

**Example: Launching Fifteen Threads**

```

07/fifteen-threads.c
3  #include <pthread.h>
4
5  const int num_threads = 15;
6
7  void *thread_start(void *ptr) {
8      int id = *((int *)ptr);
9      printf("Thread %d running...\n", id);
10     return NULL;
11 }
12
13 int main(int argc, char *argv[]) {
14     // Create threads:
15     int i;
16     pthread_t tid[num_threads];
17     for (i = 0; i < num_threads; i++) {
18         pthread_create(&tid[i], NULL,
19                        thread_start, (void *)&i);
19     }
20
21     printf("Done!\n");
22     return 0;
23 }

```

**Creating Additional Threads in C**

The pthread library is the POSIX thread library allowing you to create additional threads beyond the initial **main** thread.

Creating a new thread is a complex call with four arguments:

```

int pthread_create(
    pthread_t *thread,          /* thread struct */
    const pthread_attr_t *attr, /* usually NULL */
    void *(*start_routine) (void *), /* start func */
    void *arg                   /* thread start arg */
);

```

The **start\_routine** of **pthread\_create** has a very interesting type signature:

```
void *(*start_routine) (void *)
```

This signature is a **function pointer** (“functor”) and is the syntax we can use to pass a pointer to a function. Therefore, the third argument into **pthread\_create** must be a function with the following prototype:

```
void *_____ (void *ptr);
```

...you can use any name for the function name.

**Q1:** What is the expected output of the **fifteen-threads.c** program?

**Q2:** What actually happens?

**Q3:** What do we know about threads in C?

**Five-State Thread Model**

When the operating system has control over the CPU and needs to decide what program to run, it must maintain a model of all threads within the CPU.

We commonly refer to the “state” of a thread as part of the five-state model:

## 08/fifteen-join.c

```

13 int main(int argc, char *argv[]) {
14     // Create threads:
15     int i;
16     pthread_t tid[num_threads];
17     for (i = 0; i < num_threads; i++) {
18         int *val = malloc(sizeof(int));
19         *val = i;
20         pthread_create(&tid[i], NULL,
21                       thread_start, (void *)val);
22     }
23     // Joining Threads
24     for (i = 0; i < num_threads; i++) {
25         pthread_join(tid[i], NULL);
26     }
27
28     printf("Done!\n");
29     return 0;
30 }

```

**pthread\_join** – In the above program, we use **pthread\_join**. This call will \_\_\_\_\_ from running the program further until the specified thread has **finished and returned**.

**Q1:** What happens in this program?

**Q2:** Does the order vary each time we run it? What is happening?

**Q3:** What can we say about the relationship between “Done” and “Thread %d running...” lines?

## Counting with Threads

Here’s a new program using multiple threads, which we will compile as the executable **count** (`gcc count.c -lpthread -o count`):

## 08/count.c

```

5 int ct = 0;
6
7 void *thread_start(void *ptr) {
8     int countTo = *((int *)ptr);
9
10    int i;
11    for (i = 0; i < countTo; i++) {
12        ct = ct + 1;
13    }
14
15    return NULL;
16 }
17
18 int main(int argc, char *argv[]) {
19     /* [...check argv size...] */
20
21
22
23
24
25     const int countTo = atoi(argv[1]);
26     /* [...error checking...] */
27
28     const int thread_ct = atoi(argv[2]);
29     /* [...error checking...] */
30
31     // Create threads:
32     int i;
33     pthread_t tid[thread_ct];
34     for (i = 0; i < thread_ct; i++) {
35         pthread_create(&tid[i], NULL,
36                       thread_start, (void *)&countTo);
37     }
38
39     // Join threads:
40     for (i = 0; i < thread_ct; i++) {
41         pthread_join(tid[i], NULL);
42     }
43
44     // Display result:
45     printf("Final Result: %d\n", ct);
46     return 0;
47 }

```

**Q1:** What do we expect when we run this program?

**Q2:** What is the output of running:  
./count 100 2

**Q3:** What is the output of running:  
./count 100 16

**Q4:** What is the output of running:  
./count 10000000 2

**Q5:** What is the output of running:  
./count 10000000 16

**Q6:** What is going on???