MPG123 SSE decoder bug: not thread safe.

We are implementing an audio mixer application for multi-channel, real-time, streaming audio. To do this, we allocate each incoming (encoded) mp3 channel to a thread to decode the channel and subsequent threads are used to combine these (now) decoded streams to produce sound. Each decoding thread uses MPG123 (specifically mpg123_read() based on mpg123_to_wav.c) to perform this decode function (other variants of these threads have used mpg123_decode() based on mpglib.c).

In the course of development, we discovered that the SSE decoder is not thread safe. This problem still exists in MPG123 1.5.0.

To demonstrate the bug, we have constructed a 'cutdown' which is a modification of the code for mpg123_to_wav.c, now called mpg123_sample(), which is now the thread body. Those modifications are:

1) we made it a procedure with parameters of "experimental conditon", input file, output file, decoder, and a mutex variable.
2) we used the parameters as appropriate in mpg123_init() and mpg123_open().
3) we wrote the streaming WAV to an output file rather than stdout
4) we removed use of sndfile.c and replaced calls with fopen() and fwrite().
5) we added code to add a WAV Header to create a .wav file as output.

We wrapped the modifications with a new main procedure that retrieves these parameters, invokes mpg123_sample() with two threads, invokes the system call md5sum on the results from each thread and compares the md5sum of the two resulting files (to quickly determine if the decoded results are identical). Each test either passes (P: md5sums are the same) or fails (F: md5sums differ).

The experimental conditions that we used were:

"cond1". In this case, the two threads were synchronized so that mpg123_read() was invoked sequentially--that is thread1 is allow to complete first, then thread2 is then allowed to proceed.

"cond2" and "cond3". Each thread is permitted to proceed and compete for CPU meaning that calls to mpg123_read() will (likely*) interleave as each thread progresses. (Note: the documentation is ambiguous as to whether mpg123_init() should be invoked once for each thread or once per process--hence here "cond2" invokes mpg123_init() once per thread and whereas "cond3" invokes mpg123_init() once per process).

We ran our cutdown for a variety of different experimental conditions. Namely, "cond1, cond2, and cond3" with each against the following compile time and operating system environments. First was whether or not the mpg123 library was built statically or shared (i.e., ./configure --enable-shared=yes --enable-static=no *OR* ./configure --enable-shared=no --enable-static=yes). Secondly, was for either Win32/cygwin or Linux/ Ubuntu8.04LTS OS environments. Lastly we specifically indicated which optimized decoders was to be used, for us, './configure' deduced: SSE, MMX, i586, i386, generic (in that order).

The experimental results are pretty clear, that all of the mult-threaded decoding tests passed (P) under all conditions except for when the SSE decoder was specified. The only case where SSE passed was under "cond1" which essentially serializes the decoding so that there is no interference/interleaving between the two threads.

The following table gives the results:

Operating environment:	Win32/cygwin (gcc v3.4.4)						Linux/Ubuntu8.04LTS (gcc v4.2.3)					
Library build:	static			shared			static			shared		
Condition:	1	2	3	1	2	3	1	2	3	1	2	3
decoder: SSE	P	F	F	P	F	F	P	F	F	P	F	F
decoder: MMX	P	P	P	P	P	P	P	P	P	P	P	P
decoder: i586	P	P	P	P	P	P	P	P	P	P	P	P
decoder: i386	P	P	P	P	P	P	P	P	P	P	P	P
decoder: generic	P	P	P	P	P	P	P	P	P	P	P	P

The modified code, makefiles, Windows cmd scripts and bash shell scripts are attached to help reproduce our results.

*by likely we mean that it is possible (we have observed a 1 in 50 chance) that using 'cond2' or 'cond3' the two files will be the same due to the way the interweaving actually occurs.