ChildProcess C++ library

v1.0.1

Table of contents

- Overview
- Versions
- Library files
- ChildProcess class description
  - ChildProcess class declaration
  - getVersion method
  - run method
  - close method
- Build and connect to your project
- Simple example

Overview

ChildProcess C++ library provides interface to run external process from your application in Linux. The library runs external process (child process) and returns control to you. The library doesn't show child process output, just runs. Also the library provides function to stop child process. The library doesn't have third-party dependencies and built with C++17 standard.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	16.08.2023	First version
1.0.1	16.01.2024	- Examples updated. - Documentation updated.

Library files

The library is supplied only by source code. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ---------------- Main CMake file.
src --------------------------- Folder with library source code.
    ChildProcess.cpp ---------- Library source code file.
    ChildProcess.h ------------ Library main header file.
    ChildProcessVersion.h ----- Header file with library version.
    ChildProcessVersion.h.in -- CMake service file to generate version header.
example ----------------------- Folder of example application.
    CMakeLists.txt ------------ CMake file of example application.
    main.cpp ------------------ Source C++ file of example application.
```

ChildProcess class description

ChildProcess class declaration

ChildProcess class declared in **ChildProcess.h** file. Class declaration:

```
class ChildProcess
{
public:

    /// Get library version.
    static std::string getVersion();

    /// Class constructor.
    ChildProcess();

    /// Class destructor.
    ~ChildProcess();

    /// Create a child process and run given command.
    bool run(std::string command, std::string params = "");

    /// Close child process.
    void close();
};
```

getVersion method

getVersion() method returns string of current version of **ChildProcess** class. Method declaration:

```
static std::string getVersion();
```

Method can be used without **ChildProcess** class instance:

```
cout << "ChildProcess class version: " << ChildProcess::getVersion() << endl;
```

Console output:

```
ChildProcess class version: 1.0.1
```

run method

run() method runs given command with provided arguments under new process. Method declaration:

```
bool run(std::string command, std::string params = "");
```

Parameter	Value
command	Command/program to run (for example: **mkdir** or **myProgram** etc).
params	Optional arguments for command (for example: -l, newDirectory or myProgramInput.txt etc).

Returns: TRUE if a new process is started or FALSE if not.

close method

close() method kills created child process. When the library runs child process the library remember PID of child process. Method declaration:

```
void close();
```

Build and connect to your project

Typical commands to build **ChildProcess** library:

```
cd ChildProcess
mkdir build
cd build
cmake ..
make
```

If you want connect **ChildProcess** library to your CMake project as source code you can follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

Create folder **3rdparty** in your repository and copy **ChildProcess** repository folder there. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    ChildProcess
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

###############################################################################
## 3RD-PARTY
## dependencies for the project
###############################################################################
project(3rdparty LANGUAGES CXX)

###############################################################################
## SETTINGS
## basic 3rd-party settings before use
###############################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

###############################################################################
## CONFIGURATION
## 3rd-party submodules configuration
###############################################################################
SET(${PARENT}_SUBMODULE_CHILD_PROCESS                   ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_CHILD_PROCESS)
    SET(${PARENT}_CHILD_PROCESS                         ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_CHILD_PROCESS_EXAMPLE                 OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
if (${PARENT}_SUBMODULE_CHILD_PROCESS)
```

```
    add_subdirectory(ChildProcess)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **ChildProcess** to your project and excludes example application from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    ChildProcess
```

Next you need include folder **3rdparty** in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include ChildProcess library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} ChildProcess)
```

Done!

Simple example

Example application runs given command with arguments and after 2 seconds stops child process.

```
#include <iostream>
#include <string>
#include <chrono>
#include <thread>
#include "ChildProcess.h"

int main(void)
{
    std::cout << "ChildProcess v" <<
    cr::utils::ChildProcess::getVersion() << " example" <<
    std::endl << std::endl;

    std::string command = "";
    std::cout << "Enter command: ";
    std::cin >> command;

    std::string arguments = "";
    std::cout << "Enter arguments: ";
    std::cin >> arguments;

    // Run child process.
```

```
    cr::utils::ChildProcess process;
    if (!process.run(command, arguments))
        return 0;

    // Wait 2 seconds for example.
    std::this_thread::sleep_for(std::chrono::seconds(2));

    // Close child process.
    process.close();

    return 1;
}
```