Subject: [Fixed] TheIde continuously using CPU
Posted by jjacksonRIAB on Wed, 13 Sep 2023 13:39:37 GMT
View Forum Message <> Reply to Message

When I run TheIDE it will never stop using CPU while it's running - using about 2-3%. It did this even when I ran an empty project with no files at all and libclang disabled. By comparison a video player that is actively playing a video uses about 1.5% of CPU on my system. Any idea what it's doing? I run strace and see thousands and thousands of polling calls (resource unavailable).

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Wed, 13 Sep 2023 14:53:42 GMT
View Forum Message <> Reply to Message

After running it I saw the event loop, that was not the issue. I checked the other threads:

ide/background.cpp

```
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);
            }
        }
}
```

That Sleep(0) seems to be not very superlazy at all. If I set it to 10 the CPU usage drops. Could this be that Linux handles sleep differently from Windows? I can see the unix version is using nanosleep and I'm not sure what that does with a param of 0 but I'm assuming from what I've read it does nothing... which instead of making it lazy makes it expensive.

Subject: Re: TheIde continuously using CPU
Posted by mirek on Thu, 14 Sep 2023 09:43:51 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Wed, 13 September 2023 16:53After running it I saw the event loop, that was not the issue. I checked the other threads:

ide/background.cpp

```
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);
            }
        }
}
```

That Sleep(0) seems to be not very superlazy at all. If I set it to 10 the CPU usage drops. Could this be that Linux handles sleep differently from Windows? I can see the unix version is using nanosleep and I'm not sure what that does with a param of 0 but I'm assuming from what I've read it does nothing... which instead of making it lazy makes it expensive.

That is weird. If you look one level up

```
void IdeBackgroundThread()
{
 while(!Thread::IsShutdownThreads()) {
  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);
  }
  for(String d : dir)
```

```
   GatherAllFiles(d, filei, file);
  {
   Mutex::Lock __(s_allfiles_lock);
   s_allfiles = pick(file);
   s_allnests = dir.PickKeys();
  }
  for(int i = 0; i < 10 && !Thread::IsShutdownThreads(); i++)
   Sleep(100);
 }
}
```

It should wait before doing this for 1 second. The whole purpose of the exercise is to have somewhat actual list of all files of all assemblies (this is then used in comparison menu where now all files with the same name that are somewhat accessible through any assembly are listed - simplifies comparison sources between branches/versions). Putting Sleep(10) into GatherFiles would make it too long to happen.

Can you experiment with that loop at the end? IDK, maybe IsShutdownThreads is broken?

BTW, the idea behing Sleep(0) is to give up CPU is there is more important work to do (this is Thread::StartNice).

Mirek

---

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Thu, 14 Sep 2023 10:18:19 GMT
View Forum Message <> Reply to Message

Mirek,

If I set it back to Sleep(0) and then change Sleep(100) to Sleep(1000) in IsBackgroundThread() I get a spike of up to 3-5% (22-33% of a core) every few seconds and then it drops back to nothing.

IsShutdownThreads only appears to return true at TheIDE exit and the for loop at the end seems to be executing once every second.

Could this be related?

https://stackoverflow.com/questions/1125297/nanosleep-high-c pu-usage

I realize this is from 14 years ago but that code hasn't changed in a very long time either.

---

Subject: Re: TheIde continuously using CPU
Posted by mirek on Thu, 14 Sep 2023 14:04:42 GMT

jjacksonRIAB wrote on Thu, 14 September 2023 12:18Mirek,

If I set it back to Sleep(0) and then change Sleep(100) to Sleep(1000) in IsBackgroundThread() I get a spike of up to 3-5% (22-33% of a core) every few seconds and then it drops back to nothing.

Well, with Sleep(1000) you should be getting spike each 10 seconds (since the last spike).

With Sleep(100), it should be spike every second.

The length of spike should depend basically on the number of files accessible through all your assemblies.

Well, I was thinking this is innocent and that updated data is more important than small spike every second. But we can increase it a bit I guess..

Would Sleep(1000) work for you?

Mirek

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Thu, 14 Sep 2023 17:25:27 GMT


Mirek,

Yeah I pretty much saw it was going through a lot of stuff in there and I think a big reason for that is I was also putting subdirectories in there that, with the benefit of hindsight, I probably shouldn't. Maybe nothing needs to be changed, I probably just need to go through and move unneeded subdirectories out so they're not watched.

It does make me wonder what will happen if I put a symlink to / in there. It would probably watch everything, wouldn't it? :lol:

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Fri, 15 Sep 2023 01:46:11 GMT


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.

---

Subject: Re: TheIde continuously using CPU
Posted by mirek on Mon, 18 Sep 2023 07:08:12 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Thu, 14 September 2023 19:25Mirek,

Yeah I pretty much saw it was going through a lot of stuff in there and I think a big reason for that is I was also putting subdirectories in there that, with the benefit of hindsight, I probably shouldn't. Maybe nothing needs to be changed, I probably just need to go through and move unneeded subdirectories out so they're not watched.

It does make me wonder what will happen if I put a symlink to / in there. It would probably watch everything, wouldn't it? :lol:

Uhm, is 5% spike every 10 seconds such a big problem? theide is doing a lot of stuff in background anyway already...

Using INotify is probably fine, but a) it is new additional feature b) we would need Win32 and

MacOS versions as well.

Mirek

---

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Mon, 18 Sep 2023 12:58:41 GMT
View Forum Message <> Reply to Message

Mirek,

1) It's 3-5% as reported by KDE's system monitor which I'm assuming is an aggregate based on all cores and doesn't sample as frequently. On a single core it's closer to 60-90%. On top it becomes the #1 spot on the process list by far - nothing else uses that much CPU that frequently. Yes, other apps do spike but they spikes to more reasonable numbers. Example: firefox spikes to 13-25% while idle, plasma remains around 13-14% because it's never really idle, thunderbird spikes to around 13% (all of these are on top). The case could be easily made that theide is an outlier in this regard.

2) Yes, when I run using INotify the amount of CPU used drops considerably (top shows 0.7% at idle) and even libclang runs better. I get near instant popups on autocomplete whereas before it could sit there for a couple seconds waiting for them to show up.

I think it's worth investigating.

---

Subject: Re: TheIde continuously using CPU
Posted by mirek on Wed, 20 Sep 2023 12:10:25 GMT
View Forum Message <> Reply to Message

jjacksonRIAB wrote on Mon, 18 September 2023 14:58Mirek,

1) It's 3-5% as reported by KDE's system monitor which I'm assuming is an aggregate based on all cores and doesn't sample as frequently. On a single core it's closer to 60-90%. On top it becomes the #1 spot on the process list by far - nothing else uses that much CPU that frequently. Yes, other apps do spike but they spikes to more reasonable numbers. Example: firefox spikes to 13-25% while idle, plasma remains around 13-14% because it's never really idle, thunderbird spikes to around 13% (all of these are on top). The case could be easily made that theide is an outlier in this regard.

2) Yes, when I run using INotify the amount of CPU used drops considerably (top shows 0.7% at idle) and even libclang runs better. I get near instant popups on autocomplete whereas before it could sit there for a couple seconds waiting for them to show up.

I think it's worth investigating.

OK, I have refactored the thing, please try.

---

Subject: Re: TheIde continuously using CPU
Posted by jjacksonRIAB on Wed, 20 Sep 2023 17:03:11 GMT
View Forum Message <> Reply to Message

Mirek,

CPU usage is way down between 0.3% - 0.7% at idle (great news) but TheIDE crashes if I hover over the assist menu even if assist is disabled.

**EDIT** I cleared the cache completely, rebuilt and now it's working.

Looks good so far!