xmrig / src / base / kernel / **Platform_win.cpp**

xmrig Move Platform. 8b3f2d8 · 5 years ago History

```
/* XMRig
 * Copyright 2010      Jeff Garzik <jgarzik@pobox.com>
 * Copyright 2012-2014 pooler      <pooler@litecoinpool.org>
 * Copyright 2014      Lucas Jones <https://github.com/lucasjones>
 * Copyright 2014-2016 Wolf9466    <https://github.com/OhGodAPet>
 * Copyright 2016      Jay D Dee   <jayddee246@gmail.com>
 * Copyright 2017-2018 XMR-Stak    <https://github.com/fireice-uk>, <https://github.com
 * Copyright 2018      SChernykh   <https://github.com/SChernykh>
 * Copyright 2016-2019 XMRig       <https://github.com/xmrig>, <support@xmrig.com>
 *
 *   This program is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation, either version 3 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program. If not, see <http://www.gnu.org/licenses/>.
 */


#include <algorithm>
#include <winsock2.h>
#include <windows.h>
#include <uv.h>


#include "base/io/log/Log.h"
#include "Platform.h"
#include "version.h"


#ifdef XMRIG_NVIDIA_PROJECT
#   include "nvidia/cryptonight.h"
#endif


#ifdef XMRIG_AMD_PROJECT
static uint32_t timerResolution = 0;
#endif


```

```
    HMODULE ntdll = GetModuleHandleW(L"ntdll.dll");
    if (ntdll ) {
        RtlGetVersionFunction pRtlGetVersion = reinterpret_cast<RtlGetVersionFunction>(

        if (pRtlGetVersion) {
            pRtlGetVersion((LPOSVERSIONINFO) &result);
```

- res
- src
  - 3rdparty
  - backend
  - base
    - api
    - io
    - kernel
      - config
      - interfaces
      - Base.cpp
      - Base.h
      - Entry.cpp
      - Entry.h
      - Platform.cpp
      - Platform.h
      - Platform_mac.cpp
      - Platform_unix.cpp
      - Platform_win.cpp
      - Process.cpp
      - Process.h
      - Signals.cpp
      - Signals.h
    - net
    - tools
    - base.cmake
  - core
  - crypto
  - net
  - App.cpp
  - App.h
  - App_unix.cpp
  - App_win.cpp
  - Summary.cpp
  - Summary.h
  - config.json
  - donate.h
  - version.h
  - xmrig.cpp
- .gitignore

```
            pRtlGetVersion((LPOSVERSIONINFO) &result);
        }
    }

    return result;
}


char *Platform::createUserAgent()
{
    const auto osver = winOsVersion();
    constexpr const size_t max = 256;

    char *buf = new char[max]();
    int length = snprintf(buf, max, "%s/%s (Windows NT %lu.%lu", APP_NAME, APP_VERSION,

#   if defined(__x86_64__) || defined(_M_AMD64)
    length += snprintf(buf + length, max - length, "; Win64; x64) libuv/%s", uv_version
#   else
    length += snprintf(buf + length, max - length, ") libuv/%s", uv_version_string());
#   endif

#   ifdef XMRIG_NVIDIA_PROJECT
    const int cudaVersion = cuda_get_runtime_version();
    length += snprintf(buf + length, max - length, " CUDA/%d.%d", cudaVersion / 1000, c
#   endif

#   ifdef __GNUC__
    length += snprintf(buf + length, max - length, " gcc/%d.%d.%d", __GNUC__, __GNUC_MI
#   elif _MSC_VER
    length += snprintf(buf + length, max - length, " msvc/%d", MSVC_VERSION);
#   endif

    return buf;
}


bool Platform::setThreadAffinity(uint64_t cpu_id)
{
    if (cpu_id >= 64) {
        LOG_ERR("Unable to set affinity. Windows supports only affinity up to 63.");
    }

    return SetThreadAffinityMask(GetCurrentThread(), 1ULL << cpu_id) != 0;
}


uint32_t Platform::setTimerResolution(uint32_t resolution)
{
#   ifdef XMRIG_AMD_PROJECT
    TIMECAPS tc;

    if (timeGetDevCaps(&tc, sizeof(TIMECAPS)) != TIMERR_NOERROR) {
        return 0;
    }

    timerResolution = std::min<uint32_t>(std::max<uint32_t>(tc.wPeriodMin, resolution),

    return timeBeginPeriod(timerResolution) == TIMERR_NOERROR ? timerResolution : 0;
#   else
    return resolution;
#   endif
}


void Platform::restoreTimerResolution()
{
#   ifdef XMRIG_AMD_PROJECT
    if (timerResolution) {
        timeEndPeriod(timerResolution);
    }
#   endif
}

```

```
void Platform::setProcessPriority(int priority)
{
    if (priority == -1) {
        return;
    }

    DWORD prio = IDLE_PRIORITY_CLASS;
    switch (priority)
    {
    case 1:
        prio = BELOW_NORMAL_PRIORITY_CLASS;
        break;

    case 2:
        prio = NORMAL_PRIORITY_CLASS;
        break;

    case 3:
        prio = ABOVE_NORMAL_PRIORITY_CLASS;
        break;

    case 4:
        prio = HIGH_PRIORITY_CLASS;
        break;

    case 5:
        prio = REALTIME_PRIORITY_CLASS;
        break;

    default:
        break;
    }

    SetPriorityClass(GetCurrentProcess(), prio);
}


void Platform::setThreadPriority(int priority)
{
    if (priority == -1) {
        return;
    }

    int prio = THREAD_PRIORITY_IDLE;
    switch (priority)
    {
    case 1:
        prio = THREAD_PRIORITY_BELOW_NORMAL;
        break;

    case 2:
        prio = THREAD_PRIORITY_NORMAL;
        break;

    case 3:
        prio = THREAD_PRIORITY_ABOVE_NORMAL;
        break;

    case 4:
        prio = THREAD_PRIORITY_HIGHEST;
        break;

    case 5:
        prio = THREAD_PRIORITY_TIME_CRITICAL;
        break;

    default:
        break;
    }

    SetThreadPriority(GetCurrentThread(), prio);
}
```