

Chapter 57: Threads (native)

Section 57.1: Initialization by one thread

In most cases all data that is accessed by several threads should be initialized before the threads are created. This ensures that all threads start with a clear state and no *race condition* occurs.

If this is not possible `once_flag` and `call_once` can be used

```
#include <threads.h>
#include <stdlib.h>

// the user data for this example
double const* Big = 0;

// the flag to protect big, must be global and/or static
static once_flag onceBig = ONCE_INIT;

void destroyBig(void) {
    free((void*)Big);
}

void initBig(void) {
    // assign to temporary with no const qualification
    double* b = malloc(largeNum);
    if (!b) {
        perror("allocation failed for Big");
        exit(EXIT_FAILURE);
    }
    // now initialize and store Big
    initializeBigWithSophisticatedValues(largeNum, b);
    Big = b;
    // ensure that the space is freed on exit or quick_exit
    atexit(destroyBig);
    at_quick_exit(destroyBig);
}

// the user thread function that relies on Big
int myThreadFunc(void* a) {
    call_once(&onceBig, initBig);
    // only use Big from here on
    ...
    return 0;
}
```

The `once_flag` is used to coordinate different threads that might want to initialize the same data `Big`. The call to `call_once` guarantees that

- `initBig` is called exactly once
- `call_once` blocks until such a call to `initBig` has been made, either by the same or another thread.

Besides allocation, a typical thing to do in such a once-called function is a dynamic initialization of a thread control data structures such as `mtx_t` or `cnd_t` that can't be initialized statically, using `mtx_init` or `cnd_init`, respectively.

Section 57.2: Start several threads

```
#include <stdio.h>
```

```

#include <threads.h>
#include <stdlib.h>

struct my_thread_data {
    double factor;
};

int my_thread_func(void* a) {
    struct my_thread_data* d = a;
    // do something with d
    printf("we found %g\n", d->factor);
    // return an success or error code
    return d->factor > 1.0;
}

int main(int argc, char* argv[argc+1]) {
    unsigned n = 4;
    if (argc > 1) n = strtoull(argv[1], 0, 0);
    // reserve space for the arguments for the threads
    struct my_thread_data D[n]; // can't be initialized
    for (unsigned i = 0; i < n; ++i) {
        D[i] = (struct my_thread_data){ .factor = 0.5*i, };
    }
    // reserve space for the ID's of the threads
    thrd_t id[4];
    // launch the threads
    for (unsigned i = 0; i < n; ++i) {
        thrd_create(&id[i], my_thread_func, &D[i]);
    }
    // Wait that all threads have finished, but throw away their
    // return values
    for (unsigned i = 0; i < n; ++i) {
        thrd_join(id[i], 0);
    }
    return EXIT_SUCCESS;
}

```