Synchronization

Producers - Consumers (semaphores)

Init:

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

Thread Producer:

item = produce_item()

sem_wait(mutex)
fifo_add(buffer, item)
sem_signal(items)
sem_signal(mutex)

Thread Consumer:

sem_wait(items)
sem_wait(mutex)
item = fifo_remove(buffer)
sem_signal(mutex)

process_item(item)

Producers - Consumers
(condition variables)

Init:

fifo_init(buffer)
cond_init(cond)
mutex_init(mutex)

Thread Producer:

item = produce_item();
mutex_lock(mutex);
fifo_add(buffer, item);
cond_signal(cond);
mutex_unlock(mutex);

Thread Consumer:

mutex_lock(items);
mutex_lock(mutex);
while (fifo_isempty(buffer)) cond_wait(cond, mutex)
item = fifo_remove(buffer);
mutex_unlock(mutex);

process_item(item);

Readers - Writers
(semaphores, starvation)

Init:

sem_init(mutex, 1)
sem_init(room_empty, 1)
r_count = 0

Thread Reader:

sem_wait(mutex)
if (r_count == 0) sem_wait(room_empty)
r_count++
sem_signal(mutex)

reading()

sem_wait(mutex)
r_count—-
if (r_count == 0) sem_signal(room_empty)
sem_signal(mutex)

Thread Writer:

sem_wait(room_empty)

writing()

sem_signal(room_empty)

Readers - Writers Problem (semaphores, no starvation)

Init:

```
sem_init(mutex, 1)
sem_init(room_empty, 1)
sem_init(turnstile, 1)
r_count = 0
```

Thread Reader:

```
sem_wait(turnstile)
sem_signal(turnstile)

sem_wait(mutex)
if (r_count == 0) sem_wait(room_empty)
r_count++
sem_signal(mutex)

reading()

sem_wait(mutex)
r_count—-
if (r_count == 0) sem_signal(room_empty)
sem_signal(mutex)
```

Thread Writer:

```
sem_wait(turnstile)

sem_wait(room_empty)

writing()

sem_signal(room_empty)

sem_signal(turnstile)
```

Readers - Writers Problem
(semaphores, writers preference)

Init:

sem_init(r_mutex, 1)
sem_init(w_mutex, 1)
sem_init(room_empty, 1)
sem_init(hall_empty, 1)
r_count = 0
w_count = 0

Thread Reader:

sem_wait(hall_empty)
sem_wait(r_mutex)
if (r_count == 0) sem_wait(room_empty)
r_count++
sem_signal(r_mutex)
sem_signal(hall_empty)

reading()

sem_wait(r_mutex)
r_count—-
if (r_count == 0) sem_signal(room_empty)
sem_signal(r_mutex)

Thread Writer:

sem_wait(w_mutex)
if (w_count == 0) sem_wait(hall_empty)
w_count++
sem_signal(w_mutex)

sem_wait(room_empty)
writing()
sem_wait(room_empty)

sem_wait(w_mutex)
w_count—-
if (w_count == 0) sem_signal(hall_empty)
sem_signal(w_mutex)

Readers - Writers Problem

(condition variables, readers preference)

Init:

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

Thread Reader:

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

reading()

mutex_lock(mutex);
r_count—-
cond_signal(w_cond)
mutex_unlock(mutex);
```

Thread Writer:

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

writing()

mutex_lock(mutex);
w_count—-
cond_signal(w_cond)
cond_signal(r_cond)
mutex_unlock(mutex);
```

Readers - Writers Problem

(condition variables, writers preference)

Init:

```
mutex_init(mutex)
mutex_init(w_mutex)
cond_init(r_cond)
cond_init(w_cond)
r_count = 0
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—-
cond_signal(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_signal(w_cond)
mutex_unlock(mutex);
```

Readers - Writers Custom Problem
(semaphores, readers preference, two writers can write simultaneously)

Init:

```
fifo_init(writers)
sem_init(reader, 0)
sem_init(mutex, 1)
r_count = 0
w_in_count = 0
```

Thread Reader Init:

Thread Reader:

```
sem_wait(mutex)
if (r_count == 0 && w_in_count == 0) {
 sem_signal(reader)
}
r_count++
sem_signal(mutex)

sem_wait(reader)
sem_signal(reader)

reading()

sem_wait(mutex)
r_count—-
if (r_count == 0) {
 sem_wait(reader)
 if (!fifo_isempty(writers)) {
  w_in_count++
  sem_signal(fifo_remove(writers))
 }
}
sem_signal(mutex)
```

Thread Writer Init:

```
tid
sem_init(writer[tid], 0)
```

Thread Writer:

```
sem_wait(mutex)
if (r_count != 0 || w_in_count == 2) {
 fifo_add(writers, writer[tid])
}
else {
 w_in_count++
 sem_signal(writer[tid])
}
sem_signal(mutex)

sem_wait(writer[tid])

writing()

sem_wait(mutex)
w_in_count—-
if (r_count == 0 && !fifo_isempty(writers)) {
 w_in_count++
 sem_signal(fifo_remove(writers))
}
else if (r_count > 0 && w_in_count == 0) {
 sem_signal(reader)
}
sem_signal(mutex)
```

Readers - Writers Custom Problem
(semaphores, readers preference, two writers can write simultaneously)

Init:

```
sem_init(reader, 0)
sem_init(writer, 0)
sem_init(mutex, 1)
r_count = 0
w_count = 0
w_in_count = 0
r_in_count = 0
```

Thread Reader:

```
sem_wait(mutex)
r_count++
if (w_in_count == 0) {
 r_in_count++
 sem_signal(reader)
}
sem_signal(mutex)
sem_wait(reader)

reading()

sem_wait(mutex)
r_count--
r_in_count--
if (r_count == 0 && w_count > w_in_count) {
   w_in_count++
   sem_signal(writer)
 }
}
sem_signal(mutex)
```

Thread Writer:

```
sem_wait(mutex)
w_count++
if (r_count == 0 && w_in_count < 2) {
 w_in_count++
 sem_signal(writer)
}
sem_signal(mutex)
sem_wait(writer)

writing()

sem_wait(mutex)
w_count--
w_in_count--
if (r_count == 0 && w_count > w_in_count) {
 w_in_count++
 sem_signal(writer)
}
else if (r_count > 0 && w_in_count == 0) {
 while (r_count > r_in_count) {
   r_in_count++
   sem_signal(reader)
 }
}
sem_signal(mutex)
```

```
sem_init(reader, 0); // semafor pre writer vlakna
sem_init(writer, 0); // semafor pre writer vlakna
sem_init(mutex, 1);  // mutex pre kriticku sekciu, kde sa pracuje s pocitadlami
r_count = 0;       // pocet reader vlakien co chcu citat
w_count = 0;       // pocet writer vlakien co chcu pisat
w_in_count = 0;    // pocet reader vlakien co citaju
r_in_count = 0;    // pocet writer vlakien co zapisuju

/*
Reader vlakno chce citat, inkrementuje preto r_count. Ak ziadne writer vlakno nezapisuje,
moze ist citat, inkrementuje preto r_in_count a inkrementuje semafor reader, ktory po
opusteni kritickej sekcie dekrementuje. Ak nejake writer vlakna zapisuju, po opusteni
kritickej sekcie zacne cakat na semafore reader.
*/
sem_wait(mutex);
r_count++;
if (w_in_count == 0) {
 r_in_count++;
 sem_signal(reader);
}
sem_signal(mutex);
sem_wait(reader);

// Reader vlakno cita
reading()

/*
Reader vlakno prestalo citat, dekrementuje preto r_count a r_in_count. Ak ziadne reader vlakno nechce citat a nejake writer vlakno chce zapisovat (cize caka na semafore writer), je potrebne ho pustit.
Reader vlakno preto inkrementuje pocet writer vlakien co zapisuju a inkrementuje semafor writer.
*/
sem_wait(mutex);
r_count--;
r_in_count--;
if (r_count == 0 && w_count > w_in_count) {
  w_in_count++
  sem_signal(writer)
 }
}
sem_signal(mutex);
```

```
/*
Writer vlakno chce zapisovat, inkrementuje preto w_count. Ak ziadne reader vlakno nechce citat, a zapisuju menej ako 2 writer vlakna, moze ist zapisovat. Inkrementuje preto w_in_count a inkrementuje semafor writer, ktory po opusteni kritickej sekcie dekrementuje. Ak vlakno nemoze ist zapisovat, po opusteni kritickej sekcie zacne cakat na semafore writer.
*/
sem_wait(mutex);
w_count++;
if (r_count == 0 && w_in_count < 2) {
 w_in_count++;
 sem_signal(writer);
}
sem_signal(mutex);
sem_wait(writer);

// Writer vlakno zapisuje
writing();

/*
Writer vlakno prestalo zapisovat, dekrementuje preto w_count a w_in_count. Ak ziadne reader vlakno nechce citat a nejake writer vlakno chce zapisovat (cize caka na semafore writer), je potrebne ho pustit. Writer vlakno preto inkrementuje pocet writer vlakien co zapisuju a inkrementuje semafor writer.
*/
sem_wait(mutex);
w_count--;
w_in_count--;
if (r_count == 0 && w_count > w_in_count) {
 w_in_count++;
 sem_signal(writer);
}
/*
Ak ale nejake reader vlakna chcu citat (cize cakaju na semafore reader) a ziadne writer vlakno nezapisuje, je potrebne ich vs etky spustit. Writer vlakno preto inkrementuje pocet reader vlakien co citaju a inkrementuje semafor reader tolko krat, kolko reader vlakien cakalo (r_count - r_in_count).
*/
else if (r_count > 0 && w_in_count == 0) {
 while (r_count > r_in_count) {
  r_in_count++;
  sem_signal(reader);
 }
}
sem_signal(mutex);
```