Example: Launching Fifteen Threads

```c
#include <pthread.h>

const int num_threads = 15;

void *thread_start(void *ptr) {
    int id = *((int *)ptr);
    printf("Thread %d running...\n", id);
    return NULL;
}

int main(int argc, char *argv[]) {
    // Create threads:
    int i;
    pthread_t tid[num_threads];
    for (i = 0; i < num_threads; i++) {
        pthread_create(&tid[i], NULL,
                        thread_start, (void *)&i);
    }
    printf("Done!\n");
    return 0;
}
```

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:

```c
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:
```c
int main(int argc, char *argv[]) {
  // Create threads:
  int i;
  pthread_t tid[num_threads];
  for (i = 0; i < num_threads; i++) {
    int *val = malloc(sizeof(int));
    *val = i;
    pthread_create(&tid[i], NULL,
                   thread_start, (void *)val);
  }

  // Joining Threads
  for (i = 0; i < num_threads; i++) {
    pthread_join(tid[i], NULL);
  }

  printf("Done!\n");
  return 0;
}
```

### 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):

```c
int ct = 0;
void *thread_start(void *ptr) {
  int countTo = *((int *)ptr);
  int i;
  for (i = 0; i < countTo; i++) {
    ct = ct + 1;
  }
  return NULL;
}
```

```c
int main(int argc, char *argv[]) {
  /* [...]check argv size... */
  const int countTo = atoi(argv[1]);
  /* [...]error checking... */
  const int thread_ct = atoi(argv[2]);
  /* [...]error checking... */

  // Create threads:
  int i;
  pthread_t tid[thread_ct];
  for (i = 0; i < thread_ct; i++) {
    pthread_create(&tid[i], NULL,
                   thread_start, (void *)&countTo);
  }

  // Join threads:
  for (i = 0; i < thread_ct; i++) {
    pthread_join(tid[i], NULL);
  }

  // Display result:
  printf("Final Result: %d\n", ct);
  return 0;
}
```

**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 1000000 2
```

**Q5:** What is the output of running:
```
./count 1000000 16
```

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