Synchronization

CS 241

Oct. 9, 2013

```
int ct = 0;
int X = 10000000;

void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++)
    ct++;
}

void main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, up, NULL);
  pthread_create(&t2, NULL, up, NULL);

  printf("Count: %d\n", ct);
}
```

Critical Section

- A **critical section** is a piece of code that:
  -
  -

```
int ct = 0;
int X = 10000000;

void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++)
    atomic { ct++; }
}

void main() {
  pthread_t t1, t2;
  pthread_create(&t1, NULL, up, NULL);
  pthread_create(&t2, NULL, up, NULL);

  printf("Count: %d\n", ct);
}
```

But atomic does not exist in C!

Solution #1: Single Lock Variable

```
int lock = 0;

/* Running by two threads: T1 and T2 */
void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {




    ct++;

  }
}
```

Critical Section

- A correct **critical section** must meet three conditions:
  -
  -
  -

Solution #2: Turns w/ Strict Alternation

```
int turn;

/* Running by two threads: T1 and T2 */
void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {




    ct++;

  }
}
```

Solution #3: Other Flag

```
int owner[2] = { false, false };

/* Running by two threads: T1 and T2 */
void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {




    ct++;

  }
}
```

Solution #4: Two Flag

```
int owner[2] = { false, false };

/* Running by two threads: T1 and T2 */
void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {




    ct++;

  }
}
```

Solution #5: Two Flags and Turns!

```
int owner[2] = { false, false };

/* Running by two threads: T1 and T2 */
void *up(void *ptr) {
  int i;
  for (i = 0; i < X; i++) {




    ct++;

  }
}
```

Peterson's Solution

- The previous solution (#5, two flags and turn) is known as **Peterson's Solution**.
  - Correctly implements a critical section
  - Uses only software
  - Preforms **busy waiting**

- **Solution**: Use hardware operations to implement a better solution.
  - Requires the OS-managed resources

Synchronization Primitives

- Operating systems provide **synchronization primitives** to allow for a single thread to have exclusive access to a region of code.

- Three basic types:
  -
  -
  -