

---

# Paralelné programovanie

## Synchronizácia

Bc. št. prog. Informatika - 2010/2011

---

Ing. Michal Čerňanský, PhD.

Fakulta informatiky a  
informačných technológií,  
STU Bratislava

---

# Synchronizácia

- Model vlákien – zdieľaná pamäť
  - Komunikácia - viaceré výhody
    - Jednoduchosť – žiadne špec. API
    - Rýchlosť
  - Vlákna – nedeterministické plánovanie
    - Problémy
-

---

# Synchronizácia

- Thread 1:  $x := 1;$
  - Thread 2:  $x := 2;$
  
  - $x:=0;$
  - Thread 1:  $x := x+1;$
  - Thread 2:  $x := x+1;$
  
  - $x:=0;$
  - Thread 1: for ( $i=0;i<10000000;i++$ )  $x := x+1;$
  - Thread 2: for ( $i=0;i<10000000;i++$ )  $x := x-1;$
-

---

# Synchronizácia

- Synchronizácia
    - Úlohy vykonať v stanovenom poradí
    - Úlohy nevykonať súčasne
  
    - Kontrola prístupu do pamäte
    - Vstupno-výstupné operácie
-

---

# Vzájomné vylučovanie

- Zabezpečiť, aby sa úlohy A a B nevykonávali „súčasne“
- Mutex (Pthreads):

**Init:**

```
mutex_init(&mutex);
```

**Thread A:**

```
mutex_lock(&mutex);  
...  
// critical section A  
...  
mutex_unlock(&mutex);
```

**Thread B:**

```
mutex_lock(&mutex);  
...  
// critical section B  
...  
mutex_unlock(&mutex);
```

---

---

# Vzájomné vylučovanie

## ■ Semafóry:

### **Init:**

```
sem_init(&mutex, 1);
```

### **Thread A:**

```
sem_wait(&mutex);  
...  
// critical section  
...  
sem_signal(&mutex);
```

### **Thread B:**

```
sem_wait(&mutex);  
...  
// critical section  
...  
sem_signal(&mutex);
```

---

---

# Zabezpečenie poradia vykonávania - signalizácia

## ■ Semafóry:

### **Init:**

```
sem_init(sem,0);
```

### **Thread A:**

```
...  
sem_wait (sem);  
...  
// do some work A  
...
```

### **Thread B:**

```
...  
// do some work B  
...  
sem_signal(sem);  
...
```

---

---

# Zabezpečenie poradia vykonávania - signalizácia

## ■ Pthreads:

### Init:

```
mutex_init(mutex);  
cond_init(cond);
```

### Thread A:

```
...  
mutex_wait(mutex);  
cond_wait(cond, mutex);  
mutex_unlock(mutex);  
...  
// do some work A  
...
```

### Thread B:

```
...  
// do some work B  
...  
cond_signal(cond);  
...
```

---



---

# Vzájomné vylučovanie

- Ak B signalizuje skôr ako A - uviaznutie

## **Init:**

```
mutex_init(mutex);  
cond_init(cond);  
work_B_done = FALSE;
```

## **Thread A:**

```
...  
mutex_wait(mutex);  
if (! work_B_done)  
    cond_wait(cond, mutex);  
mutex_unlock(mutex);  
...  
// do some work A  
...
```

## **Thread B:**

```
...  
// do some work B  
...  
mutex_wait(mutex);  
work_B_done = TRUE;  
cond_signal(cond);  
mutex_unlock(mutex);  
...
```

---

---

# Vzájomné vylučovanie

- `if (! work_B_done) cond_wait(cond, mutex);` - nestačí `if`
  - `while (! work_B_done) cond_wait(cond, mutex);` - správna konštrukcia
  
  - “Spurious Wakeups” – „falošné prebudenie“
    - `cond_wait(cond,mutex)` môže skončiť aj keď podmienená premenná nebola signalizovaná
    - Dôsledok náročnosti implementácie v multiprocessorovom poč. systéme
  - Správny štýl programovania
    - Po obdržaní mutexu môže byť už podmienka neplatná
-

# Zabezpečenie poradia vykonávania - signalizácia

## ■ Správne riešenie:

### **Init:**

```
mutex_init(mutex);  
cond_init(cond);  
work_B_done = FALSE;
```

### **Thread A:**

```
...  
mutex_lock(mutex);  
while (! work_B_done)  
    cond_wait(cond, mutex);  
mutex_unlock(mutex);  
...  
// do some work A  
...
```

### **Thread B:**

```
...  
// do some work B  
...  
mutex_lock(mutex);  
work_B_done = TRUE;  
cond_signal(cond);  
mutex_unlock(mutex);  
...
```

---

# Synchronizačné primitívy

- Základné synchronizačné konštrukcie
  - Je možné vybudovať zložitejšie konštrukcie
  
  - Vzájomné vylučovanie + signalizácia
    - Mutexy + podmienené premenné
    - Semafóry
-

---

# Multiplex (Downey)

- Iba stanovený počet úloh v „kritickej oblasti“
- Semafóry:

## Init:

```
sem_init(sem, N);
```

## Thread A:

```
sem_wait(sem);  
...  
// critical section A  
...  
sem_signal(sem);
```

## Thread B:

```
sem_wait(sem);  
...  
// critical section B  
...  
sem_signal(sem);
```

## Thread C:

```
sem_wait(sem);  
...  
// critical section C  
...  
sem_signal(sem);
```

---

---

# Multiplex

## ■ Pthreads:

### **Init:**

```
mutex_init(mutex)
cond_init(cond)
count := N;
```

### **Thread A :**

```
mutex_lock(mutex)
if count == 0 then
    cond_wait(cond, mutex)
count = count - 1
...
// critical section A
...
count = count + 1
if count == 1 then
    cond_signal(cond)
mutex_unlock()
```

### **Thread B :**

```
mutex_lock(mutex)
if count == 0 then
    cond_wait(cond, mutex)
count = count - 1
...
// critical section B
...
count = count + 1
if count == 1 then
    cond_signal(cond)
mutex_unlock()
```

### **Thread C :**

```
mutex_lock(mutex)
if count == 0 then
    cond_wait(cond, mutex)
count = count - 1
...
// critical section C
...
count = count + 1
if count == 1 then
    cond_signal(cond)
mutex_unlock()
```

---

---

# Multiplex

- Iba jedno vlákno pristupuje do „kritickej sekcie“
  - Potreba uvoľňovať mutex
  - Falošné zobudenia – použiť „while“ cyklus
-

---

# Multiplex

## ■ Pthreads:

### **multiplex\_init(multiplex,N)**

```
mutex_init(mutex)
cond_init(cond)
count := N;
```

### **multiplex\_lock(multiplex) :**

```
mutex_lock(mutex);
while (count == 0)
    cond_wait(cond, mutex);
count = count - 1;
mutex_unlock(mutex);
```

### **multiplex\_unlock(multiplex) :**

```
mutex_lock(mutex);
count = count + 1;
if (count == 1) then
    cond_signal(cond);
mutex_unlock();
```

---



---

# Bariéra

- Bod stretnutia
  - Kritický bod
  - Žiadne vlákno neprejde cez kritický bod skôr ako všetky vlákna nedosiahnu bod stretnutia
-

---

# Bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&barrier, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;  
sem_signal(&mutex);
```

```
if (count == n) sem_signal(&barrier);
```

```
sem_wait(&barrier);
```

```
// critical point
```

---

---

# Bariéra

- Iba jedno vlákno prejde cez bariéru, ostatné uviaznú
- Vždy?



---

# Bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&barrier, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;  
sem_signal(&mutex);
```

```
if (count == n) sem_signal(&barrier);
```

```
sem_wait(&barrier);  
sem_signal(&barrier);
```

```
// critical point
```

---

---

# Bariéra

- Problém odstránený
  - Bariéra funkčná
  - Prístup k premennej „count“ mimo kritickej oblasti
-

---

# Bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&barrier, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;
```

```
if (count == n) sem_signal(&barrier);
```

```
sem_wait(&barrier);  
sem_signal(&barrier);  
sem_signal(&mutex);
```

```
// critical point
```

---

---

# Bariéra

- Chyba – uviaznutie
- Častá chyba – Čakanie na semafór + uzamknutý mutex



---

# Bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&barrier, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;  
  
if (count == n) sem_signal(&barrier);  
sem_signal(&mutex);  
  
sem_wait(&barrier);  
sem_signal(&barrier);  
  
// critical point
```

---



---

# Turniket

- Vzor „Turniket“ (Turnstile)
  - Turniket je zařízení, které funguje jako brána, kterou může v jednu chvíli projít pouze jeden člověk. Turnikety byly původně používány jako jiný druh ohrady, který měl dovolit průchod lidem, ale zabránit ovčím a jiným zvířatům ohradu opustit. Dnes se používají zejména k usměrnění pohybu lidí. (Zdroj: wikipedia)



---

# Turniket

## **Init:**

`sem_init(turnstile,1)`

## **Threads:**

`sem_wait(turnstile)`

`sem_signal(turnstile)`

## **Control thread:**

`sem_wait(turnstile)` - vypnutie turniketu

`sem_signal(turnstile)` - zapnutie turniketu

---

---

# Znovupoužitelná bariéra

- Bariéra - správne riešenie ale chceme znovupoužitelnú bariéru
- Vlákna vykonávajú prácu v cykle

## Threads:

```
while(do_loop) {  
  
    // some task  
  
    barrier();  
}
```

---

---

# Znovupoužitelná bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&turnstile, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;  
sem_signal(&mutex);  
  
if (count == n) sem_signal(&turnstile);  
  
sem_wait(&turnstile);  
sem_signal(&turnstile);  
  
// critical point  
  
sem_wait(&mutex);  
count := count - 1;  
sem_signal(&mutex);  
  
if (count == 0) sem_wait(&turnstile);
```

---

---

# Znovupoužitelná bariéra

- Možné viacnásobné signalizovanie semafóra „turnstile“
  - Všetky vlákna môžu teoreticky signalizovať semafór „turnstile“
-

---

# Znovupoužitelná bariéra

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&turnstile, 0);  
count := 0;  
n;
```

## **Thread:**

```
sem_wait(&mutex);  
count := count + 1;  
if (count == n) sem_signal(&turnstile);  
sem_signal(&mutex);  
  
sem_wait(&turnstile);  
sem_signal(&turnstile);  
  
// critical point  
  
sem_wait(&mutex);  
count := count - 1;  
if (count == 0) sem_wait(&turnstile);  
sem_signal(&mutex);
```

---

---

# Znovupoužitelná bariéra

- Vlákno po prechode cez bariéru, vykoná prácu a znovu môže prejsť cez bariéru



---

# Znovupoužitelná bariéra

## Init:

```
sem_init(&mutex, 1);
sem_init(&turnstile1, 0);
sem_init(&turnstile2, 1);
count := 0;
n;
```

## Thread:

```
sem_wait(&mutex);
count := count + 1;
if (count == n) {
    sem_signal(&turnstile1);
    sem_wait(&turnstile2);
}
sem_signal(&mutex);

sem_wait(&turnstile1);
sem_signal(&turnstile1);
```

```
// critical point
```

```
sem_wait(&mutex);
count := count - 1;
if (count == 0) {
    sem_signal(&turnstile2);
    sem_wait(&turnstile1);
}
sem_signal(&mutex);

sem_wait(&turnstile2);
sem_signal(&turnstile2)
```

---



---

# Bariéra

- Riešenie s použitím Pthreads prostriedkov
  - Mutex
  - Podmienené premenné
-

---

# Bariéra

## **Init:**

```
mutex_init(mutex);  
cond_init(cond);  
count := 0;  
n;
```

## **Thread:**

```
mutex_lock(mutex);  
count := count + 1;  
mutex_unlock(mutex);
```

```
if (count == n) {  
    cond_signal(cond);  
}
```

```
cond_wait(cond, mutex);  
mutex_unlock(mutex);
```

```
// critical point
```

---

---

# Bariéra

- Iba jedno vlákno je prebudené a prejde bariérou
  - Nesprávna práca s podmienenou premennou
  - Ošetrenie prístupu k premennej „count“
-

---

# Bariéra

## Init:

```
mutex_init(mutex);  
cond_init(cond);  
count := 0;  
n;
```

## Thread:

```
mutex_lock(mutex);  
count := count + 1;  
mutex_unlock(mutex);  
  
if (count == n)  
    cond_broadcast(cond);  
  
cond_wait(cond, mutex);  
mutex_unlock(mutex);  
  
// critical point
```

## Thread:

```
mutex_lock(mutex);  
count := count + 1;  
mutex_unlock(mutex);  
  
if (count == n)  
    cond_signal(cond);  
  
cond_wait(cond, mutex);  
cond_signal(cond);  
mutex_unlock(mutex);  
  
// critical point
```

---

---

# Bariéra

- Signály sa nezachovávajú pre nasledujúce použitie
  - Uviaznutie
  - Nesprávna práca s podmienenou premennou
  - Ošetrovanie prístupu k premennej „count“
-

---

# Bariéra

## Init:

```
mutex_init(mutex);  
cond_init(cond);  
count := 0;  
n;
```

## Thread:

```
mutex_lock(mutex);  
count := count + 1;  
  
if (count == n)  
    cond_broadcast(cond);  
else  
    cond_wait(cond, mutex);  
  
mutex_unlock(mutex);
```

```
// critical point
```

## Thread:

```
mutex_lock(mutex);  
count := count + 1;  
  
if (count == n)  
    cond_broadcast(cond);  
else  
    while (count != n) cond_wait(cond, mutex);  
  
mutex_unlock(mutex);
```

```
// critical point
```

---

---

# Znovupoužitelná bariéra

- Vlákna vykonávají práci v cykle



---

# Znovupoužitelná bariéra

## Init:

```
mutex_init(mutex);
cond_init(cond);
mutex_init(mutexI);
cond_init(condI);
count := 0;
in_barrier := FALSE;
n;
```

## Thread:

```
mutex_lock(mutexI);
while (in_barrier) cond_wait(condI, mutexI);
mutex_unlock(mutexI);

mutex_lock(mutex);
count := count + 1;
```

```
if (count == n) {
    in_barrier := TRUE;
    cond_broadcast(cond);
}
else
    while (!in_barrier) cond_wait(cond,mutex);

// critical point

count := count - 1;
if (count == 0) {
    in_barrier := FALSE;
    cond_broadcast(condI);
}

mutex_unlock(mutex);
```

---



---

# Znovupoužitelná bariéra

- Chyba – uviaznutie, nutnosť ošetriť nastavovanie premennej „in\_barrier“



---

# Znovupoužitelná bariéra

## Init:

```
mutex_init(mutex);
cond_init(cond);
count := 0;
in_barrier := FALSE;
n;
```

## Thread:

```
mutex_lock(mutex);
while (in_barrier) cond_wait(cond, mutex);

count := count + 1;
if (count == n) {
    in_barrier := TRUE;
    cond_broadcast(cond);
}
else
    while (!in_barrier) cond_wait(cond,mutex);

// critical point

count := count - 1;
if (count == 0) {
    in_barrier := FALSE;
    cond_broadcast(cond);
}

mutex_unlock(mutex);
```

---

---

# FIFO Mutex

- Mutex, Semafór – nie je určené, ktoré vlákna budú po signalizácii prebudené
  - Implementácia FIFO dát. štruktúry - frontu
  - Prebúdžanie vlákna na čele frontu
-

---

# FIFO Mutex

## Init:

```
sem_init(&mutex, 1);  
fifo_init(&fifo);
```

## Thread\_init:

```
sem_init(&mysem, 0);
```

## Thread:

```
fifomutex_lock()  
    sem_wait(&mutex);  
    if (fifo_isempty()) {  
        fifo_add(&mysem);  
        sem_signal(&mutex);  
    }  
    else {  
        fifo_add(&mysem);  
        sem_signal(&mutex);  
        sem_wait(&mysem);  
    }  
}
```

```
fifomutex_unlock()  
    sem_wait(&mutex);  
    fifo_remove();  
    if (!fifo_isempty()) {  
        sem = fifo_top();  
        sem_signal(&sem);  
    }  
    sem_signal(&mutex);  
}
```

---

---

# FIFO Mutex

## Init:

```
mutex_init(mutex);
fifo_init(fifo);
for (i=0; i<n; i++) cond_init(cond[i]);
```

## Thread\_init:

```
tid
```

## Thread:

```
fifomutex_lock()
  mutex_lock(mutex);
  if (!fifo_isempty()) {
    fifo_add(cond[tid]);
    while (fifo_top() !=cond[tid])
      cond_wait(cond[tid], mutex);
    fifo_remove();
  }
```

```
fifomutex_unlock()
  if (!fifo_isempty())
    cond_signal(fifo_top());
  mutex_unlock(mutex);
```

---

---

# FIFO Mutex

- Iba jedno vlákno sa dostane za mutex
- Vo fronte ziadna hodnota



---

# FIFO Mutex

## Init:

```
mutex_init(mutex);  
fifo_init(fifo);  
for (i=0; i<n; i++) cond_init(cond[i]);
```

## Thread\_init:

```
tid
```

## Thread:

```
fifomutex_lock()  
    mutex_lock(mutex);  
    fifo_add(cond[tid]);  
    while (fifo_top() != cond[tid])  
        cond_wait(cond[tid], mutex);  
    mutex_unlock(mutex);
```

```
fifomutex_unlock()  
    mutex_lock(mutex);  
    fifo_remove();  
    if (!fifo_isempty())  
        cond_signal(fifo_top());  
    mutex_unlock(mutex);
```

---

---

# Producers - Consumers

- Producenti a konzumenti
  - Producenti vytvárajú položky a umiestňujú ich do dátových štruktúr
  - Konzumenti vyberajú položky z dátových štruktúr a spracovávajú ich
-



---

# Producers - Consumers

- Synchronizačné obmedzenia
  - Dátová štruktúra – bufer – je počas zapisovania alebo čítania položky v nekonzistentnom stave
  - Keď v dát. štruktúre nie je žiadna položka, prípadný konzument začne čakať
-

---

# Producers - Consumers

## **Init:**

```
sem_init(mutex, 1);  
sem_init(items, 0);  
fifo_init(buffer);
```

## **Thread Producer:**

```
item = produce_item();  
  
sem_wait(mutex);  
buffer_add(item);  
sem_signal(items);  
sem_signal (mutex);
```

## **Thread Consumer:**

```
sem_wait(items);  
sem_wait(mutex);  
item = fifo_remove();  
sem_signal(mutex);  
  
process_item(item);
```

---

---

# Producers - Consumers

- Drobné zlepšenie v Producer vláknach
  - Consumer je okamžite zablokovaný na semafore „mutex“ ak Producer signalizuje cez semafor „items“
  - Zobúdzanie a uspávanie vlákien – drahé operácie
-

---

# Producers - Consumers

## **Init:**

```
sem_init(mutex,1);  
sem_init(items,0);  
fifo_init(buffer);
```

## **Thread Producer:**

```
item = produce_item();  
  
sem_wait(mutex);  
buffer_add(item);  
sem_signal(items);  
sem_signal(mutex);
```

## **Thread Consumer:**

```
sem_wait(mutex);  
sem_wait(items);  
item = fifo_remove();  
sem_signal(mutex);  
  
process_item(item);
```

---

---

# Producers - Consumers

- Chyba – uviaznutie
  - Consumer získa mutex a čaká na semafóre
  - Producers nemôžu vkladat' položky
-

---

# Producers - Consumers

## **Init:**

```
sem_init(mutex,1);  
sem_init(items,0);  
fifo_init(buffer);
```

## **Thread Producer:**

```
item = produce_item();  
  
sem_wait(mutex);  
buffer_add(item);  
sem_signal (mutex);  
sem_signal(items);
```

## **Thread Consumer:**

```
sem_wait(items);  
sem_wait(mutex);  
item = fifo_remove();  
sem_signal(mutex);  
  
process_item(item);
```

---

---

# Readers- Writers

- Čitatelia – pisatelia
  - Čitatelia čítajú z „kritickej oblasti“
  - Pisatelia zapisujú do kritickej oblasti
  - Viacerí čitatelia môžu súbežne čítať
  - Iba jeden pisateľ môže zapisovať (nikto iný nemôže ani čítať ani zapisovať)
-

---

# Readers- Writers

## **Init:**

```
sem_init(&mutex, 1);  
sem_init(&room_empty, 1);  
count = 0;
```

## **Thread Reader:**

```
sem_wait(&mutex);  
if (count == 0) sem_wait(&room_empty);  
count++;  
sem_signal(&mutex);  
...  
// reading  
...  
sem_wait(&mutex);  
count--;  
if (count == 0) sem_signal(&room_empty);  
sem_signal(&mutex);
```

## **Thread Writer:**

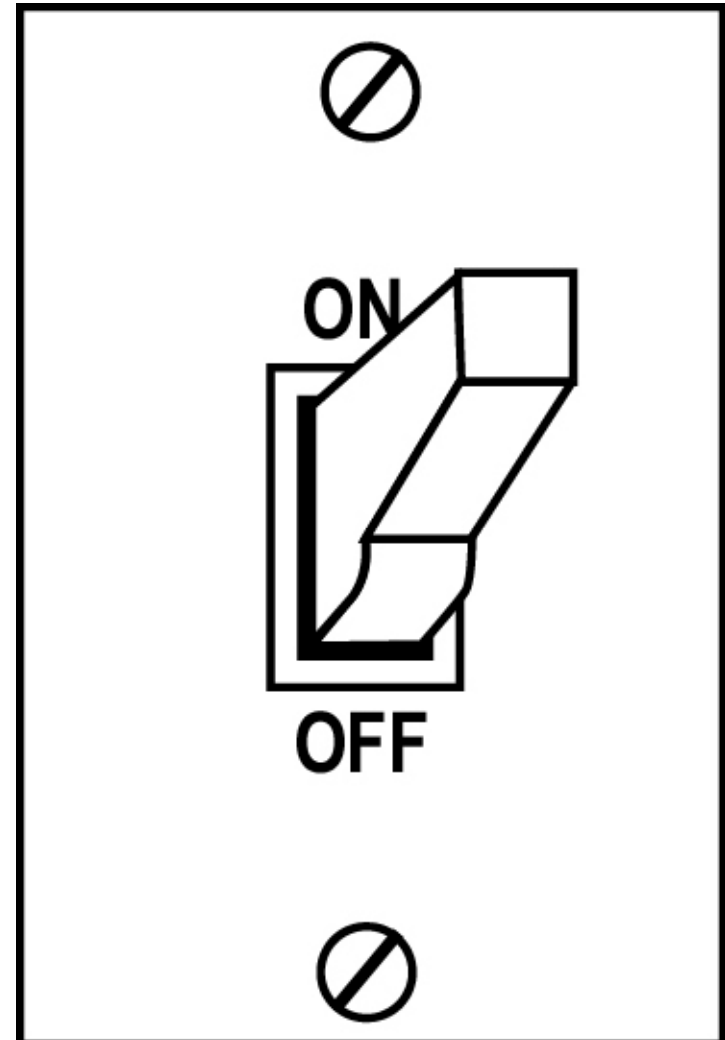
```
sem_wait(&room_empty);  
...  
// writing  
...  
sem_signal(&room_empty);
```

---



# Vypínač

- Vzor „Vypínač“ (Lightswitch)
  - Vypínač svetla v miestnosti, prvý človek, ktorý vojde do miestnosti ho zapne a posledný človek, ktorý vychádza z miestnosti ho vypne.



---

# Vypínač

## **Init:**

```
sem_init(&mutex,1) ;  
counter :=0;
```

## **Threads:**

```
lightswitch_lock(sem)  
  sem_wait(&mutex)  
  if (counter == 0) sem_wait(sem)  
  counter ++;  
  sem_signal(&mutex)
```

```
lightswitch_unlock(sem)  
  sem_wait(&mutex)  
  counter --;  
  if (counter == 0) sem_signal(sem)  
  sem_signal(&mutex)
```

---

---

# Readers- Writers

## **Init:**

```
sem_init(&room_empty,1);  
lightswitch_init(&lswitch);
```

## **Thread Reader:**

```
lightswitch_lock(&lswitch,&room_empty)  
...  
// reading  
...  
lightswitch_unlock(&lswitch,&room_empty)
```

## **Thread Writer:**

```
sem_wait(&room_empty);  
...  
// writing  
...  
sem_signal(&room_empty);
```

---

---

# Readers- Writers

- Pisatelia čakajú, pokiaľ je v oblasti nejaký čitateľ
  - Čitatelia môžu spôsobiť, že pisatelia sa nedostanú do „kritickej oblasti“
  - Vyhladovanie - Starvation
-

---

# Readers- Writers

## **Init:**

```
sem_init(room_empty,1);  
lightswitch_init(lswitch);  
sem_init(turnstile,1);
```

## **Thread Reader:**

```
sem_wait(turnstile);  
sem_signal(turnstile);  
  
lightswitch_lock(lswitch,room_empty)  
...  
// reading  
...  
lightswitch_unlock(lswitch,room_empty)
```

## **Thread Writer:**

```
sem_wait(turnstile);  
...  
// writing  
...  
sem_signal(turnstile);
```

---

---

# Readers- Writers

- Zapisovateľ vypne turniket aby čitatelia nemohli vstúpiť do „kritickej oblasti“
  - Nepočká, kým všetci čitatelia odídu
-

---

# Readers- Writers

## **Init:**

```
sem_init(room_empty,1);  
lightswitch_init(lswitch);  
sem_init(turnstile,1);
```

## **Thread Reader:**

```
sem_wait(turnstile);  
sem_signal(turnstile);  
  
lightswitch_lock(lswitch, room_empty)  
...  
// reading  
...  
lightswitch_unlock(lswitch,room_empty)
```

## **Thread Writer:**

```
sem_wait(turnstile);  
sem_wait(room_empty);  
...  
// writing  
...  
sem_signal(room_empty);  
sem_signal(turnstile);
```

---

---

# Readers- Writers

- 1st Readers – Writers Problem
    - Readers Preference – uprednostnenie čitateľov
    - Viacerí čitatelia môžu čítať
    - Vyhľadovanie pisateľov
  - 2nd Readers – Writers Problem
    - Writers Preference – uprednostnenie pisateľov
    - Pisateľ nesmie čakať dlhšie ako je nevyhnutné
    - Vyhľadovanie čitateľov
  - 3rd Readers – Writers Problem - rovnováha
-



---

# Readers- Writers

## **Init:**

```
sem_init(hall_empty,1);  
lightswitch_init(hall_lswitch);
```

```
sem_init(room_empty,1);  
lightswitch_init(room_lswitch);
```

## **Thread Reader:**

```
sem_wait(hall_empty);  
lightswitch_lock(room_lswitch, room_empty);  
sem_signal(hall_empty);  
...  
// reading  
...  
sem_wait(hall_empty);  
lightswitch_unlock(room_lswitch, room_empty);  
sem_signal(hall_empty);
```

## **Thread Writer:**

```
lightswitch_lock(hall_lswitch, hall_empty);  
sem_wait(room_empty);  
...  
...  
// writing  
...  
...  
sem_signal(room_empty);  
lightswitch_unlock(hall_lswitch, hall_empty);
```

---

---

# Readers- Writers

## Pthreads – Readers Preference

### **Init:**

```
mutex_init(&mutex);  
cond_init(&cond);  
count = 0;
```

### **Thread Reader:**

```
mutex_lock(&mutex);  
count++;  
mutex_unlock(&mutex);  
...  
// reading  
...  
mutex_lock(&mutex);  
count--;  
if (count == 0) cond_broadcast(&cond);  
mutex_unlock(&mutex);
```

### **Thread Writer:**

```
mutex_lock(&mutex);  
while (count != 0) cond_wait(&cond, &mutex);  
...  
// writing  
...  
mutex_unlock(&mutex);
```

---

---

# Readers- Writers

## Pthreads – Writers Preference

### Init:

```
mutex_init(&mutex);  
cond_init(&r_cond); count = 0;  
cond_init(&w_cond); is_writer = FALSE;
```

### Thread Reader:

```
mutex_lock(&mutex);  
while (is_writer) cond_wait(&w_cond, &mutex);  
count ++;  
mutex_unlock(&mutex);  
...  
// reading  
...  
mutex_lock(&mutex);  
count --;  
if (count == 0) cond_signal(&r_cond);  
mutex_unlock(&mutex);
```

### Thread Writer:

```
mutex_lock(&mutex);  
while (count != 0) cond_wait(&r_cond, &mutex);  
is_writer = TRUE;  
mutex_unlock(&mutex);  
...  
// writing  
...  
mutex_lock(&mutex);  
is_writer = FALSE;  
cond_signal(&w_cond);  
mutex_unlock(&mutex);
```

---

---

## Readers- Writers

- Uprednostnenie pisateľov – pisateľ, skôr ako začne čakať musí „upovedomiť“ čitateľov
    - `is_writer = TRUE` pred `while()` `cond_wait`
  - Posledný pisateľ odchádza – ktorý je posledný?
    - počítat' si pisateľov
  - Ochrana oblasti pred viacerými pisateľmi
    - Ochrana mutexom
  - Zobúdzanie všetkých čakajúcich
    - `cond_broadcast` namiesto `cond_signal`
-

---

# Readers- Writers

## Pthreads – Writers Preference

### Init:

```
mutex_init(&mutex); mutex_init(&w_mutex);  
cond_init(&r_cond); r_count = 0;  
cond_init(&w_cond); w_count = 0;
```

### Thread Reader:

```
mutex_lock(&mutex);  
while (w_count != 0) cond_wait(&w_cond, &mutex);  
r_count++;  
mutex_unlock(&mutex);
```

```
...  
// reading  
...
```

```
mutex_lock(&mutex);  
r_count--;  
if (w_count == 0) cond_broadcast(&r_cond);  
mutex_unlock(&mutex);
```

### Thread Writer:

```
mutex_lock(&mutex);  
w_count++;  
while (r_count != 0) cond_wait(&r_cond, &mutex);  
mutex_unlock(&mutex);
```

```
mutex_lock(&w_mutex);  
// writing  
mutex_unlock(&w_mutex);
```

```
mutex_lock(&mutex);  
w_count--;  
cond_broadcast(&w_cond);  
mutex_unlock(&mutex);
```

---

# Readers- Writers

## ■ Symetrické riešenia?

### Thread Reader:

#### A – pôvodné riešenie

```
mutex_lock(mutex);  
while (w_count!=0) cond_wait(w_cond, mutex);  
r_count ++;  
mutex_unlock(mutex);
```

#### B – r\_count pred while

```
mutex_lock(mutex);  
r_count ++;  
while (w_count!=0) cond_wait(w_cond, mutex);  
mutex_unlock(mutex);
```

#### C – w\_count za while

```
mutex_lock(mutex);  
while (w_count!=0) cond_wait(w_cond, mutex);  
r_count ++;  
mutex_unlock(mutex);
```

### Thread Writer:

```
mutex_lock(mutex);  
w_count ++;  
while (r_count != 0) cond_wait(r_cond, mutex);  
mutex_unlock(mutex);
```

```
mutex_lock(mutex);  
w_count ++;  
while (r_count != 0) cond_wait(r_cond, mutex);  
mutex_unlock(mutex);
```

```
mutex_lock(mutex);  
while (r_count != 0) cond_wait(r_cond, mutex);  
w_count ++;  
mutex_unlock(mutex);
```

# Readers- Writers

## ■ Riešenia preferencie

### Thread Reader:

#### A – writers preference

```
mutex_lock(mutex);  
while (w_count!=0) cond_wait(w_cond, mutex);  
r_count ++;  
mutex_unlock(mutex);
```

#### C – no starving

```
mutex_lock(mutex);  
while (w_count!=0) cond_wait(w_cond, mutex);  
r_count ++;  
mutex_unlock(mutex);
```

#### D – readers preference

```
mutex_lock(mutex);  
r_count ++;  
while (w_count!=0) cond_wait(w_cond, mutex);  
mutex_unlock(mutex);
```

### Thread Writer:

```
mutex_lock(mutex);  
w_count ++;  
while (r_count != 0) cond_wait(r_cond, mutex);  
mutex_unlock(mutex);
```

```
mutex_lock(mutex);  
while (r_count != 0) cond_wait(r_cond, mutex);  
w_count ++;  
mutex_unlock(mutex);
```

```
mutex_lock(mutex);  
while (r_count != 0) cond_wait(r_cond, mutex);  
w_count ++;  
mutex_unlock(mutex);
```

---

# Synchronizácia

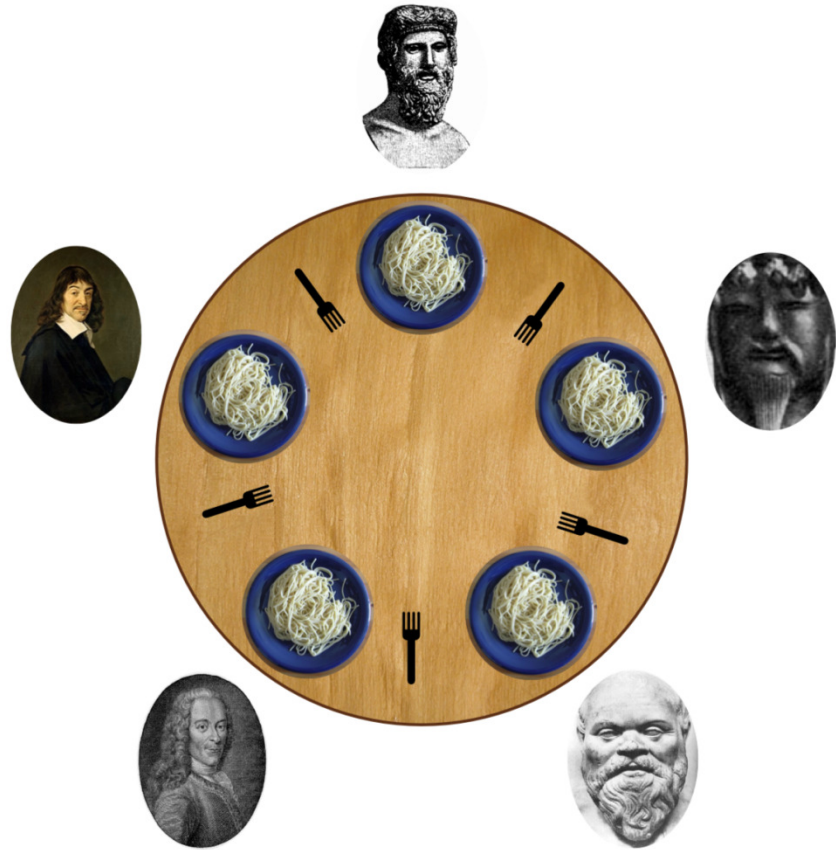
- Súťaženie - Race Hazard, Race Condition
  - Uviaznutie – Deadlock
  - Vyhladovanie – Starvation
-



---

# Dining Philosophers

```
while (true) {  
    think();  
    eat();  
}
```



---

# Dining Philosophers

- Filozof striedavo myslí a je
  - Ak chce myslieť – žiaden problém
  - Ak chce jesť potrebuje paličku (chopstick) po svojej pravej aj ľavej strane
  - Iba jeden filozof môže mať paličku v danej chvíli
-

---

# Dining Philosophers

- Nesmie nastat' uviaznutie
  - Nesmie nastat' vyhladovanie
  - Čas čakania na jedenie (koniec myslenia a začiatok jedenia) – čas blokovania nech je minimálny
  - Spravodlivý prístup k jedlu
-

---

# Dining Philosophers

```
while (true) {  
    think();  
  
    get_forks();  
    eat();  
    put_forks();  
}
```



---

# Dining Philosophers

## **Init:**

```
for (i=0;i<5;i++) sem_init(&forks[i],1);
```

## **Thread Philosopher Init:**

```
tid  
left = tid  
right = (tid+1) % 5
```

## **Thread Philosopher:**

```
get_forks()  
    sem_wait(&forks[right]);  
    sem_wait(&forks[left]);
```

```
put_forks()  
    sem_signal(&forks[right]);  
    sem_signal(&forks[left]);
```

---

---

# Dining Philosophers

- Jedna palička pre jedného filozofa – OK
- Môže nastať uviaznutie



---

# Dining Philosophers

- Blokovanie celého stola
  - Dlhý čas čakania na jedenie
-

---

# Dining Philosophers

## **Init:**

```
sem_init(&table, 1);
```

## **Thread Philosopher Init:**

## **Thread Philosopher:**

```
get_forks()  
    sem_wait(&table);
```

```
put_forks()  
    sem_signal(&table);
```

---



---

# Dining Philosophers

- Narušenie symetrie – jeden filozof ľavák
- Sused ľaváka – najlepšie šance získať paličku – nespravodlivé riešenie



---

# Dining Philosophers

## **Init:**

```
for (i=0; i<5; i++) sem_init(forks[i],1);
```

## **Thread Philosopher Init:**

```
tid
```

```
left = tid
```

```
right = (tid+1) % 5
```

## **Thread Philosopher:**

```
get_forks()
```

```
    sem_wait(forks[right]);
```

```
    sem_wait(forks[left]);
```

```
put_forks()
```

```
    sem_signal(forks[right]);
```

```
    sem_signal(forks[left]);
```

## **Thread s tid=0 Philosopher Leftie:**

```
get_forks()
```

```
    sem_wait(forks[left]);
```

```
    sem_wait(forks[right]);
```

```
put_forks()
```

```
    sem_signal(forks[left]);
```

```
    sem_signal(forks[right]);
```

---

---

# Dining Philosophers

- Ak povolíme iba 4 filozofov, aby sa snažili získať paličky, uviaznutie nenastane
  - Obsluha („Footman“) obsluhuje iba 4 filozofov
-

---

# Dining Philosophers

## **Init:**

```
sem_init(&footman,4);  
for (i=0; i<5; i++) sem_init(&forks[i],1);
```

## **Thread Philosopher Init:**

```
tid  
left = tid  
right = (tid+1) % 5
```

## **Thread Philosopher:**

```
get_forks()  
    sem_wait(&footman);  
    sem_wait(&forks[right]);  
    sem_wait(&forks[left]);  
  
put_forks()  
    sem_signal(&forks[right]);  
    sem_signal(&forks[left]);  
    sem_signal(&footman);
```

---

---

# Zdroje

- Allen B. Downey. The Little Book of Semaphores
  - Linux Man Pages. `sem_overview`
-

---

# Ako sa pripraviť na 1. úlohu

- Jednoduché príklady
    - vytváranie vlákien, mutex, semafor, podm. premenné, ...
  - Jednoduchšie synchronizačné problémy
    - Vylučovanie, signalizácia, barierová synchronizácia, znovupoužiteľná bariéra
    - Pomocou semaforov, pomocou mutexov + podm. premenných
  - Klasické synchronizačné problémy
    - Producent – konzument
    - Čitatelia – pisatelia
      - Uprednostnenie čitateľov
      - Uprednostnenie pisateľov
      - Bez vyhladovania
    - Obedujúci filozofi
-

---

# Témy na projekt

- Nájst' a spracovať synchronizačný problém
    - Popis problému – zadanie
    - Riešenie aj s chybami a postupnými opravami
    - Dokument + príklady v C
  - Vytvoriť simuláciu zvoleného synchronizačného problému
    - Projekt v C + dokument popisujúci použitie a riešenia
    - Oddeliť implementáciu problému od implementácie pomocných funkcií
    - Viaceré riešenia
  - Vytvoriť simulátor na overenie riešení (inšpirácia simulátor Sync)
-