Why does WaitForMultipleObjects return ERROR_INVALID_PARAMETER when all the parameters look valid to me?

devblogs.microsoft.com/oldnewthing/20110225-00

February 25, 2011

Raymond Chen

A customer asked for assistance with the `WaitForMultipleObjects` function:

> I am getting `ERROR_INVALID_PARAMETER` when calling `WaitForMultipleObjects` even though all the parameters are valid as far as I can tell. I've narrowed it down to this simple program.

```
int main()
{
 int i;
 HANDLE Handles[4];
 // Create the events
 for (i = 0; i < 4; i++) {
  Handles[i] = CreateEvent(NULL, FALSE, FALSE, TEXT("Test"));
  if (Handles[i] == NULL) {
   printf("Failed to create event - test failed\n"); return 0;
  }
 }
 // Set them all to signaled
 for (i = 0; i < 4; i++) SetEvent(Handles[i]);
 // Wait for all of them - we expect this to return WAIT_OBJECT_0
 printf("WaitForMultipleObjects returned %d\n",
        WaitForMultipleObjects(4, Handles, TRUE, INFINITE));
 return 0;
}
```

First of all, *thank you* for narrowing the issue down to a minimal program that illustrates the problem. You'd be surprised how often a customer says, "I'm having problem with function X. Here's a program that illustrates the problem." And then attaches a huge project that doesn't compile because it is written in some development environment different from the one you have on your machine.

The problem here is that you are passing four handles to the same event to `WaitForMultipleObjects` with the `bWaitAll` parameter set to `TRUE`. The `WaitForMultipleObjects` function rejects duplicates if you ask it to wait for all of the objects. Why is that?

Well, consider this program: It creates a named auto-reset event (as is "obvious" from the `FALSE` second parameter passed to `CreateEvent`) and stores a handle to it in `Handles[0]`. The second through fourth calls to `CreateEvent` merely create new handles to the same auto-reset event because the name matches an existing event. The second loop sets that same event four times. And then the `WaitForMultipleObjects` asks to wait for all of the handles to be signaled. But since all four handles refer to the same object, it's being asked to wait until the event has reached the state where the wait can complete four times simultaneously. (Huh?)

Recall that `WaitForMultipleObjects` does not alter the state of any of the waited objects until the wait completes. If you ask it to wait for both an event and a semaphore, and the event is signaled but the semaphore is not, then the function will leave the event signaled while it waits for the semaphore. Only when all the items being waited for are signaled will the `WaitForMultipleObjects` function perform whatever actions are appropriate for acquiring a signaled object and return.

Okay, so we asked it to wait on the same auto-reset event four times. But that's nonsense: An auto-reset event is just a stupid semaphore which can have at most one token. But in order for the wait to succeed, it needs four tokens. That's never going to happen, so the wait is nonsensical.

More generally speaking, `WaitForMultipleObjects` returns `ERROR_INVALID_PARAMETER` if you pass `bWaitAll = TRUE` and there are any duplicates in the handle array (either identical handles, or different handles to the same underlying object). It doesn't try to puzzle out the objects and say, "Well, let me see if this is a reasonable combination of objects to wait on more than once"; it just sees the duplicate and says "Forget this!"

Going back to the customer's original problem: We asked why they were creating four handles to the same object, and what they expected when waiting for an auto-reset event to have four available tokens (which it never will), and the customer admitted that it was just an error in their code. The original version of the code used a named event and waited on it with `WaitForSingleObject`, and when they modified the code to make it support multiple events, they forgot to give each event a different name.

Raymond Chen

Follow