Лекция 12. Стратегии обработки ошибок. Отладка

Разумная стратегия обработки ошибок

- Идентификация: что является ошибкой
- Обнаружение: какой код обнаруживает ошибку
- Распространение: как информация об ошибке распространяется в каждом модуле (и в т.ч. кроссмодульное распространение)
- Обработка: где выполняются действия, связанные с ошибкой (возможно, не единственное место)
- Уведомление: как информируется пользователь

А ошибка ли?

- Ошибка – сбой, который не дает функции успешно завершиться:
  - Нарушение предусловия. Пример: нарушения ограничения на параметр или состояние класса.
  - Нет возможности выполнить постусловие. Пример: функция должна вернуть значение, а получить его не удалось.
  - Нет возможности восстановить инвариант, за поддержку которого она отвечает. Пример: любая не private функция должна восстанавливать инвариант к своему окончанию.

Как обработать ошибку в C++

- Выставить глобальную переменную и вернуть какое-нибудь невалидное значение
- Вернуть код ошибки
- Остановить программу
- Сгенирировать исключение

Информирование кодами ошибки

- Часто забывают проверить
- Глобальные переменные:
  - вернуть невалидное значение не всегда возможно,
  - возникают сложности в многопоточном приложении.
- Возвращение кода ошибки
  - приводит к «разбуханию» кода

```
1.
2.  double lg_value = 0;
3.  double tmp = 0;
4.  if (lexical_cast<double>(str, &tmp))
5.      return log(tmp, &lg_value);
6.  return false;
7.  // vs
8.  return lg_value = log(lexical_cast<double>(str));
```

Исключения

+ Невозможно проигнорировать.
+ Позволяют писать компактный код.
+ Автоматически отвечают за распространение.
+ Генерируются там, где ошибка найдена; обрабатываются там, где понятно как.
+ Неперехваченное искючение заканчивает программу.

assert

- Работают только в режиме отладки (NDEBUG).
- Выводят сообщение об ошибке, с указанием какое условие и в какой точке кода не выполнилось.
- Под отладчиком позволяют встать в точку ошибки.
- Важно! Не пишите обязательный к выполнению код внутри assert (см. п.1)

```
1.  template<class T>
2.  T& vector<T>::operator[](size_t index)
3.  {
4.      assert(index < size_);
5.      return data_[index];
6.  }
```

Журналирование

- Наличие журнала (лога) работы программы очень помогает понять, что с ней происходит и отладить в случае падения. Часто используются свои простые решения или внешние библиотеки, например log4cxx, boost::log.

```
1.  LogInfo("TCP connection established: " << endp);
2.  LogInfo("Received udp port " << msg.port);
```

```
arm_atc 02.12.2013 INFO state_dumper (state_thread.cpp   : 100) Started process_state thread
arm_atc 02.12.2013 INFO wms          (png_client.cpp     : 50 ) Png client connected to 10.10.78.94:45117
arm_atc 02.12.2013 WARN time_sync    (time_synchronizer.h: 50 ) Too long round trip for sync message...
```

Crash reporting

- При возникновении ошибки возможно получить не только описание ошибки, но и полное состояние программы во время падения. Помогут в этом, например google break-pad или microsoft minidumps.

```
Access violation
    Crashed thread: 11000
    Physical memory usage: 5719 Mb / 8183 Mb
    Used by this process : 71 Mb

    ==============
    Threads Stacks
    ==============
    RVAFunctionFile:Line

    Thread id: 11000
Stack:
    000ce966  aircraft::model::get_phys_pos()        ...\aircraft\aircraft_model.cpp:165
    003adb17  vehicle::model::update()               ...\vehicle\vehicle_model.cpp:76
    0009f32b  kernel::system_base::do_update()       ...\common\system_base.cpp:476
    0006b566  kernel::model_system_impl::do_update() ...\model_system\model_system_impl.cpp:35
    000ae546  kernel::system_base::update()          ...\common\system_base.cpp:124
...
```

Отладка

- `printf`-style
- Отладчик:
  - Запуск и присоединение к процессу
  - Run, Pause, Stop
  - Call stack
  - Breakpoints
  - Watch, changing variables
  - Step into, over, out
  - Aux windows: Autos, Locals
  - Threads, Modules, Processes
  - Moving instruction pointer
  - Memory
  - Disassembly

Вопросы?