Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Fri, 15 Sep 2023 01:46:11 GMT
View Forum Message <> Reply to Message

Any chance of using inotify instead? I wrote some example code that appears to work but likely has bugs and could probably be written better.

```
#include <Core/Core.h>
#include <ide/Core/Core.h>
#include <sys/inotify.h>

using namespace Upp;

void GatherAllFiles(const String& path, Index<String>& filei, VectorMap<String, String>& file)
{
    Sleep(0); // This is supposed to be superlazy
    for(FindFile ff(path + "/*.*"); ff && !Thread::IsShutdownThreads(); ff.Next())
        if(ff.IsFolder() && *ff.GetName() != '.')
            GatherAllFiles(ff.GetPath(), filei, file);
        else
        if(ff.IsFile()) {
            String p = NormalizePath(ff.GetPath());
            String lp = ToLower(p);
            if(filei.Find(lp) < 0) {
                filei.Add(lp);
                file.Add(GetFileName(p), p);
            }
        }
}

class INotify {
    int          fd         {};
    Index<String> directories {};
    Buffer<char>  buffer     {};
    Index<int>    watches    {};

    const int BUFFER_SIZE = 4096;

    Vector<String> add;
    Vector<String> remove;

public:
    void AddWatch(String directory) {
        add.Add(directory);

        if(directories.Find(directory) < 0) {
            auto wd = inotify_add_watch(fd, directory, IN_CREATE | IN_DELETE | IN_DELETE_SELF
```

```
| IN_MOVED_TO | IN_MOVED_FROM | IN_CLOSE_WRITE);

            if(wd == -1) {
                LOG("watch failed!");
                return;
            }

            directories.Add(directory);
            watches.Add(wd);
        }
    }

    void AddWatchRecursive(String path) {
        AddWatch(path);

        for(FindFile ff(path + "/*.*"); ff && !Thread::IsShutdownThreads(); ff.Next()) {
            if(ff.IsFolder() && *ff.GetName() != '.') {
                AddWatchRecursive(ff.GetPath());
            }
        }
    }

    void RemoveWatch(String directory) {
        auto idx = directories.Find(directory);

        if(idx >= 0) {
            inotify_rm_watch(fd, watches[idx]);
            directories.Unlink(idx);
            watches.Unlink(idx);
        }
    }

    void Run() {
        const size_t EVENT_SIZE     = ( sizeof (struct inotify_event) );
        const size_t EVENT_BUF_LEN  = ( 1024 * ( EVENT_SIZE + 16 ) );

        ssize_t numBytes = read(fd, buffer.begin(), BUFFER_SIZE);

        if(numBytes == -1 && errno != EAGAIN) {
            LOG("read failed!");
        }

        struct inotify_event* event = reinterpret_cast<struct inotify_event*>(buffer.begin());

        int i = 0;

        while(i < numBytes) {
            struct inotify_event *event = ( struct inotify_event * ) &buffer[i];
```

```
        auto idx = watches.Find(event->wd);

        if(idx < 0) break;

        String name = directories[idx];
        name.Cat("/");
        name.Cat(event->name);

        LOG(name);

        if(event->mask & IN_ISDIR) {
            if(event->mask & IN_CREATE) {
                DirCreated(name);
            }
            else if(event->mask & IN_DELETE) {
                DirDeleted(name);
            }
            else if(event->mask & IN_DELETE_SELF) {
                DirDeleted(name);
            }
            else if(event->mask & IN_MOVED_FROM) {
                DirMovedFrom(name);
            }
            else if(event->mask & IN_MOVED_TO) {
                DirMovedTo(name);
            }
        }
        else {
            if(event->mask & IN_CREATE) {
                FileCreated(name);
            }
            else if(event->mask & IN_DELETE) {
                FileDeleted(name);
            }
            else if(event->mask & IN_MOVED_FROM) {
                FileMovedFrom(name);
            }
            else if(event->mask & IN_MOVED_TO) {
                FileMovedTo(name);
            }
        }

        i += EVENT_SIZE + event->len;
    }

    directories.Sweep();
    watches.Sweep();
```

```
    }

    INotify() : fd(inotify_init1(IN_NONBLOCK)) {
        if(!fd) {
            LOG("inotify failed!");
        }

        LOG("inotify started");
        buffer.Alloc(BUFFER_SIZE);
    }

    ~INotify() {
        for(int i = 0; i < watches.GetCount(); i++) {
            inotify_rm_watch(fd, watches[i]);
        }

        LOG("inotify closed");
        close(fd);
    }

    Event<String> FileCreated;
    Event<String> FileDeleted;
    Event<String> FileMovedFrom;
    Event<String> FileMovedTo;

    Event<String> DirCreated;
    Event<String> DirDeleted;
    Event<String> DirMovedFrom;
    Event<String> DirMovedTo;
};

void IdeBackgroundThread()
{
    StdLogSetup(LOG_COUT);
    VectorMap<String, String> file;
    Index<String> dir;
    Index<String> filei;

    for(FindFile ff(ConfigFile("*.var")); ff && !Thread::IsShutdownThreads(); ff.Next()) {
        VectorMap<String, String> var;
        LoadVarFile(ff.GetPath(), var);
        for(String d : Split(var.Get("UPP", ""), ';'))
            dir.FindAdd(NormalizePath(d));
        Sleep(0);
    }

    INotify notify;
    notify.FileCreated   = [&](String name) { LOG(Format("File Created: %s", name));
```

```
filei.Add(name);       };
   notify.FileDeleted   = [&](String name) { LOG(Format("File Deleted: %s", name));
filei.RemoveKey(name); };
   notify.FileMovedFrom = [&](String name) { LOG(Format("File MovedFrom: %s", name));
filei.RemoveKey(name); };
   notify.FileMovedTo   = [&](String name) { LOG(Format("File MovedTo: %s", name));
filei.Add(name);       };
   notify.DirCreated    = [&](String name) { LOG(Format("Dir Created: %s", name));
dir.Add(name);         notify.AddWatchRecursive(name);    };
   notify.DirDeleted    = [&](String name) { LOG(Format("Dir Deleted: %s", name));
dir.RemoveKey(name);   notify.RemoveWatch(name); };
   notify.DirMovedFrom  = [&](String name) { LOG(Format("Dir MovedFrom: %s", name));
dir.RemoveKey(name);   notify.RemoveWatch(name); };
   notify.DirMovedTo    = [&](String name) { LOG(Format("Dir MovedTo: %s", name));
dir.Add(name);         notify.AddWatchRecursive(name);    };

   LOG(dir);

   for(String d : dir) {
      GatherAllFiles(d, filei, file);
      notify.AddWatch(d);
   }

   for(;;) {
      notify.Run();
      Sleep(100);
   }
}

CONSOLE_APP_MAIN {
   auto configFile = ConfigFile("*.var");

   LOG(ConfigFile("*.var"));
   IdeBackgroundThread();
}
```

I copied .var files over to the config directory for this project.
