CS 340 #9: Thread Creation, Join, & Five State Model

Computer Systems Sep. 19, 2023 · G Carl Evans

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);
      printf("Thread %d running...\n", id);
 9
10
      return NULL;
11
    }
12
   int main(int argc, char *argv[]) {
13
      // Create threads:
14
15
      int i:
16
      pthread_t tid[num_threads];
      for (i = 0; i < num_threads; i++) {</pre>
17
        pthread_create(&tid[i], NULL,
18
                                    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:

 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[]) {
     // Create threads:
14
15
     int i:
     pthread_t tid[num_threads];
16
     for (i = 0; i < num_threads; i++) {</pre>
17
       int *val = malloc(sizeof(int));
18
       *val = i;
19
       pthread_create(&tid[i], NULL,
20
                                  thread_start, (void *)val);
21
     }
22
     // Joining Threads
23
24
     for (i = 0; i < num_threads; i++) {</pre>
       pthread_join(tid[i], NULL);
25
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	<pre>int ct = 0;</pre>	Q1: What do we
6		expect when we run
7	void *thread_start(void *ptr) {	this program?
8	int countro = *((int *)ptr);	
10	int i	
11	for $(i = 0; i < countTo; i++)$	
12	ct = ct + 1:	Q2: What is the
13	}	output of running:
14		./count 100 2
15	return NULL;	
16	}	
17		Oo. What is the
18	<pre>int main(int argc, char *argv[]) {</pre>	Q3: What is the
••••	/* [check argv size] */	output of running:
24		./count 100 10
25	<pre>const int countlo = atol(argv[1]); /* [</pre>	
28	const int thread $ct = atoi(argy[2])$:	
20	/* [error_checking] */	0 4. What is the
30	, [output of running.
31	// Create threads:	/count 10000000 2
32	int i;	
33	pthread_t tid[thread_ct];	
34	<pre>for (i = 0; i < thread_ct; i++) {</pre>	
35	<pre>pthread_create(&tid[i], NULL,</pre>	05: What is the
	thread_start, (void *)&countTo);	output of running:
36	}	./count 10000000 16
3/	// loin threads:	
30	for $(i = 0; i < thread ct; i++)$	
40	pthread join(tid[i], NULL):	
41	}	Q6: What is going
42		on???
43	// Display result:	
44	<pre>printf("Final Result: %d\n", ct);</pre>	
45	return 0;	
46	}	