Operating System (OS)
CS232

Concurrency: Common Concurrency Bugs

Dr. Muhammad Mobeen Movania

Outlines

- What are concurrency bugs?
- Classification of concurrency bugs
- Non-deadlock bugs and their types
- Deadlock bugs
- Conditions which cause deadlocks
- Deadlock prevention/avoidance
- Summary

Concurrency bugs and their types

- Errors or exceptions that may arise during execution of concurrent code
- Two types
  - Non-deadlock bugs
  - Deadlock bugs

Types of Concurrency Bugs

- Non-deadlock
  - Atomicity violation
  - Order violation bugs
- Deadlock

Non-deadlock bugs-Atomicity violation

```
1   Thread 1::
2   if (thd->proc_info) {
3     ...
4     fputs(thd->proc_info, ...);
5     ...
6   }
7
8   Thread 2::
9   thd->proc_info = NULL;
```

```
1   pthread_mutex_t proc_info_lock = PTHREAD_MUTEX_INITIALIZER;
2
3   Thread 1::
4   pthread_mutex_lock(&proc_info_lock);
5   if (thd->proc_info) {
6     ...
7     fputs(thd->proc_info, ...);
8     ...
9   }
10  pthread_mutex_unlock(&proc_info_lock);
11
12  Thread 2::
13  pthread_mutex_lock(&proc_info_lock);
14  thd->proc_info = NULL;
15  pthread_mutex_unlock(&proc_info_lock);
```

Non-deadlock bugs-Order violation

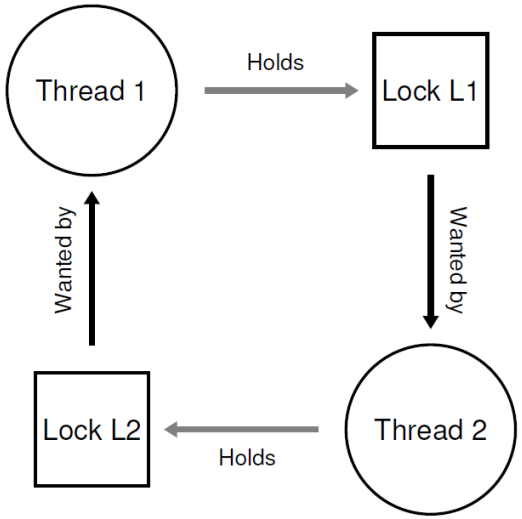
```
1   Thread 1::
2   void init() {
3       ...
4       mThread = PR_CreateThread(mMain, ...);
5       ...
6   }
7
8   Thread 2::
9   void mMain(...) {
10      ...
11      mState = mThread->State;
12      ...
13  }
```

```
1   pthread_mutex_t mtLock = PTHREAD_MUTEX_INITIALIZER;
2   pthread_cond_t  mtCond = PTHREAD_COND_INITIALIZER;
3   int mtInit             = 0;
4
5   Thread 1::
6   void init() {
7       ...
8       mThread = PR_CreateThread(mMain, ...);
9
10      // signal that the thread has been created...
11      pthread_mutex_lock(&mtLock);
12      mtInit = 1;
13      pthread_cond_signal(&mtCond);
14      pthread_mutex_unlock(&mtLock);
15      ...
16  }
17
18  Thread 2::
19  void mMain(...) {
20      ...
21      // wait for the thread to be initialized...
22      pthread_mutex_lock(&mtLock);
23      while (mtInit == 0)
24          pthread_cond_wait(&mtCond, &mtLock);
25      pthread_mutex_unlock(&mtLock);
26
27      mState = mThread->State;
28      ...
29  }
```

Deadlock Bugs

- Easily identified using loops in graphs
- Interdependency of two threads on each other for a shared resource

```
Thread 1:                       Thread 2:
pthread_mutex_lock(L1);         pthread_mutex_lock(L2);
pthread_mutex_lock(L2);         pthread_mutex_lock(L1);
```



Conditions for deadlock

- All four conditions must be met
  - Mutual exclusion
  - Hold-and-wait
  - No preemption
  - Circular wait

Deadlock Prevention

- Circular wait
  - Impose ordering when acquiring locks, L1 to be acquired first before L2
- Hold-and-wait
  - Acquire all resources atomically

```
1    pthread_mutex_lock(prevention);   // begin lock acquisition
2    pthread_mutex_lock(L1);
3    pthread_mutex_lock(L2);
4    ...
5    pthread_mutex_unlock(prevention); // end
```

Deadlock Prevention

- No pre-emption
  - Release acquired resources if unsuccessful

```
1   top:
2     pthread_mutex_lock(L1);
3     if (pthread_mutex_trylock(L2) != 0) {
4       pthread_mutex_unlock(L1);
5       goto top;
6     }
```

- Mutual Exclusion
  - Use atomic operations

```
1   int CompareAndSwap(int *address, int expected, int new) {
2     if (*address == expected) {
3       *address = new;
4       return 1; // success
5     }
6     return 0; // failure
7   }
```

```
1   void AtomicIncrement(int *value, int amount) {
2     do {
3       int old = *value;
4     } while (CompareAndSwap(value, old, old + amount) == 0);
5   }
```

Deadlock Prevention

- Mutual Exclusion
  - Use atomic operations

```
1  void insert(int value) {
2    node_t *n = malloc(sizeof(node_t));
3    assert(n != NULL);
4    n->value = value;
5    n->next  = head;
6    head     = n;
7  }
```

```
1  void insert(int value) {
2    node_t *n = malloc(sizeof(node_t));
3    assert(n != NULL);
4    n->value = value;
5    pthread_mutex_lock(listlock);   // l
6    n->next  = head;
7    head     = n;
8    pthread_mutex_unlock(listlock); // u
9  }
```

```
1  void insert(int value) {
2    node_t *n = malloc(sizeof(node_t));
3    assert(n != NULL);
4    n->value = value;
5    do {
6      n->next = head;
7    } while (CompareAndSwap(&head, n->next, n) == 0);
8  }
```

Deadlock Avoidance

- By scheduling if we have information about threads and available hardware resources

	T1	T2	T3	T4
L1	yes	yes	no	no
L2	yes	yes	yes	no

- Detect and recover
  - Used in databases to prevent data loss

Summary

- We have seen what are the different concurrency bugs that may arise in concurrent code.
- We saw the four necessary conditions for deadlock
- We saw many deadlock avoidance approaches