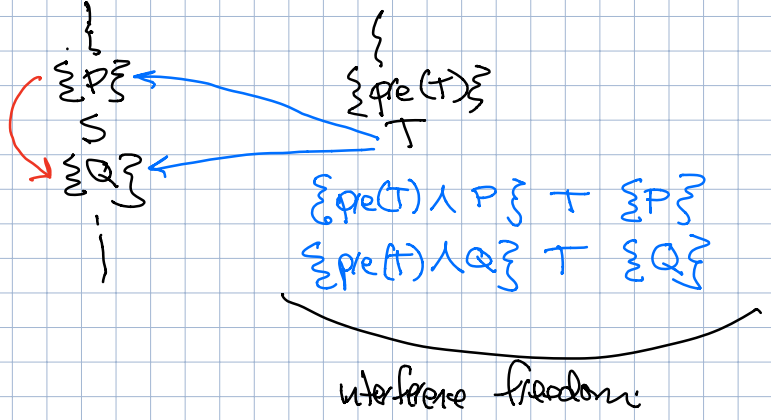
Classic Problems

Critical section / mutual exclusion
selective mutual exclusion
k-mutual exclusion

Bounded Buffer
N producers / consumers
1 producer 1 consumer

Techniques

- Assertions characterize state.



- Encode "conditions" to await as sough value

  E.g. await $N-(a-r) > 0$

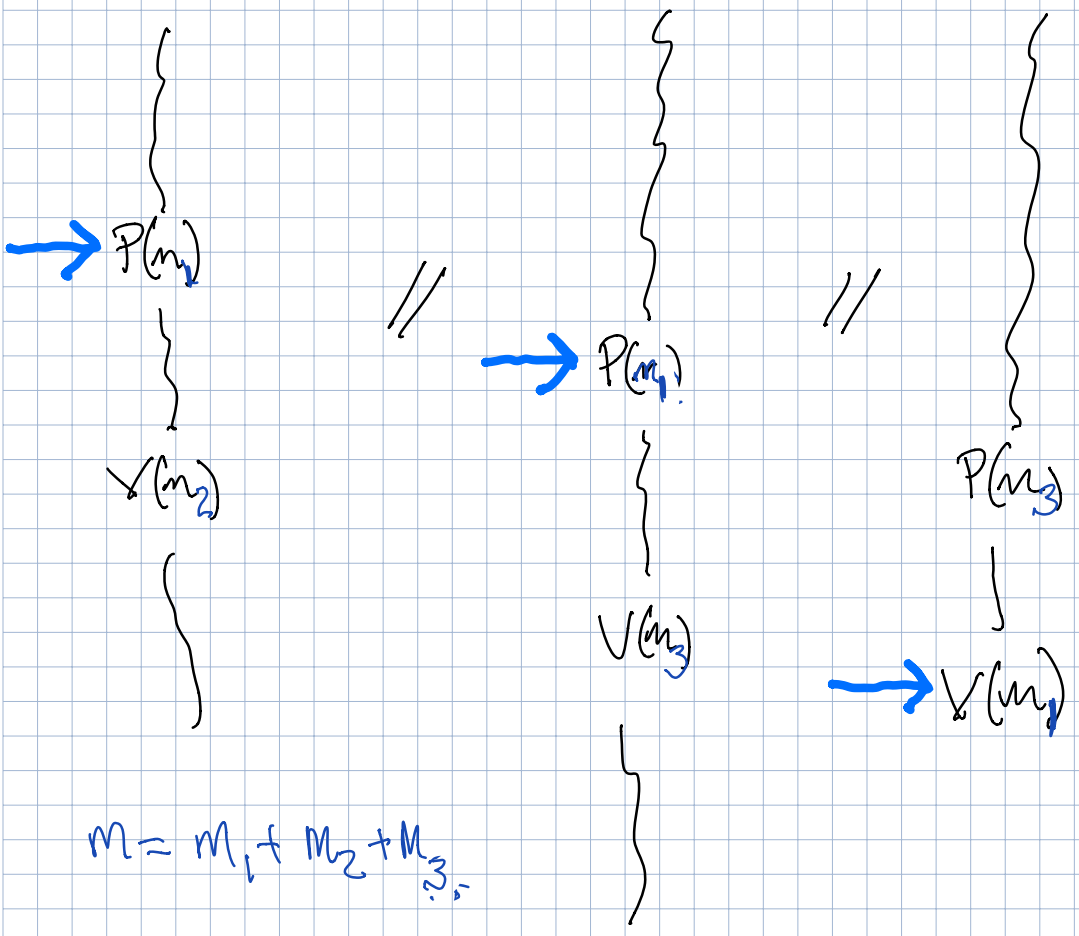
  vs P(slots)

  where I: .... slots = $N-(a-r)$

- "split binary semaphores"

  ← today!

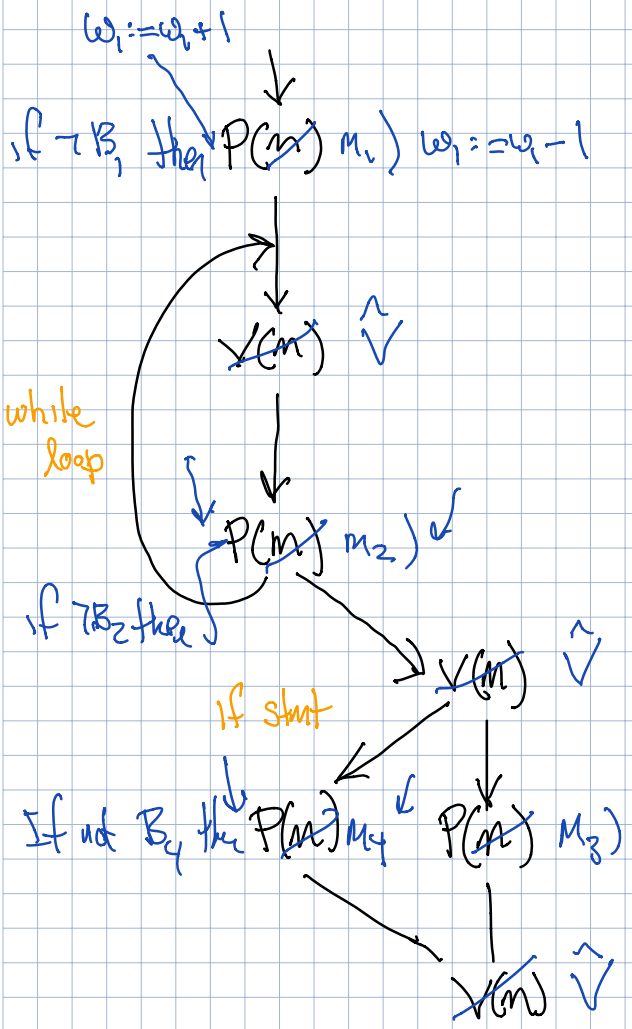
V(m) causes <u>some</u> process delayed on P(m) to resume. Choice is non-deterministic.



$m = m_1 + m_2 + m_3$ ∴

Program execution as an alternating sequence of P(m), V(m) operations.

Example program flow:



$M = M_1 + M_2 + M_3 + M_4$

B_1 B_2 B_3 B_4

w_1 w_2

... M_2 ...

waiting on M_1

$\hat{V}$: ~ does a V(M_i) where B_i holds & some process is waiting

P(m) → P(M_i)

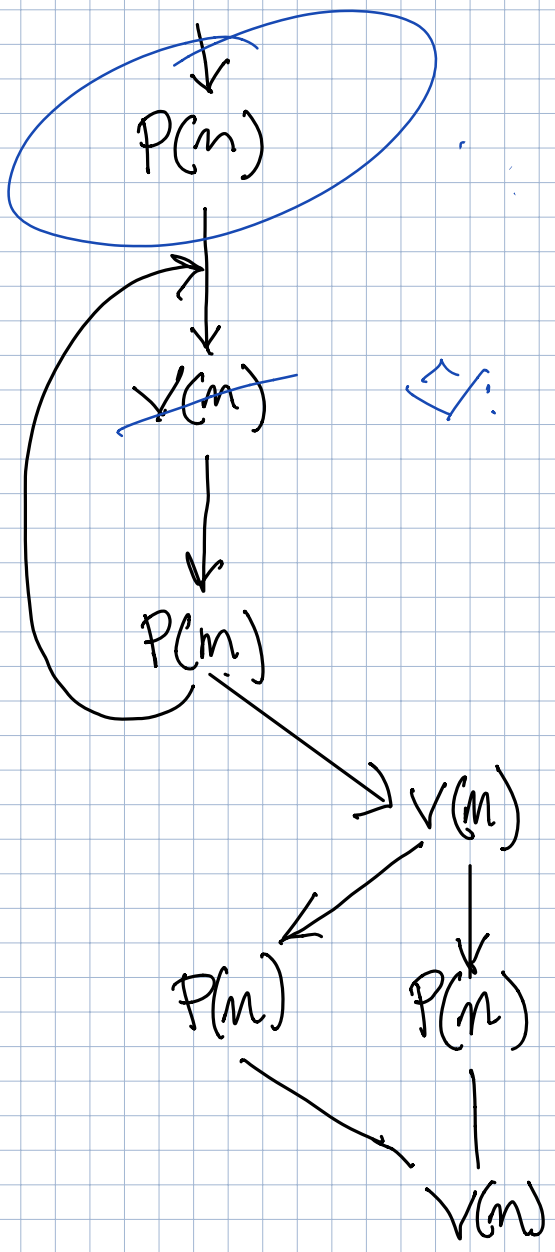
V(m) → $\hat{V}$

FC)

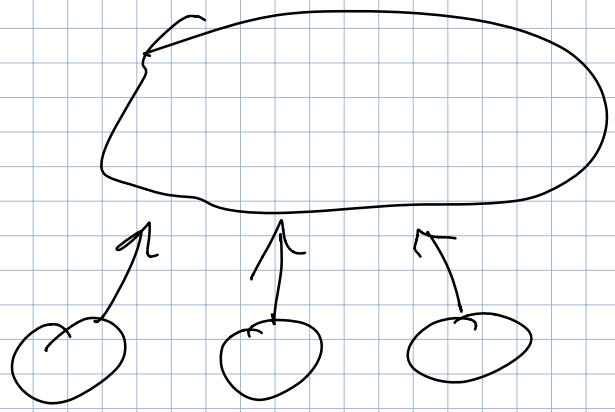
if B_i then
$w_i := w_i + 1$
· P(mi)

V: if ... B_i : $w_i > 0$ then
$w_i := w_i - 1$
V(mi)

else if ...



Readers / Writers Problem



Assume "end_" follows "start_"

StartRead
Read
EndRead

StartWrite
Write
EndWrite

Readers exclude writers (but not readers)
Writers exclude writers and readers

Readers / Writers Problem

Def:

r: # of current readers
w: # of current writers

Inv: $r \geq 0 \wedge w \geq 0 \wedge (w = 0 \vee (w = 1 \wedge r = 0))$

(9)

Inv: $r \geq 0 \wedge w \geq 0 \wedge (w = 0 \vee (w = 1 \wedge r = 0))$

StartRead: ~~await $w = 0$~~ P(m) ↓ if $w \neq 0$ then ↓P(sw) ; $ww := ww + 1$ ◇
{$w = 0$}
$r := r + 1$ ◇

EndRead: P(m)
$r := r - 1$ ◇

StartWrite: P(m) ↓ if $w \neq 0 \vee r \neq 0$ then ↓P(sr) ; $ww := ww + 1$ ◇
~~await $w = 0 \wedge r = 0$~~
{$w = 0 \wedge r = 0$}
$w := w + 1$ ◇

EndWrite: P(m)
$w := w - 1$
◇
◇ if $w = 0 \wedge r = 0 \wedge ww > 0$ then ↓V(sr) ; $ww := ww - 1$
else if $w = 0 \wedge wr > 0$ then ↓V(sw) ; $wr := wr - 1$
else V(m)

① add assigns to r & w
② add await to ensure Inv holds
③ use semphs for blocking: add P ops
④ add ◇ ops to unblock
⑤ only unblock if waiting; adjust ww, wr gives cond
6 add mutex to protect vars - P(m) ◇

(10)

Inv: $r \geq 0 \wedge w \geq 0 \wedge (w=0 \vee (w=1 \wedge r=0))$

StartRead P(n)
if $w \neq 0$ then $wr := wr+1$ $\hat{\vee}$ (1) $P(sw)$
$\{w=0\}$
$r := r+1$
$\hat{\vee}$ (2)

EndRead P(n)
$r := r-1$
$\hat{\vee}$ (3)

StartWrite P(n)
if $w \neq 0 \vee r \neq 0$ then $ww := ww+1$ $\hat{\vee}$ (4) $P(sr)$
$w := w+1$
$\hat{\vee}$ (5)

EndWrite P(n)
$w := w-1$
$\hat{\vee}$ (6)

$\hat{\vee}$: if $w=0 \wedge r=0 \wedge ww>0$ then $ww := ww-1$ $\vee(sr)$
else if $w=0 \wedge wr>0$ then $wr := wr-1$ $\vee(sw)$
else $\vee(n)$

Optimizedes of $\hat{V}$ given context.

① $w \neq 0$ holds

$\hat{V} \longrightarrow V(n)$

② $r \neq 0$ holds so delete

$\hat{V}$: if $w=0 \wedge r=0 \wedge ww>0$ then $ww := ww-1$ $V(ss)$
else if $w=0 \wedge wr>0$ then $wr := wr-1$ $V(sw)$
else $V(n)$

③ no change

④ $\neg(w=0 \wedge r=0)$ holds so delete

$\hat{V}$: if $w=0 \wedge r=0 \wedge ww>0$ then $ww := ww-1$ $V(ss)$
else if $w=0 \wedge wr>0$ then $wr := wr-1$ $V(sw$
else $V(n)$

⑤ $w>0$ holds so delete

$\hat{V}$: if $w=0 \wedge r=0 \wedge ww>0$ then $ww := ww-1$ $V(ss)$
else if $w=0 \wedge wr>0$ then $wr := wr-1$ $V(sw)$
else $V(n)$

R/W solns admit starvation

- Steady stream of readers will block writers
- Steady stream of writers will block readers

Solns:

1. give writers priority over late readers
2. give readers priority over late writers

Summary of split binary semaphore method

1. Define vars needed for problem
2. Define invariant that implies correctness
3. Add assignments to variables in (1)
4. Add await stmts for invariant
5. Add P to implement await
6. Add V to unblock at awaits
   - requires counters of blocked at each condition
7. Add P(m) to add mutex protection for variables
8. Simplify V based on precondition at each location