Subject: : AmigaOS4
Topic: : MineCraft (MineTest) work in progress help need it
Re: MineCraft (MineTest) work in progress help need it
Author: : kas1e
Date: : 2021/1/13 21:29:09
URL:

@All

So that what happen when we just tried to pass KillThread() test:

I put printfs all over the ways in test-function inself:

```
 void TestThreading::testThreadKill()
{
   SimpleTestThread *thread = new SimpleTestThread(300);

   printf("1n");
   UASSERT(thread->start() == true);

   // kill()ing is quite violent, so let's make sure our victim is sleeping
   // before we do this... so we don't corrupt the rest of the program's state
   printf("2n");
   sleep_ms(100);
   printf("3n");
   UASSERT(thread->kill() == true);

   printf("4n");
   // The state of the thread object should be reset if all went well
   UASSERT(thread->isRunning() == false);
   printf("5n");
   UASSERT(thread->start() == true);
   printf("6n");
   UASSERT(thread->stop() == true);
   printf("7n");
   UASSERT(thread->wait() == true);

   printf("8n");
   // kill() after already waiting should fail.
   UASSERT(thread->kill() == false);

   delete thread;
}
```

What we see in the console output that it's: 1,2,3 and never 4 or more.

On the serial with adding prints to our threading implementation we have that:

Quote:

> Before ObtainSemaphore in __gthread_create
> After ObtainSemaphore in __gthread_create
> after findtask + strcutprocess_userData , but before Wait for the parent task
> after while with (SIGBREAKF_CTRL_F) & SIGBREAKF_CTRL_F)
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
> Waiting for thread
>
> ... busy forever ...

All is see is that first UASSERT(thread->kill() == true); didn't works.

And their kill() are there:

https://github.com/minetest/minetest/b ... threading/thread.cpp#L139

I put prinfs all over it as well, like this:

```
bool Thread::kill()
{
    if (!m_running) {
        printf("from kill() 1n");
        wait();
        return false;
    }

    printf("from kill() 2n");
    m_running = false;

    printf("from kill() 3n");
```

```
#if defined(_WIN32)
    // See https://msdn.microsoft.com/en-us/library/hh920601.aspx#thread__native_handle_method
    TerminateThread((HANDLE) m_thread_obj->native_handle(), 0);
    CloseHandle((HANDLE) m_thread_obj->native_handle());
#else
    // We need to pthread_kill instead on Android since NDKv5's pthread
    // implementation is incomplete.
# ifdef __ANDROID__
    pthread_kill(getThreadHandle(), SIGKILL);
# else
    printf("from kill() 4n");
    pthread_cancel(getThreadHandle());
# endif
    printf("from kill() 5n");
    wait();
#endif

    printf("from kill() 6n");
    m_retval       = nullptr;
    m_joinable     = false;
    m_request_stop = false;

    return true;
}
```

And we can see 2,3,4,5, but not 6. Meaning that the last wait() executed and we didn't go futher.

So i added debug prinfs to their wait() as well, like that:

```
 bool Thread::wait()
{
    printf("wait 1n");
    MutexAutoLock lock(m_mutex);

    printf("wait 2n");
    if (!m_joinable)
        return false;


    printf("wait 3n");
    m_thread_obj->join();

    printf("wait 4n");
    delete m_thread_obj;
    m_thread_obj = nullptr;

    printf("wait 5    n");
    assert(m_running == false);
    m_joinable = false;
```

```
    printf("wait 6n");
    return true;
}
```

And as a result, we have there: 1,2,3, and never 4. So, join() fail, again our majesty join() :)

**In summary** what we have with that test: test start a thread, then wait a bit, and send kill. Kill is just called pthread_cancel and then call internal wait() which check: if not joinable, then return false, but if joinable then tried to join. And we fail to join there with our forever busy "waiting for a thread" from native implementation.