A thread that joins itself

considered harmful

Disclaimer

- Warning: PowerPoint is not a C++ compiler [-Wpowerpoint]
- Warning: Toy example detected [-Wtoyexample]

https://en.cppreference.com/w/cpp/thread/thread/join

Exceptions

- std::system_error if an error occurs.

Error Conditions

- no_such_process if the thread is not valid
- invalid_argument if joinable() is false
- std::errc::resource_deadlock_would_occur if this->get_id() == std::this_thread::get_id() (deadlock detected)

It looks like a corner case, doesn’t it?

```
std::thread t;
t = std::thread( [](std::thread* pt) { pt->join(); }, &t );
t.join();
```

```
Start
terminate called after throwing an instance of 'std::system_error’
what(): Resource deadlock avoided
Aborted
Finish
```

My software is not that convoluted

```
class job;

class queue {
    std::vector<job> v;
    std::mutex mut;
    std::atomic<bool> stop{false};
    std::thread worker;

    void do_the_work();
public:
     queue();

     void add_another(job&& w);
     void you_can_stop_now();
};
```

```
queue q;

int main()
{
   while (there_is_something_to_do())
   {
      q.add_another(get_job());
   }
   q.you_can_stop_now();
}
```

```
class job;

class queue {
    std::vector<job> v;
    std::mutex mut;
    std::atomic<bool> stop{false};
    std::thread worker;

    void do_the_work();
public:
     queue();

     void add_another(job&& w);
     void you_can_stop_now();
};
```

```
queue::queue()
{
  worker =
   std::thread([this] { do_the_work(); });
}

void queue::do_the_work()
{
   while (!stop)
   {
      std::vector<job> w;
      std::unique_lock l(mut);
      w.swap(v);
      l.unlock();

      for (auto&& j : w)
         j.do_it();
   }
}
```

```
class job;

class queue {
    std::vector<job> v;
    std::mutex mut;
    std::atomic<bool> stop{false};
    std::thread worker;

    void do_the_work();
public:
     queue();

     void add_another(job&& w);
     void you_can_stop_now();
};
```

```
void queue::add_another(job&& w)
{
   if (stop) return;
   std::unique_lock lock(mut);
   v.push_back(std::move(w));
}

void queue::you_can_stop_now()
{
   stop = true;
   worker.join();
}
```

Good. Let’s add a new feature

```
class job;

class queue {
    std::vector<job> v;
    std::mutex mut;
    std::atomic<bool> stop{false};
    std::thread worker;

    void do_the_work();
public:
     queue();

     void add_another(job&& w);
     void you_can_stop_now();
};
```

```
queue q;

int main()
{
   while (there_is_something_to_do())
   {
      q.add_another(get_job());
   }
   q.you_can_stop_now();
}
```

Good. Let’s add a new feature

- Let’s exit the program when CTRL-C is pressed
- That’s a POSIX signal:
  - A signal is an external event, with an associated integer code
  - Each signal has a disposition (i.e. how the process behaves when it is delivered the signal):
    - perform the default action
    - ignore the signal
    - catch the signal with a signal handler, a programmer-defined function that is automatically invoked
- So we just need to register a signal handler that stops the queue

Here it is…

```
#include <signal.h>

// we get:

// typedef void (*sig_t)(int); // signal-handler
// sig_t signal(int sig, sig_t func);

// SIGINT is the code corresponding to CTRL-C

queue q;

int main()
{
   …
```

Here it is…

```
#include <signal.h>

queue q;

void exit_gracefully(int)
{
   q.you_can_stop_now();
}

int main()
{
   signal(SIGINT, exit_gracefully);

   while (there_is_something_to_do())
   {
      q.add_another(get_job());
   }
}
```

…but does it really work?

- Each individual thread can decide which signals to block
    - By default they are all unblocked
- A signal is delivered to a single arbitrarily selected thread that does not currently have the signal blocked
- By default, a signal handler is invoked on the normal process stack

…but does it really work?

- In other words, any thread can be resumed anytime inside the signal handler:

```
queue q;

void exit_gracefully(int)
{
   q.you_can_stop_now();
}
```

May or may not work

- …So we have a 50% chance that ‘worker’ thread executes:
  - q.you_can_stop()
    - worker.join()
      - Uh-Oh, that’s an exception!

```
queue q;

void exit_gracefully(int)
{
   q.you_can_stop_now();
}
```

Thanks. I think I got the idea.

- Thank you for listening
- Questions?