

PTHREAD CONDITION VARIABLE

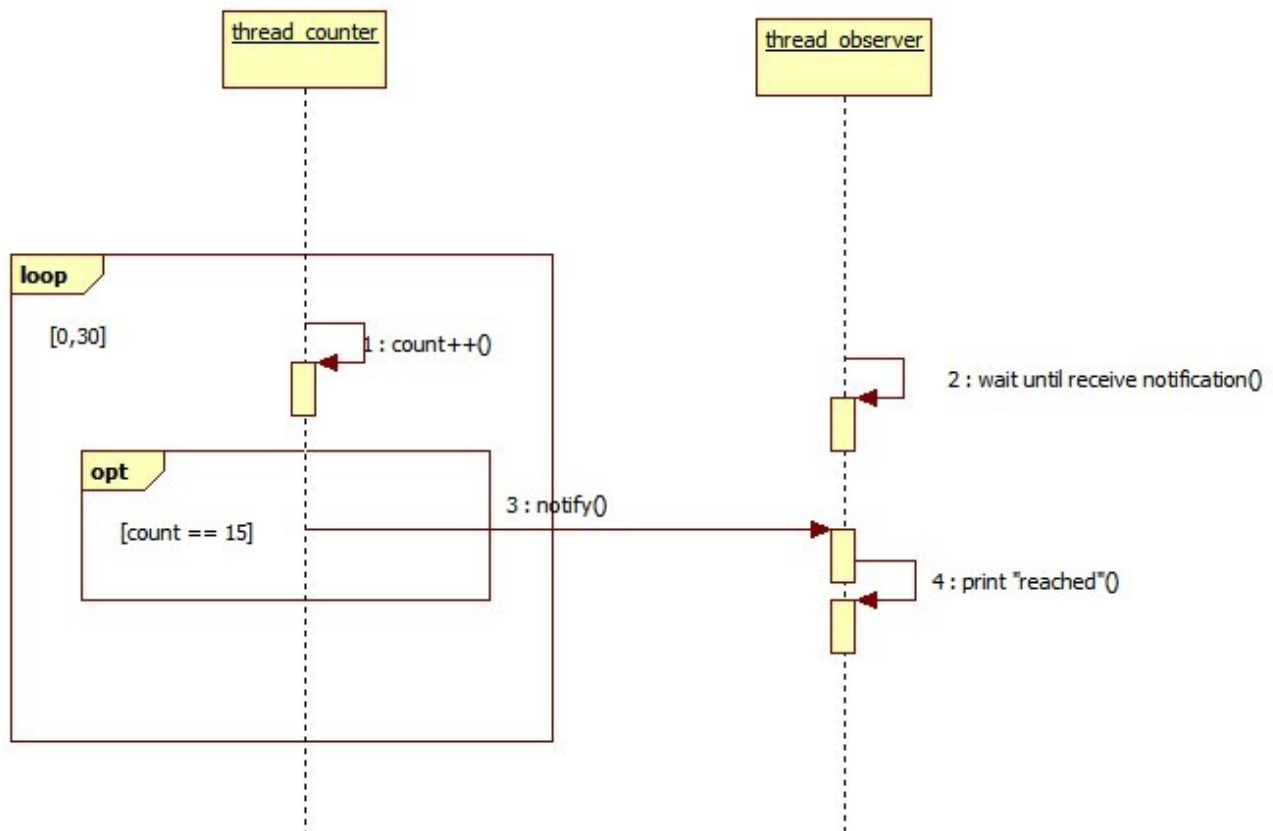
What is condition variable

Condition variable is a synchronization primitive that can be used to:

- Allow a thread to wait for a condition.
- Allow a thread to notify other threads when condition happens.

Lets start with Counter & Observer sample

- thread_counter increase count variable from 0 to 30.
- thread_observer wait until received notification from thread_counter to print "reached".



```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

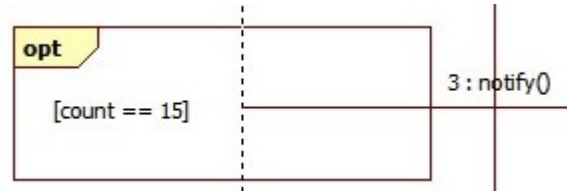
```
int count = 0;
```

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

```
void* inc_count(void* arg) {
    for(int i = 0; i < 30; i++){
        sleep(1);

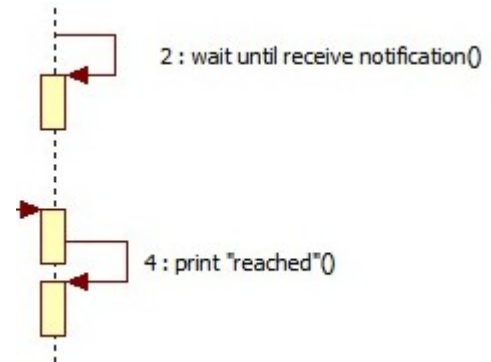
        count++;
        printf("thread counter: %d\n", count);

        if (count == 15) {
            pthread_cond_signal(&cond);
        }
    }
}
```



```
void* watch_count(void* arg) {
    pthread_cond_wait(&cond, &mtx);

    printf("\nthread observer: reached\n\n");
}
```



```
int main(){
    pthread_t thread_counter, thread_observer;

    pthread_create(&thread_counter, NULL, inc_count, NULL);
    pthread_create(&thread_observer, NULL, watch_count, NULL);

    pthread_join(thread_counter, NULL);
    pthread_join(thread_observer, NULL);

    return 0;
}
```

Result

```
thread counter: 11
thread counter: 12
thread counter: 13
thread counter: 14
thread counter: 15

thread observer: reached

thread counter: 16
thread counter: 17
thread counter: 18
thread counter: 19
```

How does above program work?

Variable `cond` work as a signal.

`pthread_cond_t cond`

thread_observer call `pthread_cond_wait()` to suspend and wait until it receive signal on **cond**. While waiting, thread_observer does not consume CPU resources.

`pthread_cond_wait(&cond, &mtx)`

thread_counter call `pthread_cond_signal()` on **cond** to notify **thread_observer**.

`pthread_cond_signal(&cond)`

Initialize and Destroy

How to Initialize?

```
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t attr)
```

`pthread_cond_init` function initialize the condition variable `cond` with attributes defined by `attr`.
If `attr` is `NULL`, condition variable will initialized with default attribute.

`PTHREAD_COND_INITIALIZER` is a helpful macro to initialize static condition variables with default attributes

How to Destroy?

```
int pthread_cond_destroy(pthread_cond_t *cond)
```

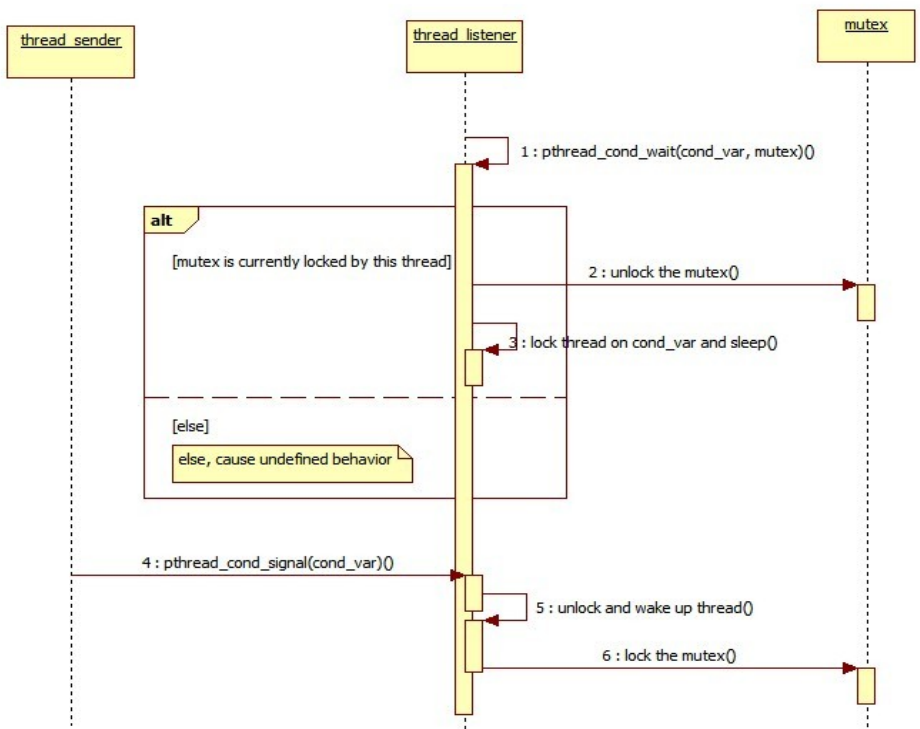
`pthread_cond_destroy` function deinitialize condition variable and release underlying memory.

Destroy the condition variable which is currently blocked by a thread lead to undefined behavior

Wait & Signal

We can use `pthread_cond_wait()` to command thread to wait for a condition variable and `pthread_cond_signal()` to notify that the condition happened.

Each condition variable is used together with a mutex.



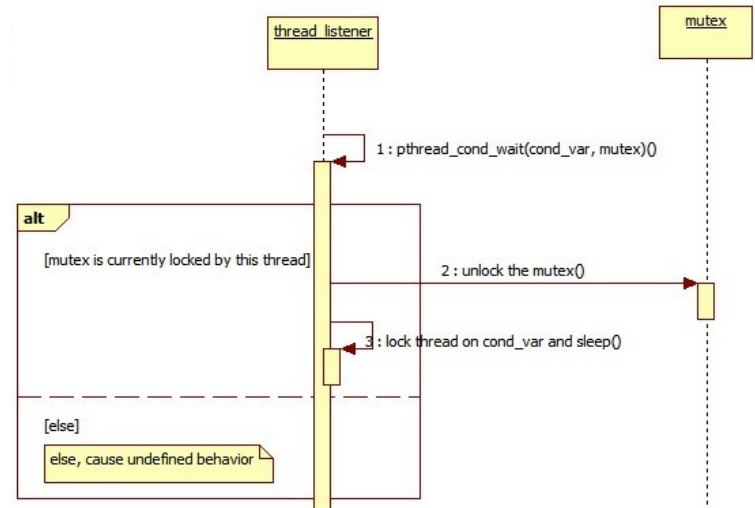
What does `pthread_cond_wait` do?

```
pthread_cond_wait(pthread_cond_t* cond_var, pthread_mutex_t* mutex)
```

If mutex is currently locked by the calling thread, the function performs following steps:

- Release the mutex
- Block the thread on condition variable. Until another thread signals on the condition variable. While blocking, the thread does not consume CPU resources.

If the mutex is not locked by the calling thread, it will cause undefined behavior.



What does `pthread_cond_signal` do?

```
pthread_cond_signal(pthread_cond_t* cond_var)
```

- Unblock at least one of threads that are blocked on the condition variable `cond_var`.
- The unblocked thread will release the mutex that is in conjunction with the condition variable.

What wrong with Counter & Observer sample?

```
void* inc_count(void* arg) {
    for(int i = 0; i < 30; i++){
        sleep(1);

        count++;
        printf("thread counter: %d\n", count);

        if (count == 15) {
            pthread_cond_signal(&cond);
        }
    }
}
```

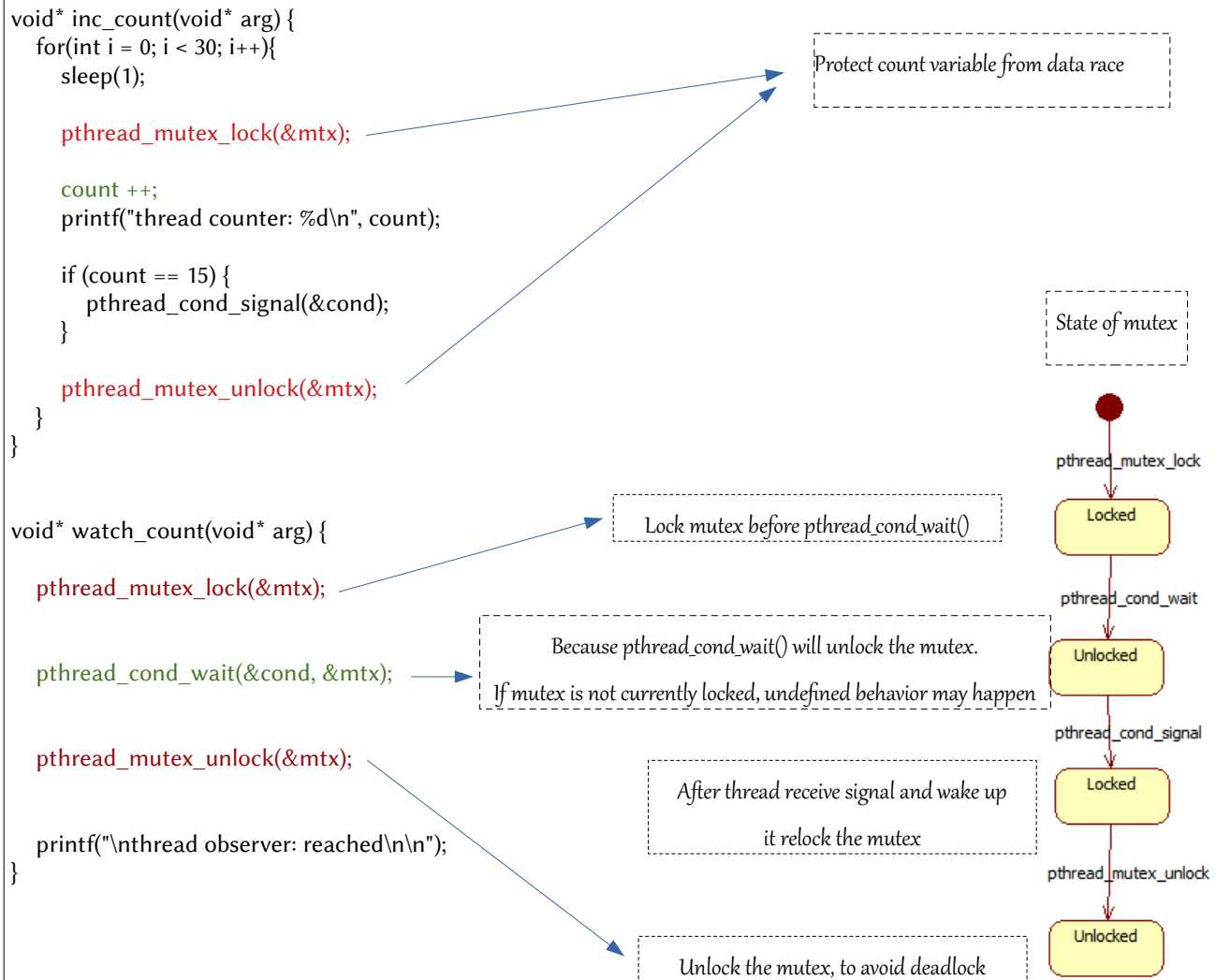
count variable can be read/write by multiple threads at the same time
but it is not protected by any exclusive mechanism
This may cause data race.

```
void* watch_count(void* arg) {
    pthread_cond_wait(&cond, &mtx);

    printf("\nthread observer: reached\n\n");
}
```

`pthread_cond_wait` is called on a unlocked mutex.
This may cause undefined behavior.

Lets fix the issue



All code

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int count = 0;

pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void* inc_count(void* arg) {
    for(int i = 0; i < 30; i++){
        sleep(1);

        pthread_mutex_lock(&mtx);

```

```

    count++;
    printf("thread counter: %d\n", count);

    if (count == 15) {
        pthread_cond_signal(&cond);
    }

    pthread_mutex_unlock(&mtx);
}

void* watch_count(void* arg) {
    pthread_mutex_lock(&mtx);
    pthread_cond_wait(&cond, &mtx);
    pthread_mutex_unlock(&mtx);

    printf("\nthread observer: reached\n\n");
}

int main(){
    pthread_t thread_counter, thread_observer;

    pthread_create(&thread_counter, NULL, inc_count, NULL);
    pthread_create(&thread_observer, NULL, watch_count, NULL);

    pthread_join(thread_counter, NULL);
    pthread_join(thread_observer, NULL);

    return 0;
}

```

//continue

pthread_cond API usages (static/dynamic), UML & description, relation between cond & mutex

improve sample with mutex to avoid data race

add harassing thread, to unlock the mutex

improve sample with while predict