Cancellation and C++ Exceptions

In NPTL thread cancellation is implemented using exceptions. This does not in general conflict with the mixed use of cancellation and exceptions in C++ programs. This works just fine. Some people, though, write code which doesn't behave as they expect. This is a short example:

```
#include <cstdlib>
#include <iostream>
#include <pthread.h>

static pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t c = PTHREAD_COND_INITIALIZER;

static void *tf (void *)
{
  try {
    ::pthread_mutex_lock(&m);
    ::pthread_cond_wait (&c, &m);
  } catch (...) {
    // do something
  }
}

int main ()
{
  pthread_t th;
  ::pthread_create (&th, NULL, tf, NULL);
  // do some work; simulate using sleep
  std::cout << "Wait a bit" << std::endl;
  sleep (1);
```

```
  // cancel the child thread
  ::pthread_cancel (th);
  // wait for it
  ::pthread_join (th, NULL);
}
```

The problem is in function `tf`. This function contains a catch-all clause which does not rethrow the exception. This is possible to expect but should really never happen in any code. The rules C++ experts developed state that catch-all cases must rethrow. If not then strange things can happen since one doesn't always know exactly what exceptions are thrown. The code above is just one example. Running it will produce a segfault:

```
$ ./test
Wait a bit
FATAL: exception not rethrown
Aborted (core dumped)
```

The exception used for cancellation is special, it cannot be ignored. This is why the program aborts.

Simply adding the rethrow will cure the problem:

```
@@ -13,6 +13,7 @@
     ::pthread_cond_wait (&c, &m);
   } catch (...) {
     // do something
+    throw;
   }
 }
```

But this code might not have the expected semantics. Therefore the more general solution is to change the code as such:

```
@@ -1,6 +1,7 @@
 #include <cstdlib>
 #include <iostream>
 #include <pthread.h>
+#include <cxxabi.h>

 static pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
 static pthread_cond_t c = PTHREAD_COND_INITIALIZER;
@@ -11,6 +12,8 @@
   try {
     ::pthread_mutex_lock(&m);
     ::pthread_cond_wait (&c, &m);
+  } catch (abi::__forced_unwind&) {
+    throw;
   } catch (...) {
     // do something
   }
```

The header `cxxabi.h` comes with gcc since, I think, gcc 4.3. It defines a special tag which corresponds to the exception used in cancellation. This exception is not catchable, as already said, which is why it is called `__forced::unwind`.

That's all. That is needed. This code can easily be added to existing code, maybe even with a single hidden use:

```
#define CATCHALL catch (abi::__forced_unwind&) { throw; } catch (...)
```

This macro can be defined predicated on the gcc version and the platform.

I still think it is better to always rethrow the execption, though.