The difference between thread-safety and re-entrancy

devblogs.microsoft.com/oldnewthing/20040629-00

June 29, 2004

Raymond Chen

An operation is “thread-safe” if it can be performed from multiple threads safely, even if the calls happen simultaneously on multiple threads.

An operation is re-entrant if it can be performed while the operation is already in progress (perhaps in another context). This is a **stronger** concept than thread-safety, because the second attempt to perform the operation can even come **from within the same thread**.

Consider the following function:

```
int length = 0;
char *s = NULL;
// Note: Since strings end with a 0, if we want to
// add a 0, we encode it as "\0", and encode a
// backslash as "\\".
// WARNING! This code is buggy - do not use!
void AddToString(int ch)
{
  EnterCriticalSection(&someCriticalSection);
  // +1 for the character we're about to add
  // +1 for the null terminator
  char *newString = realloc(s, (length+1) * sizeof(char));
  if (newString) {
    if (ch == '\0' || ch == '\\') {
      AddToString('\\'); // escape prefix
    }
    newString[length++] = ch;
    newString[length] = '\0';
    s = newString;
  }
  LeaveCriticalSection(&someCriticalSection);
}
```

This function is thread-safe because the critical section prevents two threads from attempting to add to the string simultaneously. However, it is not re-entrant.

The internal call to AddToString occurs while the data structures are unstable. At the point of the call, execution re-enters the start of the function AddToString, but this time the attempt to realloc the memory will use a pointer (s) that is no longer valid. (It was invalidated by the call to realloc performed by the caller.)

Raymond Chen

Follow