boostcon
BOOST LIBRARIES CONFERENCE 2011

Boost.Process: Process management in C++

Boris Schäling, May 2011, www.highscore.de

- How many attempts have there been to finish the library?
- What has been tried and didn't work?
- What's the current status?
- What are the plans for the future?

Boost.Process: A simple library?

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
#include <boost/process/all.hpp>
#include <string>

using namespace boost::process;

int main()
{
  std::string exe = find_executable_in_path("hostname");
  create_child(exe);
}
```

100 %

Error List Output Find Symbol Results

Item(s) Saved Ln 1 Col 1 Ch 1 INS

Solution Explorer

Boost.Process: A simple library?

I want cross-platform code and no #ifdefs.

I want to use system specific features.

I want an interface as simple as possible.

I want to be flexible and extend the library.

Boost.Process: A simple library?

I want cross-platform code and no #ifdefs.

Only few concepts exist on all supported platforms (Windows and POSIX) which makes the library small and limited in usefulness

Boost.Process: A simple library?

I want to use system specific features.

Boost C++ libraries try to be platform-independent – it‘s not possible to model all system specific features in a cross-platform manner

Boost.Process: A simple library?

I want an interface as simple as possible.

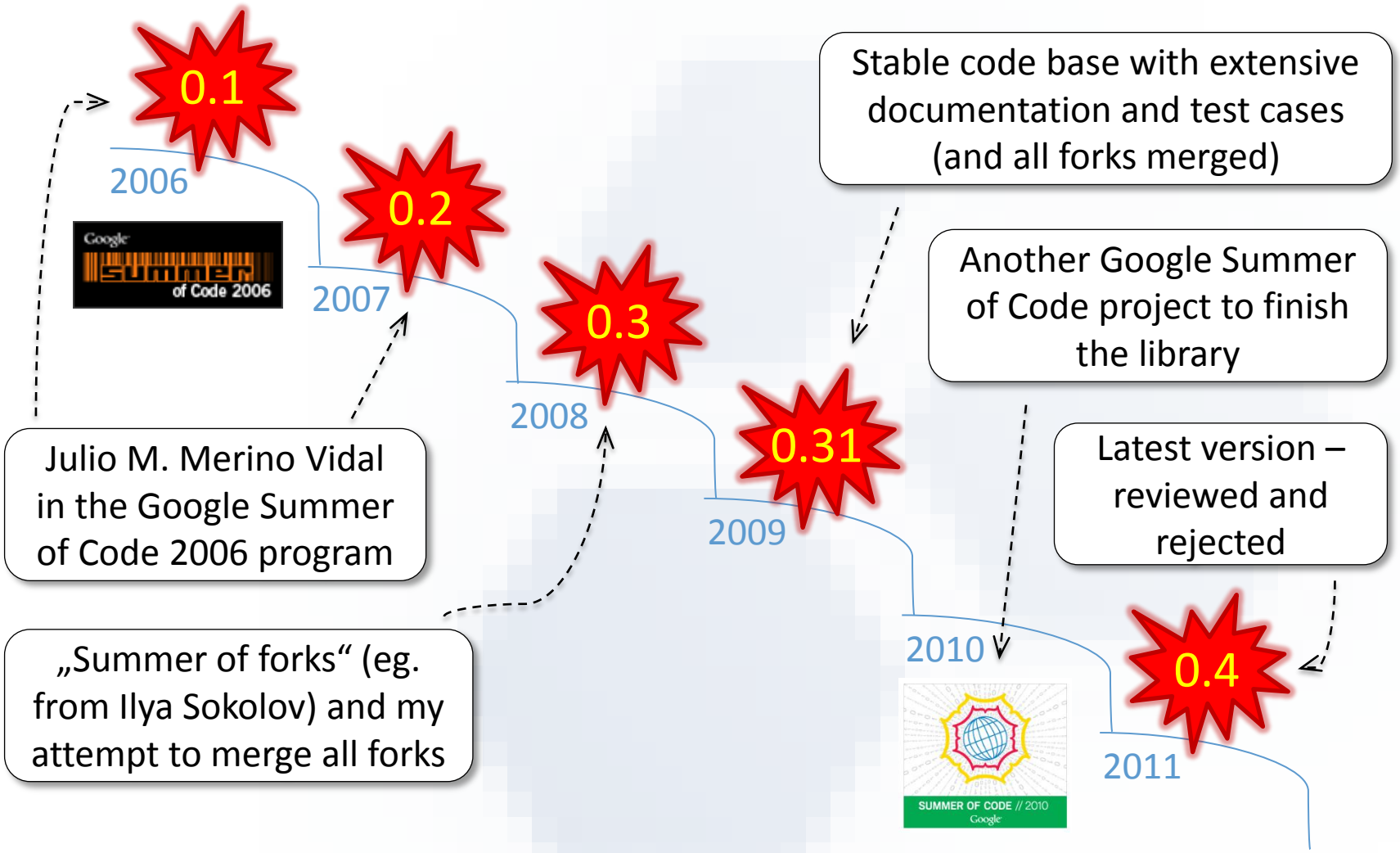
A simple interface requires to make assumptions which could be very wrong for some developers or certain use cases

Boost.Process: A simple library?

I want to be flexible and extend the library.

As supported platforms (Windows and POSIX) have lots of different concepts, it‘s difficult to define an extension mechanism

Boost.Process: Long road to ruin



Boost.Process: Abandoned concepts

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
context ctx;
child c = launch(exec, args, ctx);
```

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
posix_context ctx;
posix_child c = posix_launch(exec, args, ctx);
```

100 %

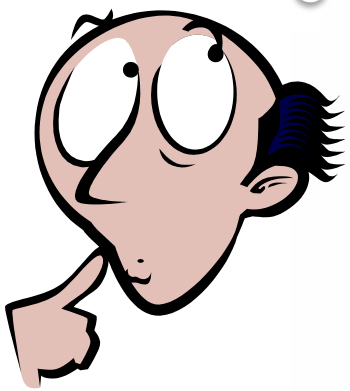
Error List

Item(s) Saved

Solution Explorer

Boost.Process: Abandoned concepts

Create generic classes for a minimum set of cross-platform features and create platform-specific classes for everything beyond.



Boost.Process would be three libraries in one as platform-specific classes would be rather important given the limited usefulness of generic classes.

Boost.Process: Abandoned concepts

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
context ctx;
ctx.process_name = "hostname";
ctx.work_dir = "C:\\";
ctx.env.insert(std::make_pair("new_variable", "value"));
create_child(exe, args, ctx);
```

100 %

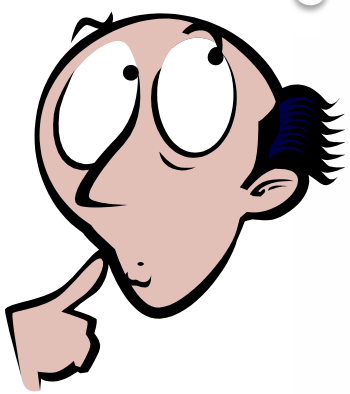
Error List Output Find Symbol Results

Item(s) Saved Ln 1 Col 1 Ch 1 INS

Solution Explorer

Boost.Process: Abandoned concepts

Create a context class to collect all settings required to launch a process and to configure its runtime context.



As settings are very different, context looks like a bunch of random variables without any clear design. Furthermore the context class is not extensible.

Boost.Process: Abandoned concepts

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
context ctx;
ctx.streams[stdout_id] = behavior::pipe();
child c = create_child(exe, args, ctx);
pistream is(c.get_handle(stdout_id));
std::cout << is.rdbuf() << std::flush;
```

100 %

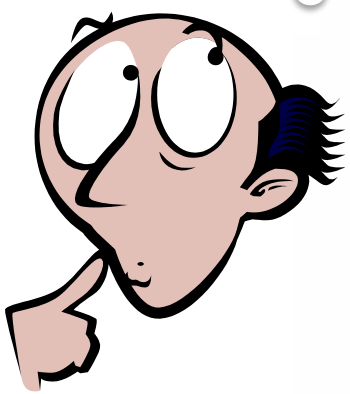
Error List Output Find Symbol Results

Item(s) Saved Ln 1 Col 1 Ch 1 INS

Solution Explorer

Boost.Process: Abandoned concepts

Create stream behaviors to define how streams of child processes should behave.



Stream behaviors are a superset of configuration options and are more abstract than eg. a boost::path variable to make a child process write to a file.

Boost.Process: Abandoned concepts

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
void setup(STARTUPINFOA &sainfo)
{
}

context ctx;
ctx.setup = &setup;
create_child(exe, args, ctx);
```

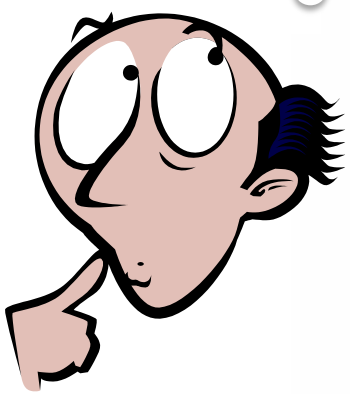
100 %

Error List Output Find Symbol Results

Item(s) Saved Ln 1 Col 1 Ch 1 INS

Boost.Process: Abandoned concepts

Add a function pointer (boost::function) to a context to call a user function just before a child process is started.



The context class and create_child() are still too much hardcoded as if calling a user function alone was flexible enough.

Boost.Process: Abandoned concepts

Asio.Test - Microsoft Visual C++ 2010 Express

File Edit View Project Debug Tools Window Help

main.cpp

(Global Scope)

```
void end_wait(const error_code &ec, int exit_code)
{
}

child c = create_child(exe, args);
status s(ioservice);
s.async_wait(c.get_id(), end_wait);
ioservice.run();
```

100 %

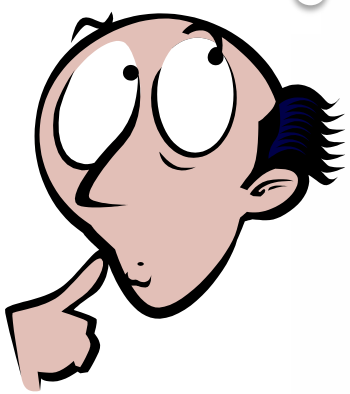
Error List Output Find Symbol Results

Item(s) Saved Ln 1 Col 1 Ch 1 INS

Solution Explorer

Boost.Process: Abandoned concepts

Support asynchronous I/O operations to communicate with child processes and to wait for child processes to exit.

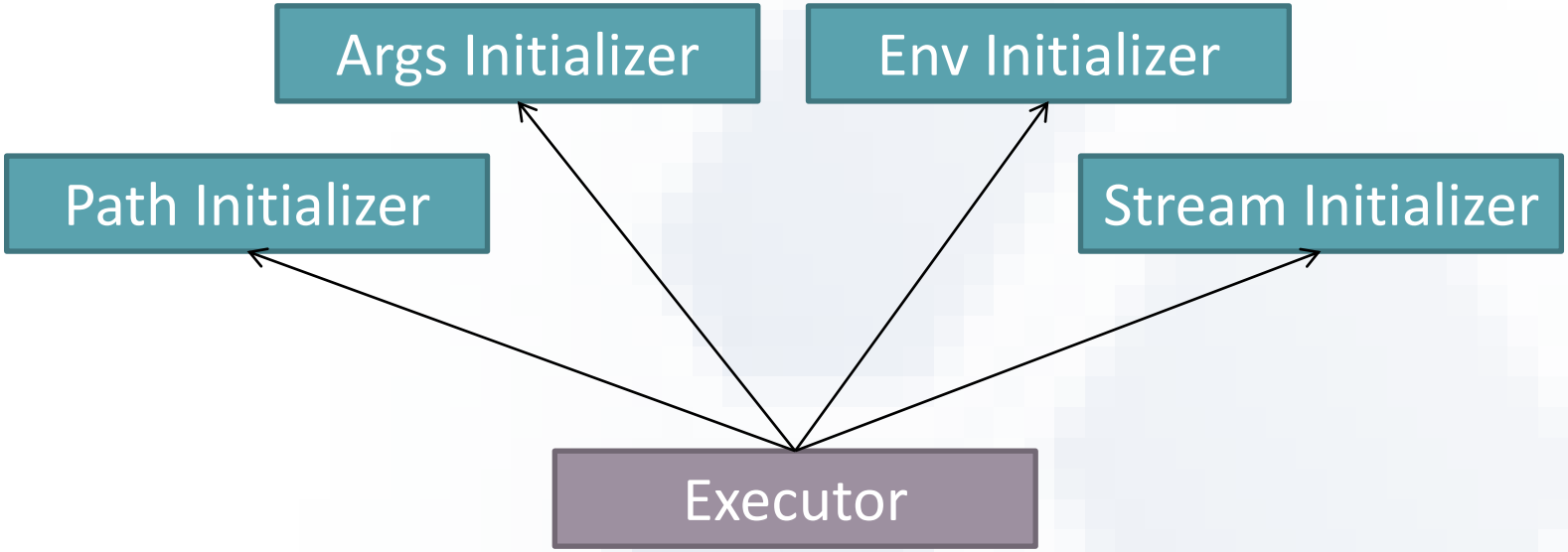


Waiting asynchronously for child processes to exit is done differently on Windows (WaitForMultipleObjects) and POSIX (signals).

New ideas: Jeff‘s executor concept

Context is replaced with an executor whose interface is not a bunch of public member variables but is initialized with initializers. Initializers are classes with pre_create(), post_create() and failed_create() member functions.

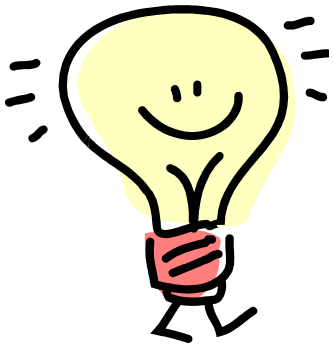
New ideas: Jeff‘s executor concept



- Initializers have their own specific interface (constructors)
- Initializers initialize the executor in pre_create()
- Initializers can clean up in post_create() and failed_create()
- Executor returns a handle on success (no child class needed)

New ideas: Platform-specific extensions

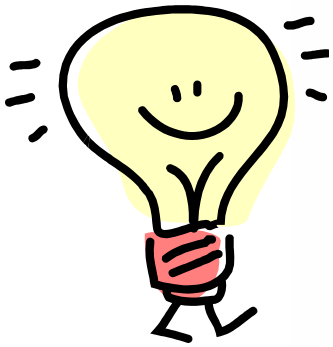
Waiting for child processes to terminate asynchronously will be supported via platform-specific Boost.Asio extensions.



Windows extension based on WaitForMultipleObjects, POSIX extension on signals (see Boost.Asio 1.5.3 or Trac #2879).

New ideas: Better implementations

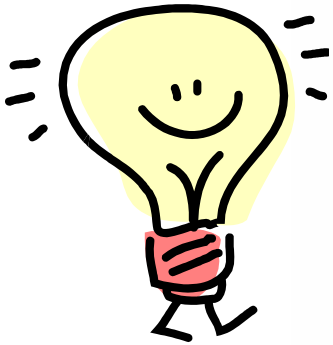
find_path_for_executable() will be implemented based on FileFindFirst() and FileFindNext() on Windows.



Current implementation based on SearchPath() finds directories which are called like executable and doesn‘t support resuming searching.

New ideas: Better implementations

On Windows COMSPEC will be used instead of a hardcoded path to cmd.exe. On POSIX users should know if fork() or execve() failed.



Current implementation is too limited. Additional flexibility can be provided without making the library unnecessarily complicated.

New ideas: Even more?