Operating Systems and Concurrency

Concurrency 6 + Revision
COMP2007

Dan Marsden
(Geert De Maere)
{Geert.DeMaere,Dan.Marsden}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2023

©University of Nottingham

Goals

Today

- Writers before readers.
- Beyond locks - alternative approaches to concurrency.

The Readers – Writers Problem (Last Time)

Solution 2: Readers First

- We implemented as solution to readers-writers that allowed concurrent reading when safe to do so.
- Recall the **room with a light switch analogy** - writers only enter a darkened room, readers happy to mix together.
- Unfortunately readers can overwhelm writers, and the **writers can starve**.
- Today - try to reverse the situation and **favour the writers**.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First

Solution 3 gives priority to **writers** and uses:

- Integers `iReadCount` and `iWriteCount` to keep track of the number of readers/writers.
- Mutexes `sRead` and `sWrite` to synchronise the **reader's/writer's critical section**.
- Semaphore `sReadTry` to stop readers when there is **a writer waiting**.
- Semaphore `sResource` to **synchronise** the resource for **reading/writing**.

©University of Nottingham

The Readers – Writers Problem

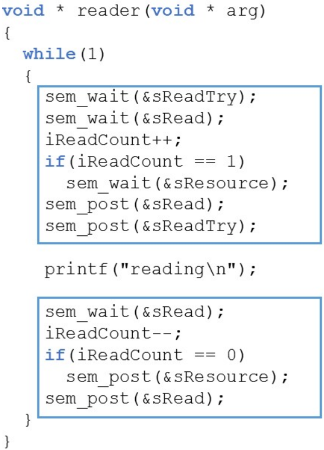
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

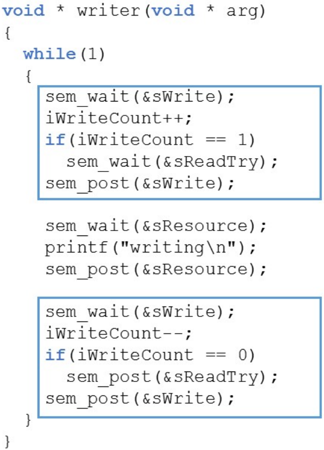
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

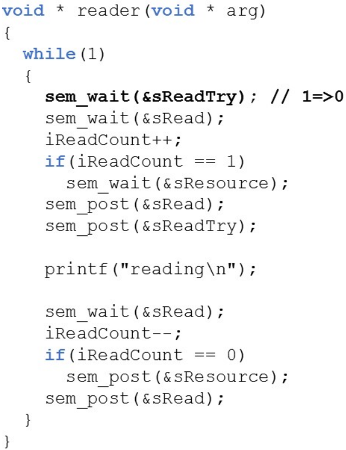


```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

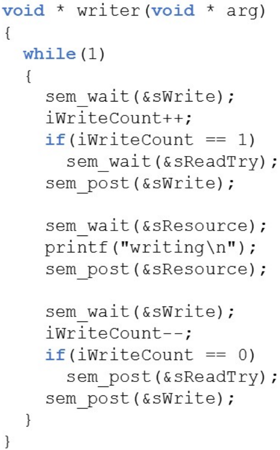
©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First



```c
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry); // 1=>0
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```c
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead); // 1=>0
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;// 0=>1
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

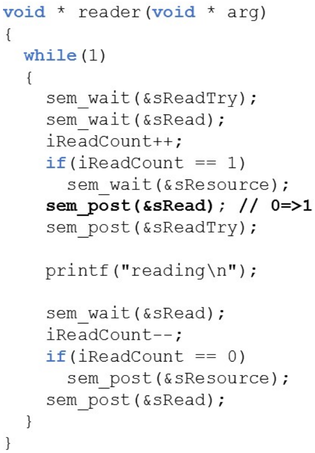
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource); // 1=>0
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

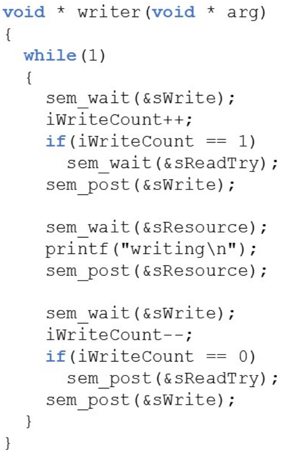
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead); // 0=>1
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

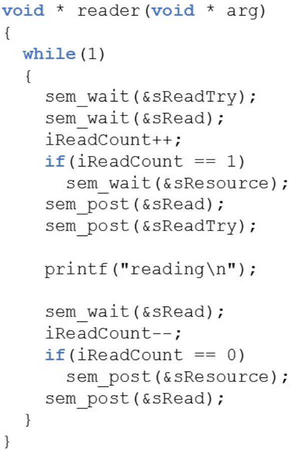
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);// 0=>1

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

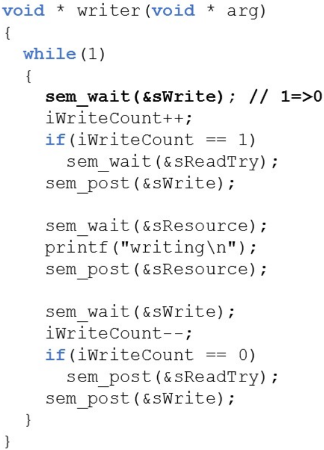
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite); // 1=>0
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++; // 0=>1
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry); // 1=>0
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

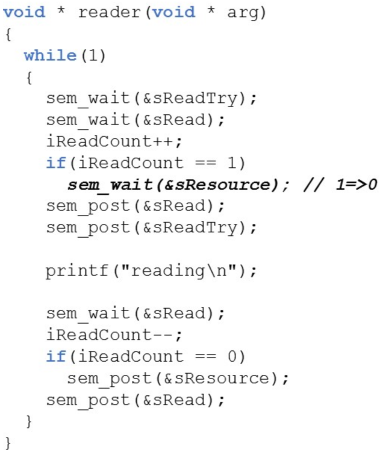
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

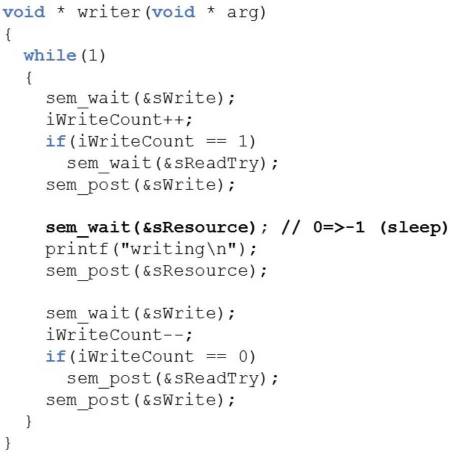
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite); // 0=>1

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource); // 1=>0
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource); // 0=>-1 (sleep)
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead); // 1=>0
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--; // 1=>0
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource); // -1=>0
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource); // wakeup
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

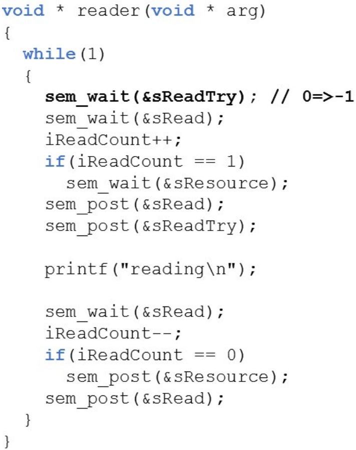
```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead); // 0=>1
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

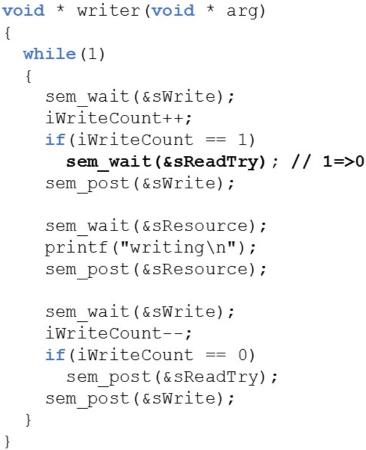
©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First



```c
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry); // 0=>-1
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```c
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry); // 1=>0
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

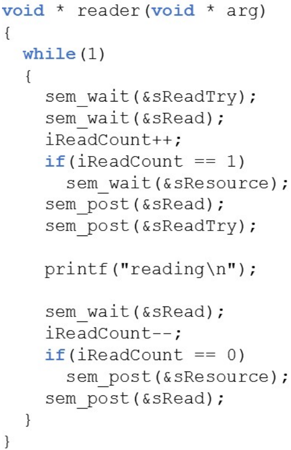
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

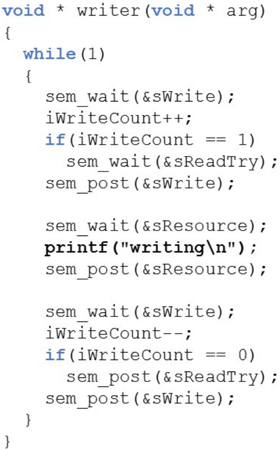
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource); // (woken up)
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



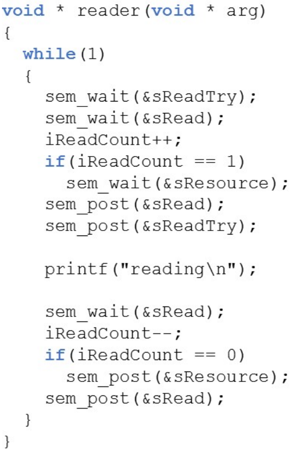
```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



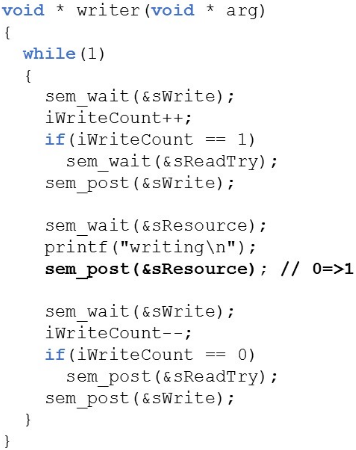
```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First



```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```



```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource); // 0=>1

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite); // 1=>0
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--; // 1=>0
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite);
  }
}
```

The Readers – Writers Problem

Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);  // wakeup
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry); // -1=>0
    sem_post(&sWrite);
  }
}
```

©University of Nottingham

The Readers – Writers Problem

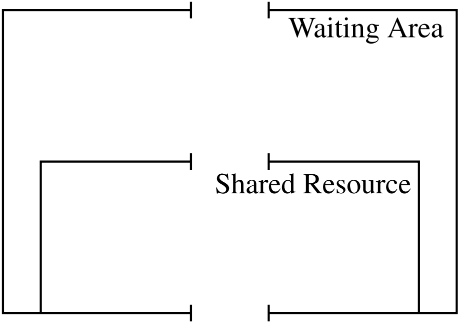
Solution 3: Writers First

```
void * reader(void * arg)
{
  while(1)
  {
    sem_wait(&sReadTry);
    sem_wait(&sRead);
    iReadCount++;
    if(iReadCount == 1)
      sem_wait(&sResource);
    sem_post(&sRead);
    sem_post(&sReadTry);

    printf("reading\n");

    sem_wait(&sRead);
    iReadCount--;
    if(iReadCount == 0)
      sem_post(&sResource);
    sem_post(&sRead);
  }
}
```

```
void * writer(void * arg)
{
  while(1)
  {
    sem_wait(&sWrite);
    iWriteCount++;
    if(iWriteCount == 1)
      sem_wait(&sReadTry);
    sem_post(&sWrite);

    sem_wait(&sResource);
    printf("writing\n");
    sem_post(&sResource);

    sem_wait(&sWrite);
    iWriteCount--;
    if(iWriteCount == 0)
      sem_post(&sReadTry);
    sem_post(&sWrite); // 0=>1
  }
}
```

The Readers – Writers Problem

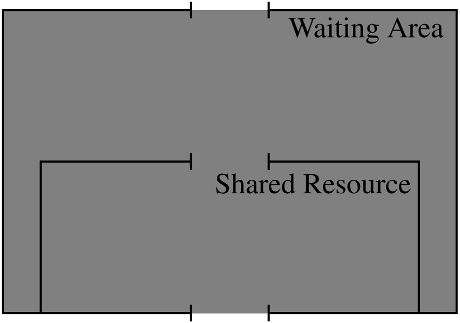
Solution 3: Writers First - Intuitions



©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

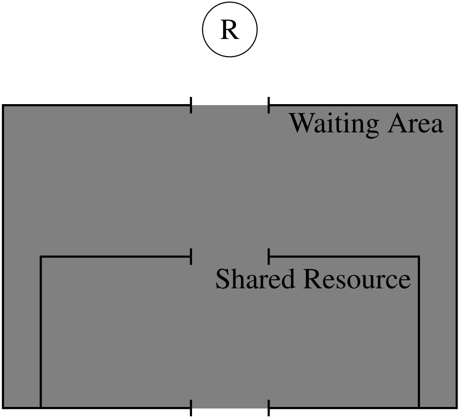


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

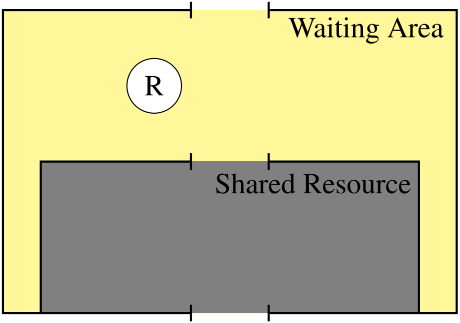


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

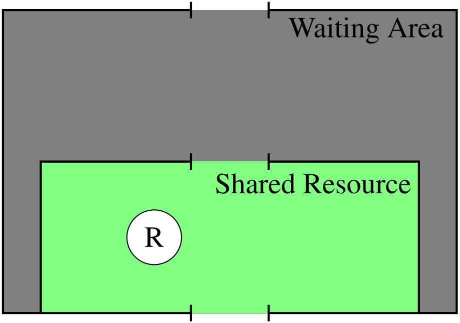


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

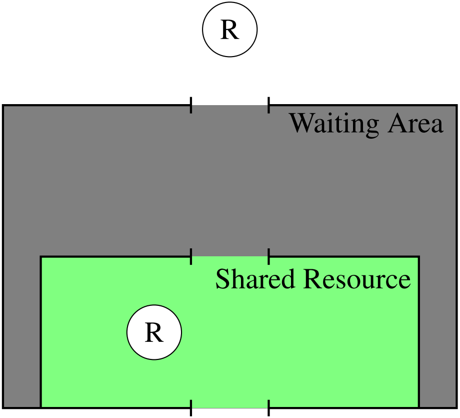


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

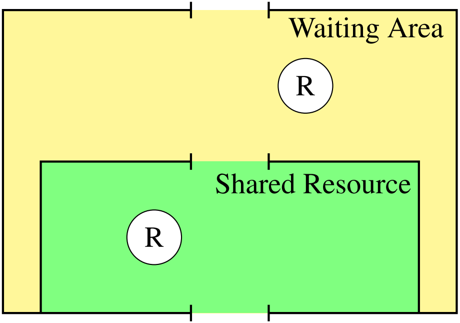


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

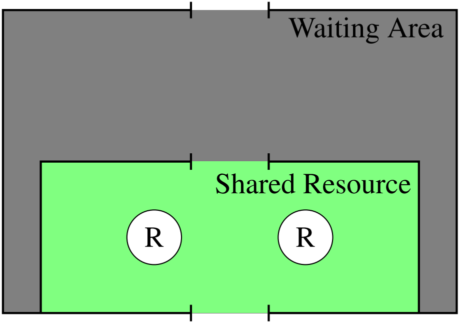


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



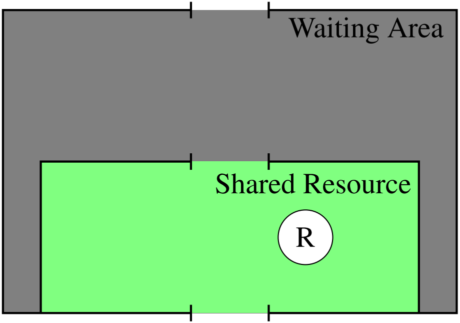
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



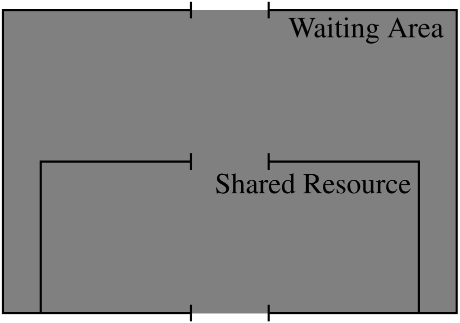
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

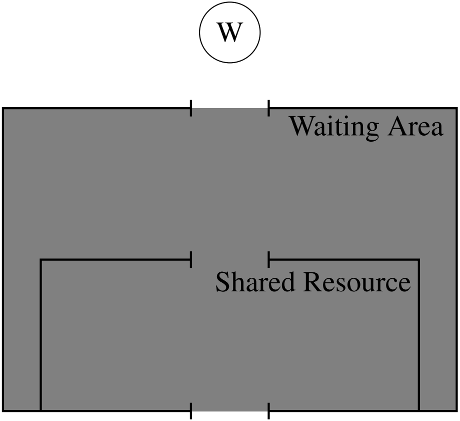


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



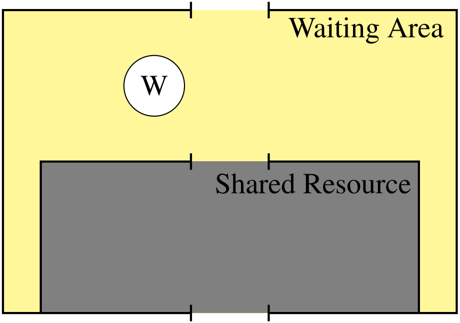
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

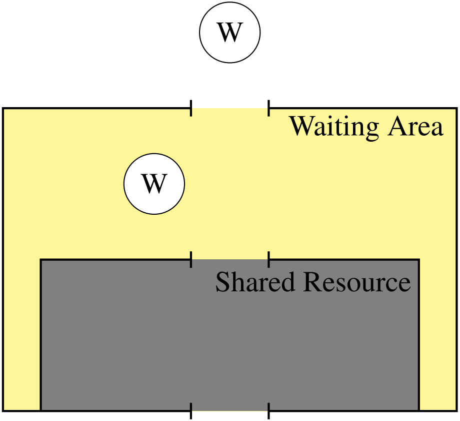


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



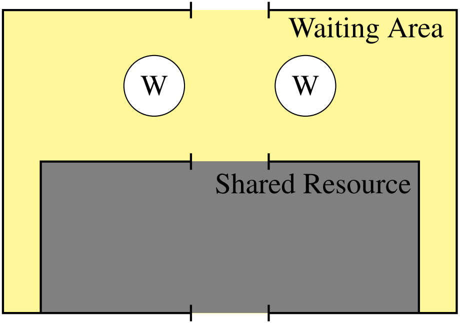
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



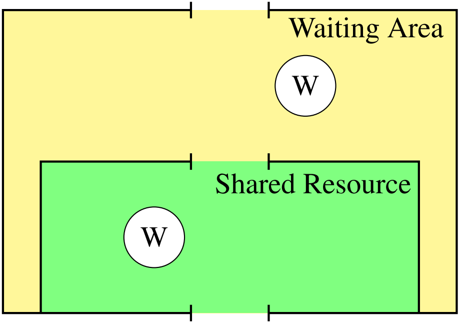
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



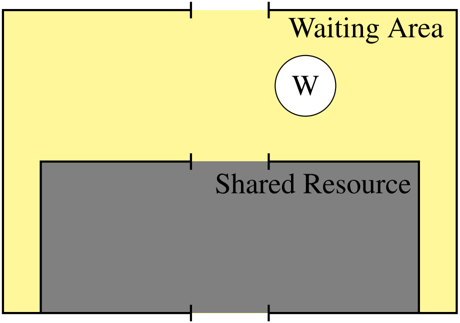
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

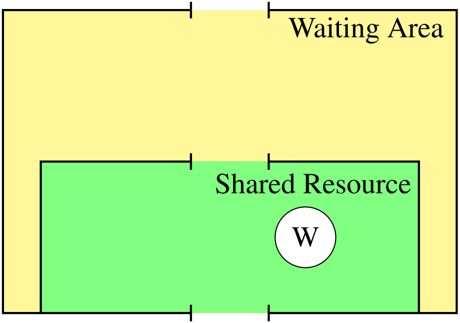


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



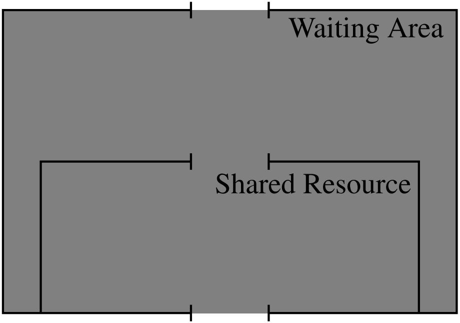
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



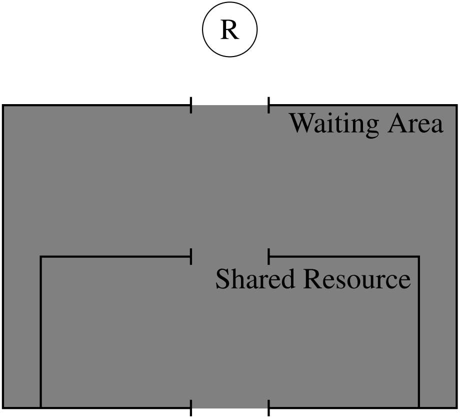
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

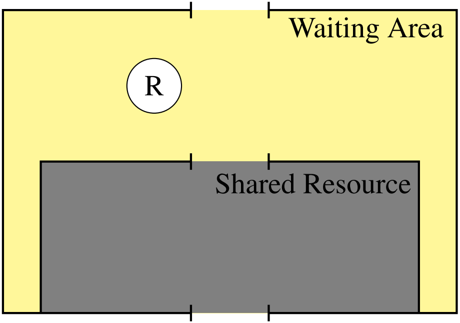


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



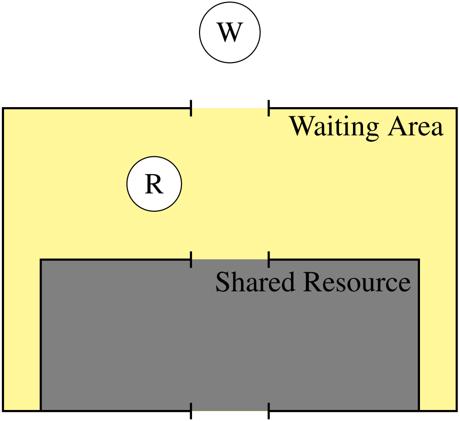
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

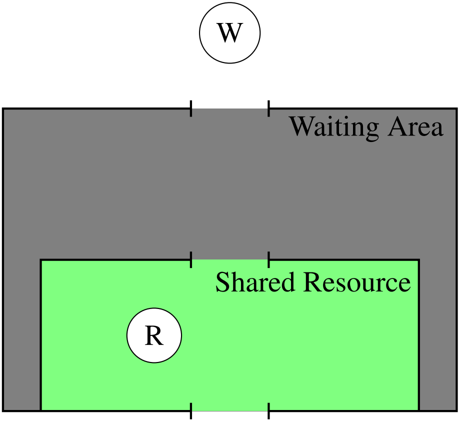


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



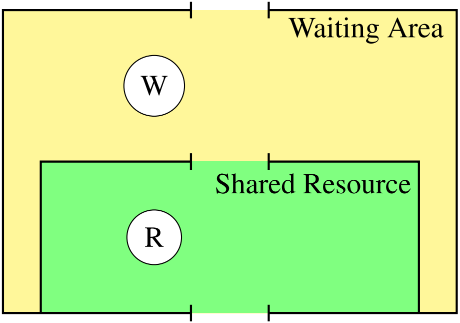
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



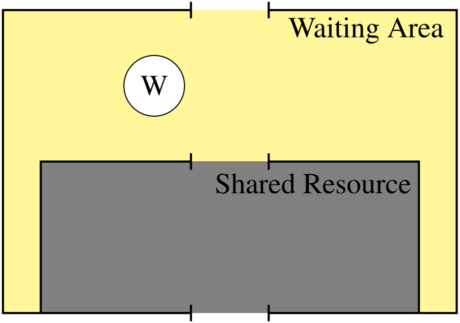
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



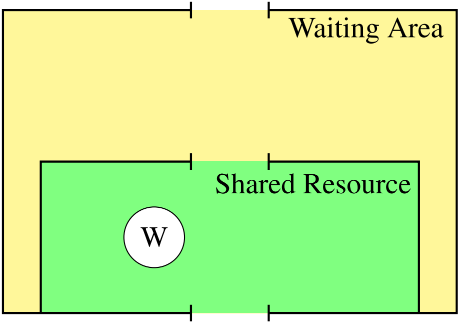
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

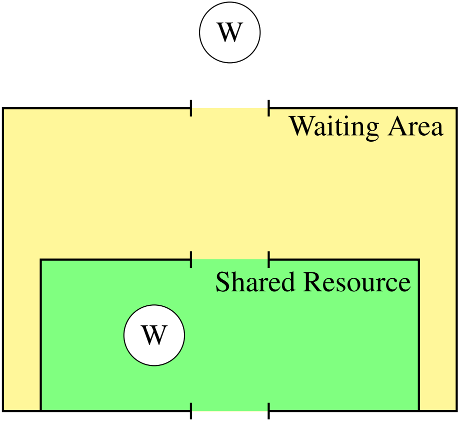


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions

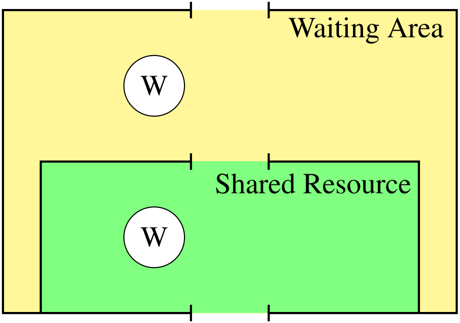


Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don't like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



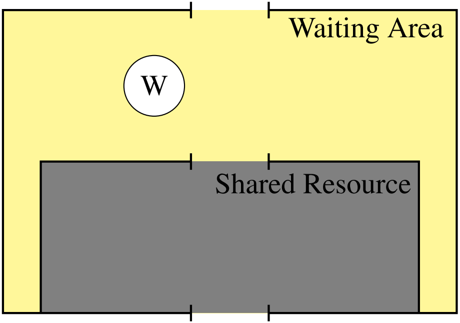
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



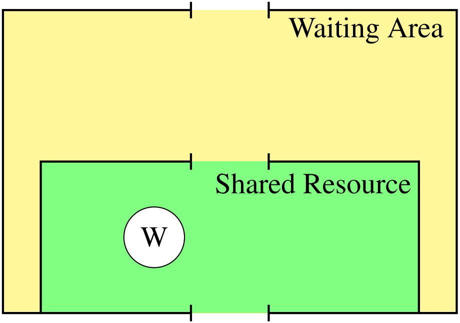
Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

©University of Nottingham

The Readers – Writers Problem

Solution 3: Writers First - Intuitions



Constraints:

- Everyone happy to go into a room with the light off.
- Readers and writers don’t like each other.
- Readers happy to enter a room with a green light on, but not a yellow one.
- Writers happy to enter a room with a yellow light on, but not a green one.
- Only readers are allowed to use the bottom exit.

The trouble with locks

Locking style coordination of threads, such as with mutexes and semaphores has multiple problems. For example:

- **Deadlocks**.
- **Starvation**.
- **Priority inversion**.

The trouble with locks

Locking style coordination of threads, such as with mutexes and semaphores has multiple problems. For example:

- **Deadlocks**.
- **Starvation**.
- **Priority inversion**.
- Lack of **compositionality** - you cannot compose small correct concurrent programs to form large correct concurrent programs.

The trouble with locks

Locking style coordination of threads, such as with mutexes and semaphores has multiple problems. For example:

- **Deadlocks**.
- **Starvation**.
- **Priority inversion**.
- Lack of **compositionality** - you cannot compose small correct concurrent programs to form large correct concurrent programs.
- We could try to grab locks and back-off if we cannot acquire them all - we risk another problem - **livelock**.

The trouble with locks

Locking style coordination of threads, such as with mutexes and semaphores has multiple problems. For example:

- **Deadlocks**.
- **Starvation**.
- **Priority inversion**.
- Lack of **compositionality** - you cannot compose small correct concurrent programs to form large correct concurrent programs.
- We could try to grab locks and back-off if we cannot acquire them all - we risk another problem - **livelock**.
- **Access to shared state is implicit** - it can be hard to tell who's modifying what, and if suitable locks are held at the time.
- **Lock convoying** - threads end up queuing up behind a thread holding a lock - linearizes behaviour and can take a long time to clear.

Alternative Concurrency Approaches

We should not just be satisfied with such a status-quo. Therefore, we shall look at two alternatives:

Alternative Concurrency Approaches

We should not just be satisfied with such a status-quo. Therefore, we shall look at two alternatives:

1. Message passing - **synchronous** or **asynchronous**.

Alternative Concurrency Approaches

We should not just be satisfied with such a status-quo. Therefore, we shall look at two alternatives:

1. Message passing - **synchronous** or **asynchronous**.
2. **Transactional memory**.

Message Passing

Synchronous Message Passing

- We avoid sharing memory, and instead have have channels $c, d, \ldots$
- There are two operations on channel c:
  1. $c(x)$ sends message x over the channel.
  2. $\overline{c}(x)$ receives a value over the channel into x.
- Communication is **synchronous** - a send must wait for a corresponding receive and vice-versa. **Think phone calls**.

©University of Nottingham

Message Passing

Synchronous Message Passing

- We avoid sharing memory, and instead have have channels $c, d, \ldots$
- There are two operations on channel c:
  1. $c(x)$ sends message x over the channel.
  2. $\overline{c}(x)$ receives a value over the channel into x.
- Communication is **synchronous** - a send must wait for a corresponding receive and vice-versa. **Think phone calls**.

Thread 1

$c(data)$

Thread 2

$\ldots$

Message Passing

Synchronous Message Passing

- We avoid sharing memory, and instead have have channels $c, d, \ldots$
- There are two operations on channel c:
  1. $c(x)$ sends message x over the channel.
  2. $\overline{c}(x)$ receives a value over the channel into x.
- Communication is **synchronous** - a send must wait for a corresponding receive and vice-versa. **Think phone calls**.

Thread 1	**Thread 2**
$c(data)$	$\ldots$
blocked	$\ldots$

Message Passing

Synchronous Message Passing

- We avoid sharing memory, and instead have have channels $c, d, \ldots$
- There are two operations on channel c:
  1. $c(x)$ sends message x over the channel.
  2. $\overline{c}(x)$ receives a value over the channel into x.
- Communication is **synchronous** - a send must wait for a corresponding receive and vice-versa. **Think phone calls**.

Thread 1	**Thread 2**
$c(data)$	$\ldots$
blocked	$\ldots$
blocked	$\overline{c}(x)$

Message Passing

Synchronous Message Passing

- We avoid sharing memory, and instead have have channels $c, d, \ldots$
- There are two operations on channel c:
  1. $c(x)$ sends message x over the channel.
  2. $\overline{c}(x)$ receives a value over the channel into x.
- Communication is **synchronous** - a send must wait for a corresponding receive and vice-versa. **Think phone calls**.

Thread 1	**Thread 2**
$c(data)$	$\ldots$
blocked	$\ldots$
blocked	$\overline{c}(x)$
$\ldots$	$\ldots \quad (x = data)$

Message Passing

Synchronous Message Passing

Pros and cons of synchronous message passing:

- Flow of data is **explicit** - no exposure to race conditions and other difficult debugging.
- Synchronizing senders and receivers can slow performance.

Message Passing

Synchronous Message Passing

Pros and cons of synchronous message passing:

- Flow of data is **explicit** - no exposure to race conditions and other difficult debugging.
- Synchronizing senders and receivers can slow performance.
- Can implement mutexes and semaphores using synchronous message passing - **deadlocks, starvation etc. all still possible**.

Message Passing

Asynchronous Message Passing

In an attempt to mitigate the issues of synchronous message passing, **senders do not block** waiting for an available receiver. **Think sending an email**. Often use what is known as an **Actor** based model.

[1] Example adapted from notes of Ian Stark

©University of Nottingham

Message Passing

Asynchronous Message Passing

In an attempt to mitigate the issues of synchronous message passing, **senders do not block** waiting for an available receiver. **Think sending an email**.

Often use what is known as an **Actor** based model.

For example, to specify a simple counter [1]:

```
class Counter extends Actor {
  var counter = 0

  def receive = {
    case Zero => counter = 0
    case Inc => counter = counter + 1
  }
}
```

[1] Example adapted from notes of Ian Stark

Message Passing

Asynchronous Message Passing

In an attempt to mitigate the issues of synchronous message passing, **senders do not block** waiting for an available receiver. **Think sending an email**.

Often use what is known as an **Actor** based model.

For example, to specify a simple counter [1]:

```
class Counter extends Actor {
  var counter = 0

  def receive = {
    case Zero => counter = 0
    case Inc => counter = counter + 1
  }
}
```

Then to increment a `Counter` object, you just send it a message:

```
mycounter ! Inc // Send Inc message - don't wait for receiver
mycounter ! Inc // Send another - queued up until dealt with
```

[1] Example adapted from notes of Ian Stark

Message Passing

Asynchronous Message Passing

Pros and cons of synchronous message passing:

- As with the synchronous model, flow of data is **explicit** - less exposure to race conditions and other difficult debugging.
- Less **explicit** coupling between senders and receivers.
- *May* be less efficient than shared memory.

©University of Nottingham

Message Passing

Asynchronous Message Passing

Pros and cons of synchronous message passing:

- As with the synchronous model, flow of data is **explicit** - less exposure to race conditions and other difficult debugging.
- Less **explicit** coupling between senders and receivers.
- *May* be less efficient than shared memory.
- Can simulate synchronous message passing.

©University of Nottingham

Message Passing

Asynchronous Message Passing

Pros and cons of synchronous message passing:

- As with the synchronous model, flow of data is **explicit** - less exposure to race conditions and other difficult debugging.
- Less **explicit** coupling between senders and receivers.
- *May* be less efficient than shared memory.
- Can simulate synchronous message passing.
- Can implement mutexes using asynchronous message passing - **deadlocks, starvation etc. all still possible**.

Transactional Memory

A Database Analogy

Imagine I owe Geert £10.

- Inside the bank, our account balances might be stored in a database table:

Holder	**Balance**
Geert	10000000
Dan	15

- Transferring the money to Geert requires two steps:
  1. Deduct £10 from the Dan account.
  2. Add £10 to the Geert account.

©University of Nottingham

Transactional Memory

A Database Analogy

Requirements for the bank transfer

- We would like the transfer to be **atomic** - In particular it either completely succeeds or totally fails.
- If other (atomic) changes are being made to the database, we would like them to be **serializable** - as if they don't overlap in time.

©University of Nottingham

Transactional Memory

Database Transactions

To satisfy our requirements, we wrap our changes in a **transaction**:

```
begin transaction;
update accounts set balance = balance + 10 where holder = 'Geert';
update accounts set balance = balance - 10 where holder = 'Dan';
commit;
```

Either both updates take place, or neither.

Transactional Memory

Database Transactions

To satisfy our requirements, we wrap our changes in a **transaction**:

```
begin transaction;
update accounts set balance = balance + 10 where holder = 'Geert';
update accounts set balance = balance - 10 where holder = 'Dan';
commit;
```

Either both updates take place, or neither.

Rollback

Realistic transactions may be more complex, and can encounter reasons they cannot succeed. Instead of `commit` they explicitly call `rollback` to fail the whole transaction.

©University of Nottingham

Transactional Memory

Transactions for thread coordination

The idea of transactional memory is to have the memory shared between threads behave transactionally, just like a database does.

```
do {
  begin_transaction();
  modify_shared_data();
  commit();
} while(!transaction_succeeds());
```

Transactional Memory

Transactions for thread coordination

The idea of transactional memory is to have the memory shared between threads behave transactionally, just like a database does.

```
do {
  begin_transaction();
  modify_shared_data();
  commit();
} while(!transaction_succeeds());
```

- No locks, and **no risk of deadlock**.

Transactional Memory

Transactions for thread coordination

The idea of transactional memory is to have the memory shared between threads behave transactionally, just like a database does.

```
do {
  begin_transaction();
  modify_shared_data();
  commit();
} while(!transaction_succeeds());
```

- No locks, and **no risk of deadlock**.
- A form of **livelock** still possible - as we back-off and try again after rollbacks.

©University of Nottingham

Transactional Memory

Transactions for thread coordination

The idea of transactional memory is to have the memory shared between threads behave transactionally, just like a database does.

```
do {
  begin_transaction();
  modify_shared_data();
  commit();
} while(!transaction_succeeds());
```

- No locks, and **no risk of deadlock**.
- A form of **livelock** still possible - as we back-off and try again after rollbacks.
- **Starvation** still possible - smaller transactions may cause repeated rollbacks of longer ones.

Thanks!

Change of lecturer back to Geert and a new topic on Wednesday!

©University of Nottingham