Operating systems and concurrency B09

David Kendall

Northumbria University

Introduction

- Semaphores can cause problems if not used carefully
- We will consider:
  - Deadlock
  - Starvation
  - Priority inversion (next time)

A mistake in the producer/consumer solution

Producer (pseudo-code)

```
1  while (true) {
2
3    // produce an item
4
5    wait (bufMutex);
6    wait (emptySlot);
7
8    // add item to buffer
9
10   post (bufMutex);
11   post (fullSlot);
12 }
```

Consumer (pseudo-code)

```
1  while (true) {
2    wait (fullSlot);
3    wait (bufMutex);
4
5    // remove item from buffer
6
7    post (bufMutex);
8    post (emptySlot);
9
10   // consume the item
11
12 }
```

- Spot the difference with the correct solution!

A mistake in the producer/consumer solution

Producer (pseudo-code)

```
1  while (true) {
2
3    // produce an item
4
5    wait (bufMutex);
6    wait (emptySlot);
7
8    // add item to buffer
9
10   post (bufMutex);
11   post (fullSlot);
12 }
```

Consumer (pseudo-code)

```
1  while (true) {
2    wait (fullSlot);
3    wait (bufMutex);
4
5    // remove item from buffer
6
7    post (bufMutex);
8    post (emptySlot);
9
10   // consume the item
11
12 }
```

- Spot the difference with the correct solution!
- Lines 5 and 6 in the producer are the wrong way round

What can go wrong?

- Imagine...
  - the buffer is full
  - a producer wants to add to the buffer
  - the producer acquires the `bufMutex`
  - the producer is suspended by `wait(emptySlot)`
  - a consumer tries to remove an item from the buffer
  - the consumer acquires `fullSlot` successfully
  - the consumer is suspended by `wait(bufMutex)`
- What now?

What can go wrong?

- The producer will be blocked until the `emptySlot` semaphore is posted by the consumer
- The consumer will be blocked until the `bufMutex` semaphore is posted by the producer
- The producer can't post `bufMutex` until the consumer posts `emptySlot`
- The consumer can't post `emptySlot` until the producer posts `bufMutex`
- DEADLOCK

Necessary conditions for deadlock

1. Mutual exclusion
   - some resources cannot be shared
2. Hold and wait
   - tasks can hold non-shareable resources while waiting to acquire other non-shareable resources
3. No pre-emption
   - once a task is holding a resource, the resource can't be taken away from it
4. Circular wait
   - task T_0 can wait for a resource held by task T_1 that is waiting for a resource held by task T_2, ..., that is waiting for a resource held by task T_n that is waiting for a resource held by task T_0

A problem with the readers/writers solution

Writers (pseudo-code)

```
1 while (true) {
2   wait(writeMutex);
3
4  // write the data
5
6   post(writeMutex);
7
8  // do non-critical stuff
9 }
```

Readers (pseudo-code)

```
1  while (true) {
2    wait(nReadersMutex);
3    nReaders += 1;
4    if (nReaders == 1) {
5      wait(writeMutex);
6    }
7    post(nReadersMutex);
8
9    // read the data
10
11   wait(nReadersMutex);
12   nReaders -= 1;
13   if (nReaders == 0) {
14     post(writeMutex);
15   }
16
17   // do non-critical stuff
18 }
```

A problem with the readers/writers solution

Writers (pseudo-code)

```
1 while (true) {
2   wait(writeMutex);
3
4  // write the data
5
6   post(writeMutex);
7
8  // do non-critical stuff
9 }
```

- What happens to writers if a reader always arrives before the last reader finishes?

Readers (pseudo-code)

```
1  while (true) {
2    wait(nReadersMutex);
3    nReaders += 1;
4    if (nReaders == 1) {
5      wait(writeMutex);
6    }
7    post(nReadersMutex);
8
9    // read the data
10
11   wait(nReadersMutex);
12   nReaders -= 1;
13   if (nReaders == 0) {
14     post(writeMutex);
15   }
16
17   // do non-critical stuff
18 }
```

A problem with the readers/writers solution

Writers (pseudo-code)

```
1 while (true) {
2   wait(writeMutex);
3
4  // write the data
5
6   post(writeMutex);
7
8  // do non-critical stuff
9 }
```

- What happens to writers if a reader always arrives before the last reader finishes?
- STARVATION

Readers (pseudo-code)

```
1  while (true) {
2    wait(nReadersMutex);
3    nReaders += 1;
4    if (nReaders == 1) {
5      wait(writeMutex);
6    }
7    post(nReadersMutex);
8
9    // read the data
10
11   wait(nReadersMutex);
12   nReaders -= 1;
13   if (nReaders == 0) {
14     post(writeMutex);
15   }
16
17   // do non-critical stuff
18 }
```

Acknowledgements

- Silberschatz, Galvin, Gagne, *Operating System Concepts*, John Wiley, 2013