Eclipse & oostubs

Die Nutzung einer professionellen IDE bietet viele Vorteile, wie automatische Code-Vervollständigung, Syntax-Highlighting, Refactoring des Codes, einfacheres Manövrieren und Debuggen. Natürlich kann man alle Arbeit auch unter der Konsole verrichten, aber wer sich die Mühe macht einmal ein richtiges Projekt für Eclipse einzurichten, wird sehen, dass das Arbeiten danach umso leichter fällt.

1. Installation

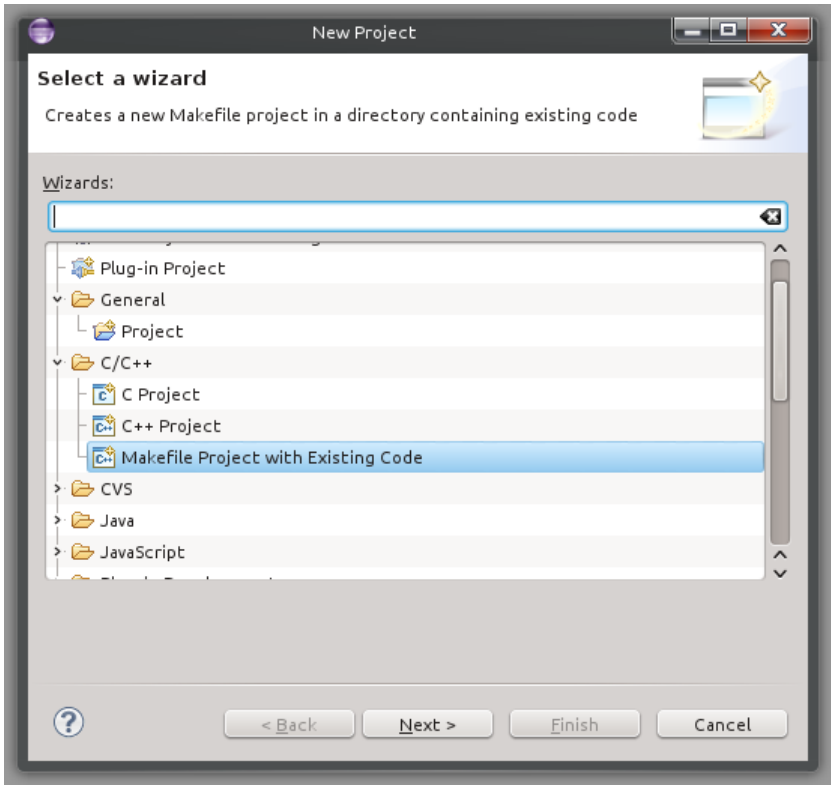
Eclipse kann kostenlos unter dem folgenden Link herunter geladen werden:

https://www.eclipse.org/downloads/

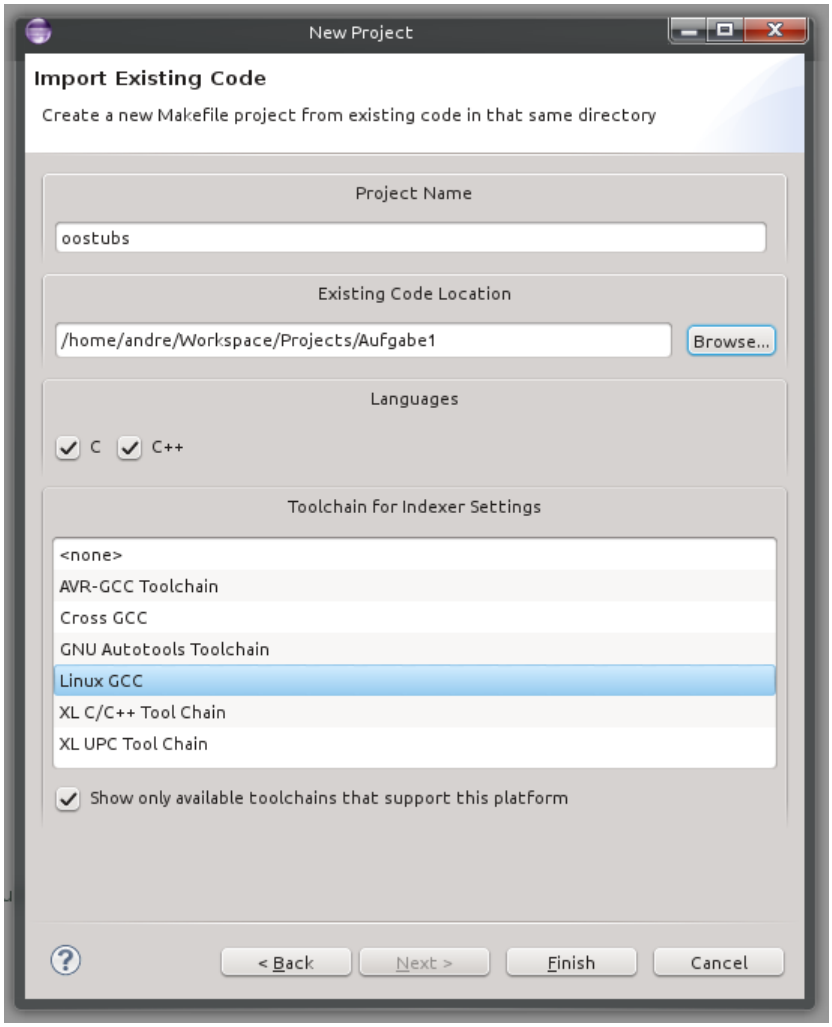
Es lohnt sich sofort die „Eclipse IDE for C/C++ Developers“ herunterzuladen, oder später muss noch CDT nachinstalliert werden. Nach dem Herunterladen einfach entpacken an den Ort eurer Wahl und ausführen.

2. Erstellen eines Projekts

Wenn ihr bereits Aufgabe1 heruntergeladen habt, einfach einen Rechtsklick im „Project Explorer“ tätigen und dann per „New → Project“ das neue Projekt als „C/C++ → Makefile Project with Existing Code“ zu Eclipse hinzufügen (siehe Abbildung).



Nachdem ihr auf „Next“ geklickt habt müsste dann der folgende Dialog erscheinen, den ihr bitte mit den auf eurem System geltenden Einstellungen vervollständigt („none“ ist glaube ich ausreichend).

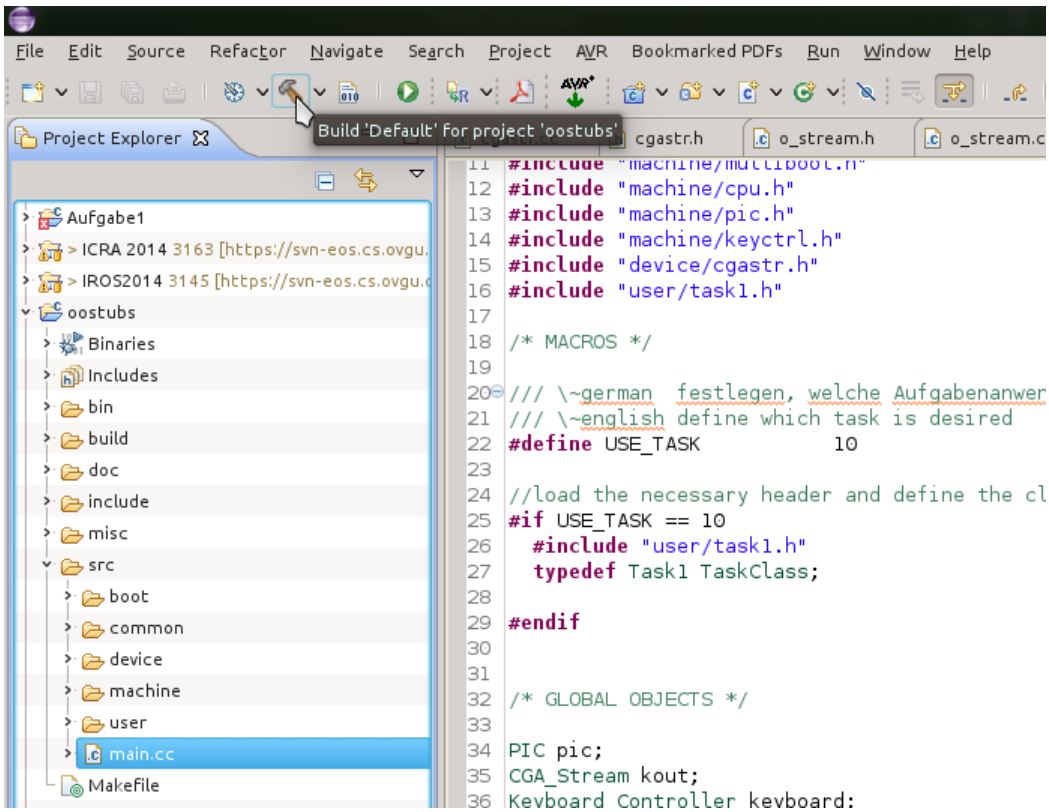


Nachdem ihr auf „Finish“ geklickt habt müsste ein neues Projekt erscheinen (siehe nächste Abbildung). In meinem Fall war dies „oostubs“ achtet im Folgenden darauf welchen Namen ihr eurem Projekt gegeben habt.

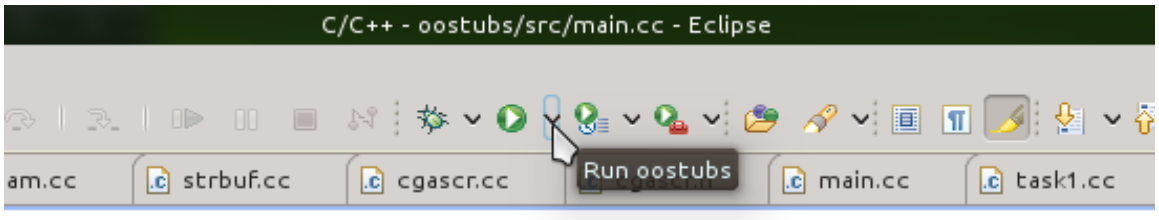
2. Kompilieren und Ausführen

Das Projekt zu kompilieren dürfte nicht schwer sein, einfach wie abgebildet auf den Hammer klicken und das Projekt müsste kompilieren. Es müssen keine weiteren Einstellungen vorgenommen werden, da nur die bereits enthaltenen Makefiles genutzt werden. Ihr könnt euer Projekt also immer noch per Hand kompilieren wie auf der Webseite beschrieben.

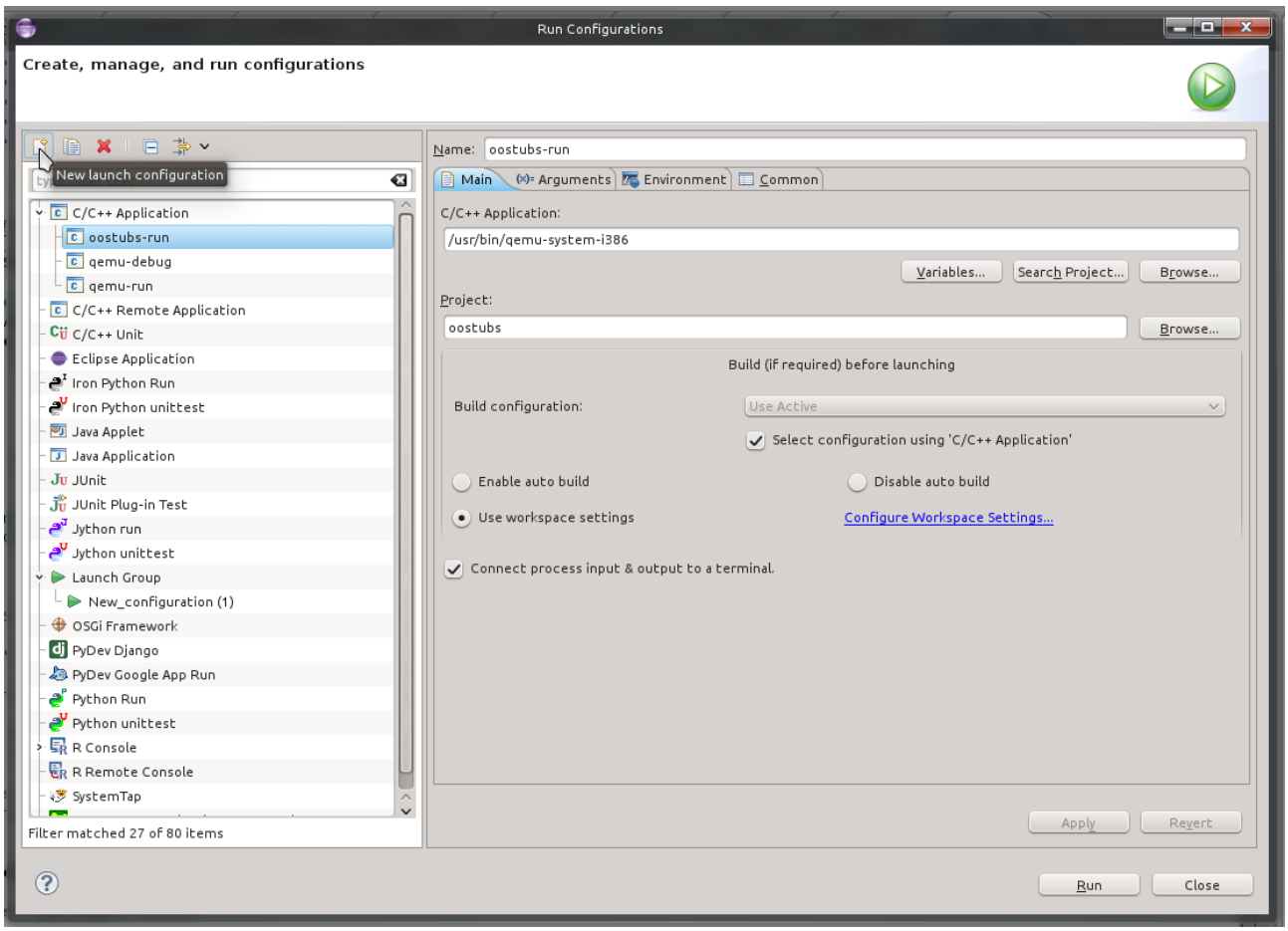
Es lohnt sich immer einen Blick in die Makefiles zu werfen, um zu erfahren welche Einstellungsmöglichkeiten vorgenommen werden können bzw. mit welchen Parametern make aufgerufen werden kann!



Ausführen ist nicht so leicht möglich, hier müsst ihr erst noch eine kleine Einstellung tätigen. Zwar wird oostubs kompiliert und erzeugt eine „ausführbare“ Datei, die sich im Verzeichnis „bin/oostubs“ befindet, diese kann jedoch nicht direkt ausgeführt werden, sondern nur mithilfe eines Emulator (in unserem Fall qemu). Das bedeutet, wir konfigurieren zunächst den grünen Playbutton (siehe Abbildung), dass dieser oostubs mithilfe von qemu ausführt.



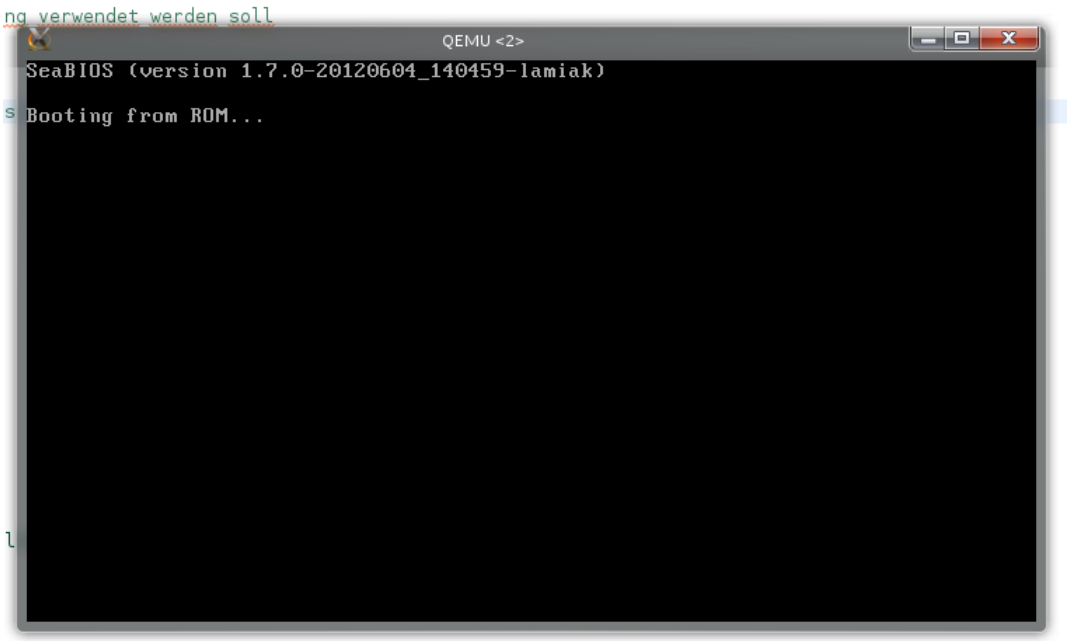
Dazu klickt ihr einfach auf den Pfeil „v“, wie in der Abbildung oben dargestellt, und dann auf den Menüpunkt „Run Configurations ...“. Es öffnet sich sofort ein neuer Dialog, wo ihr unter „C/C++ Application“eine neue Einstellung einfügt. Seht auf das untere Bild und wo sich der Mauszeiger befindet. Nehmt hier die Einstellungen vor, wie sie auch in der Abbildung dargestellt sind. Eigentlich sollte sich unter dem Eintrag „Application“ die kompilierte Datei befinden, ihr fügt aber hier den Pfad zu eurem qemu Emulator ein. Unter Linux befindet sich die ausführbare Datei oder der Link zu dieser meist im Verzeichnis „/usr/bin/“.



Klickt dann auf den Reiter „(x)=Arguments“ und tragt dort unter „Program arguments“ die folgenden Parameter ein:

-no-kvm -net none -vga std -kernel bin/oostubs

Die gleichen Parameter findet ihr auch in den Makefiles „make run“, sie wurden einfach nur rauskopiert … Ein abschließender klick auf „Apply“ und dann „Run“ sollte daraufhin zum ersten mal euer kleines noch mit Betriebssystemwissen zu füllendes Projekt starten.



Das heißt, wenn ihr das nächste mal auf den grünen Playbutton mit der Konfiguration „oostubs-run“ klickt dann wird euer Betriebssystem gleich wieder aufgeführt.

4. Debuggen

Ein Vorteil einer IDE wie Eclipse ist natürlich auch der Debugger, den ihr höchstwahrscheinlich brauchen werdet, da es noch kein „print“gibt ;)

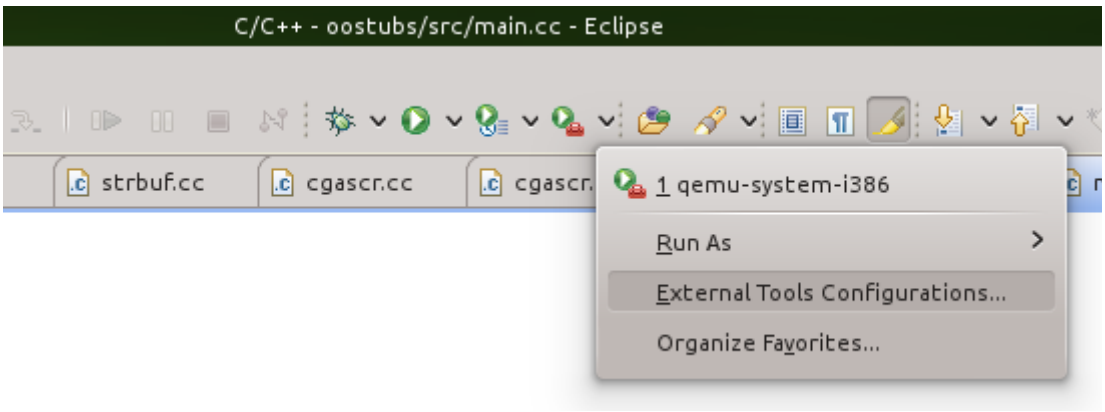
Als Debugger nutzen wir natürlich gdb. Eigentlich müssten wir einfach nur auf das Käfersymbol klicken und gdb müsste anspringen und uns mit interessanten Fakten überraschen, aber … Debuggt werden soll oostubs, welches im Emulator ausgeführt wird, deswegen müssen wir uns hier noch ein paar Sachen zurecht biegen.

4.1 QEMU im Debugmodus starten

Zum Glück gab es schon ein paar findige Informatiker die sich des Problems der quasi doppelten binär-Dateien bewusst waren, ihr wollt ja nicht qemu debuggen sondern das was damit ausgeführt wird, deshalb kann qemu bereits im Debugmodus gestartet werden und funktioniert dann als Server und nimmt gdb-Kommandos entgegen und gibt sie auch über tcp wieder nach draußen. Ein kleiner Blick in die Makefiles zeigt auch wie dies funktioniert:

```
debug: ${TARGET}
	@echo "(DEBUG)"
	${EMU} ${EMUFLAGS} $< ${EMUDEBUG} -S -s && ${RAWGDB} ${TARGET} ${GDBFLAGS}
```

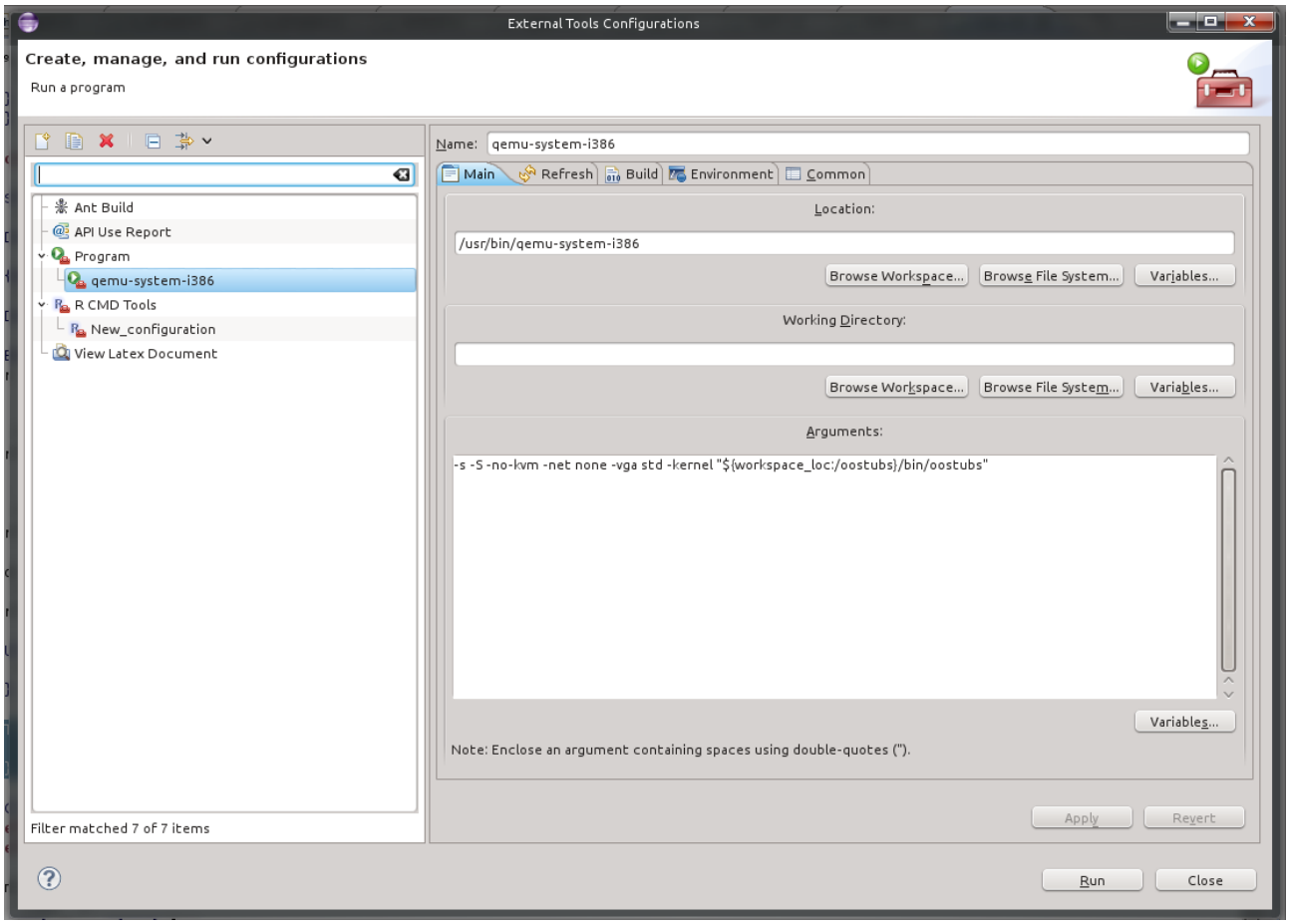
Der Inhalt der sich hinter diesem Kommando und den Variablen verbirgt muss nur noch Eclipse beigebracht werden. Wählt dazu zunächst den Eintrag „External Tools Configuration„aus, wie in der unteren Abbildung dargestellt.



Es erscheint der folgende Dialog, den Ihr wie abgebildet mit Inhalten füllt. Unter Programm einfach einen neuen Eintrag hinzufügen, wie ihr es auch zuvor für die „Run Configuration“ gemacht habt. Die einzutragenden Parameter sind:

-s -S -no-kvm -net none -vga std -kernel "${workspace_loc:/oostubs}/bin/oostubs"

Der einzige Unterschied zu den Parametern, die ihr schonmal eingetragen habt sind die beiden -s -S, die den Server (standard-Port 1234) starten und die Ausführung in der Emulationsumgebung anhalten, bis ihr euch mit eurem Debugger meldet und sagt, wann das Programm fortgesetzt werden kann.

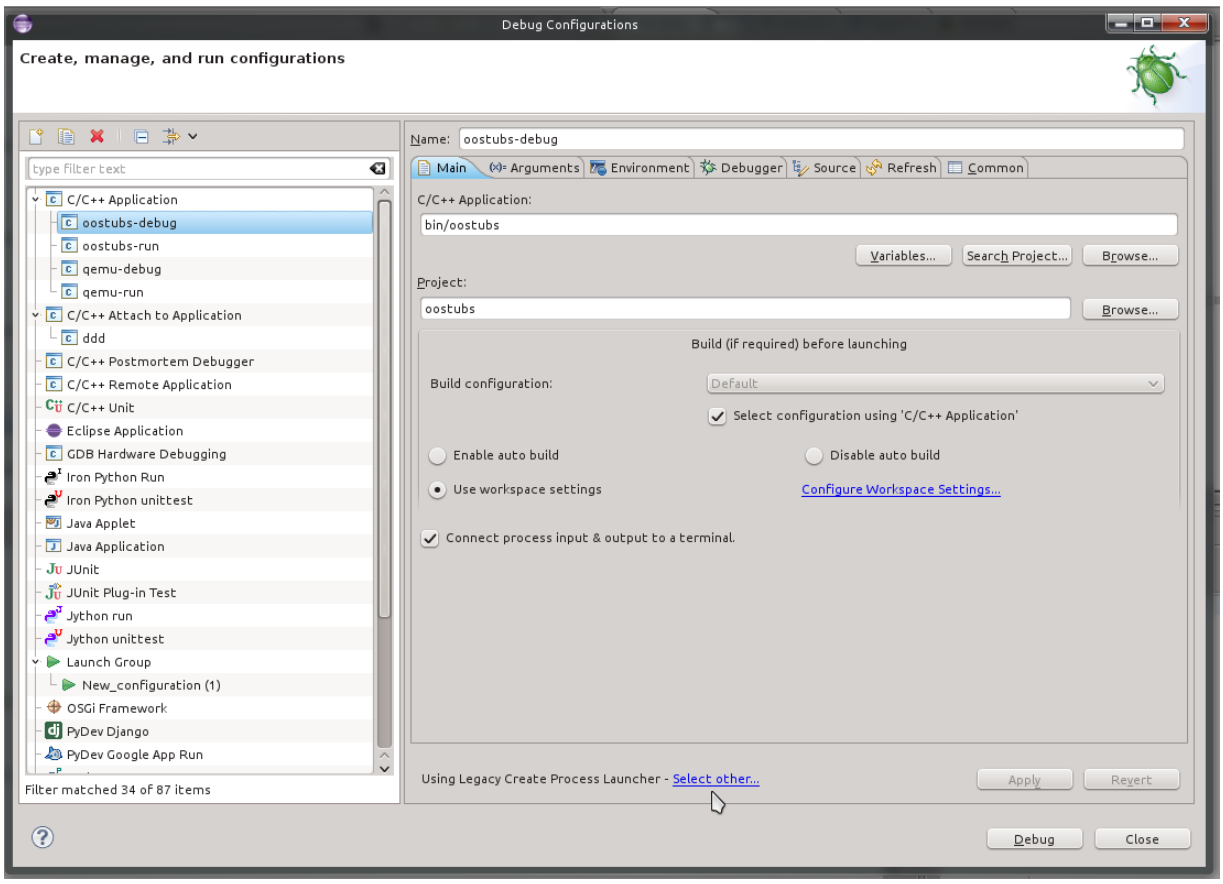


Wenn ihr dann auf „Apply“ und „Run“ klickt, merkt ihr, dass sich unter qemu nicht sonderlich viel bewegt ...

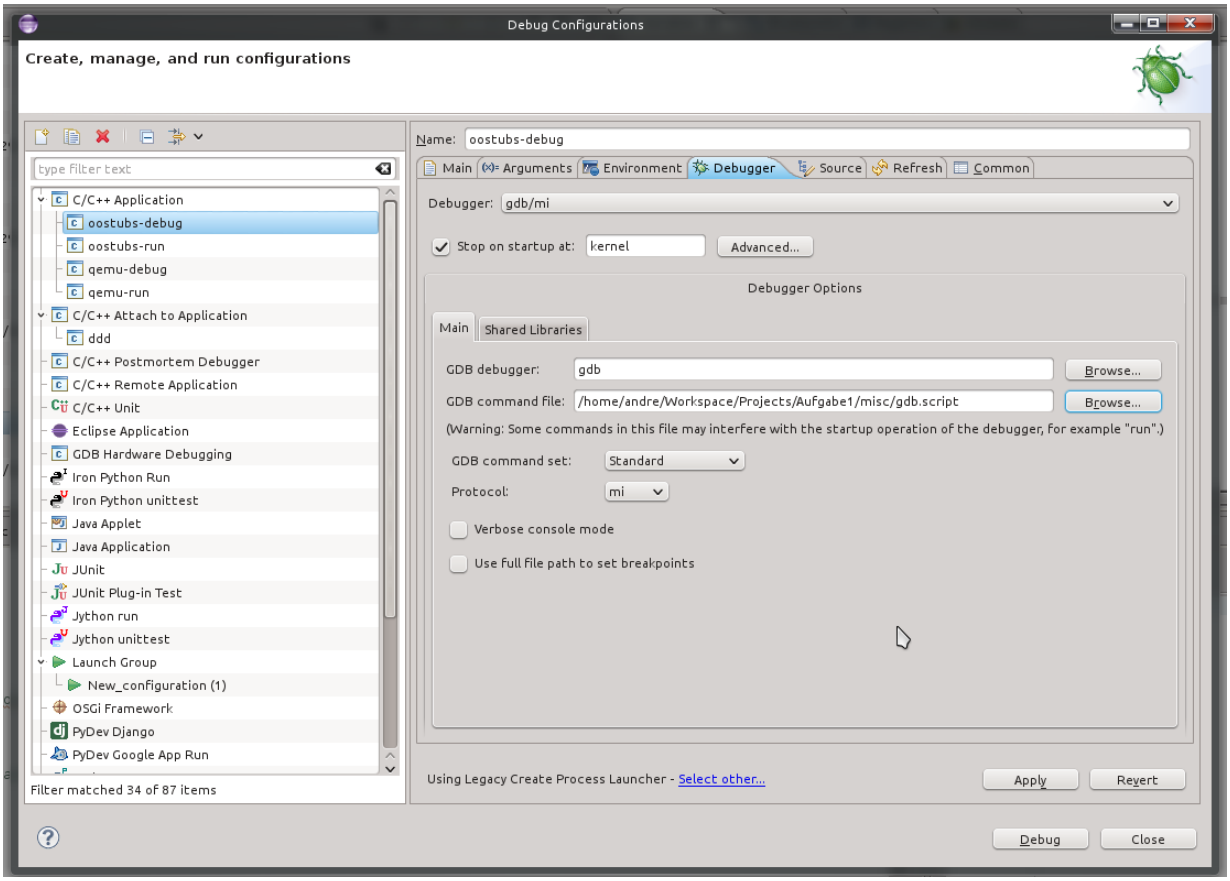
4.2 Eclipse-Better-Debugging

Jetzt müssen wir nur noch Eclipse mit dem „Debugserver“ verbinden. Hierzu wie gewohnt auf das DropDown-Menü des grünen Käfers klicken und den Menü-Punkt „Debug Configurations ...“ auswählen (ich glaube hier muss ich nicht noch einen Screenshot zeigen ;). Es erscheint wieder ein Dialog wie unten abgebildet.

Unter „C/C++ Applications“ fügt ihr wiederum ein neues Element ein, ich habe es „oostubs-debug“ genannt. Hier tragt ihr, nicht wie zuvor, im Textfeld unter „Application“ die tatsächliche Binärdatei ein „bin/oostubs“. Und ganz wichtig, dort wo der Mauszeiger abgebildet ist muss „Legacy Create Process Lauchner“ eingestellt sein! Wenn das nicht der Fall ist, klickt auch „Select other“ und wählt diesen aus!!!



Danach ist nur noch eine Einstellung notwendig. Geht auf den Reiter „Debugger“ und füllt diesen wie dargestellt aus:



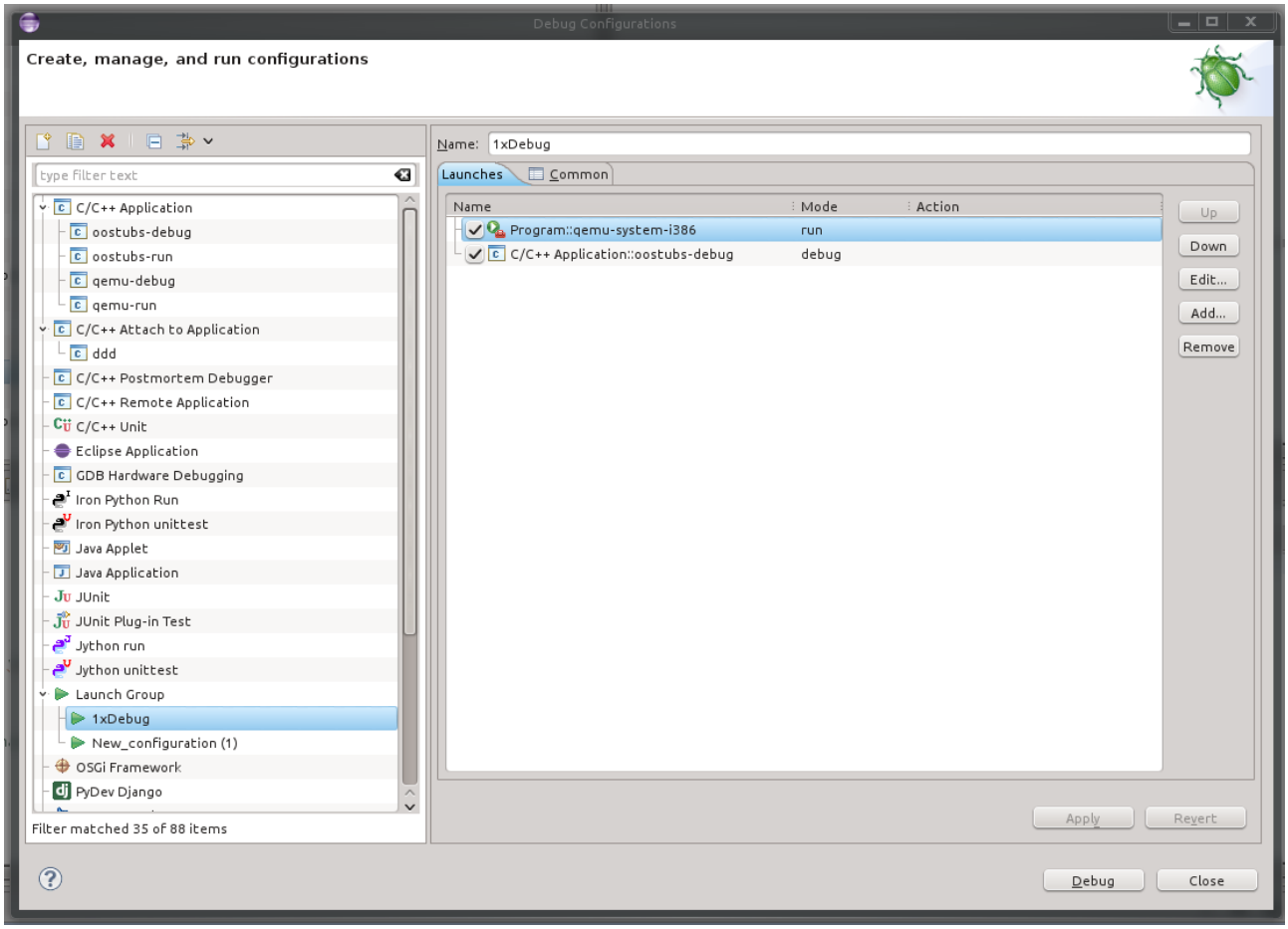
Im Textfeld „Stop on start at“ tragt ihr kernel ein, dass ist eure neue Hauptfunktion, eure main, eine main wie ihr es vielleicht gewohnt seid, gibt es im Projekt nicht.

„GDB command file“: in eurem Projekt im Verzeichnis „misc“ liegt eine kleine gdb-Konfigurationsdatei und enthält Informationen wie den über den Kommunikation-Port, diese wird hier einfach angegeben

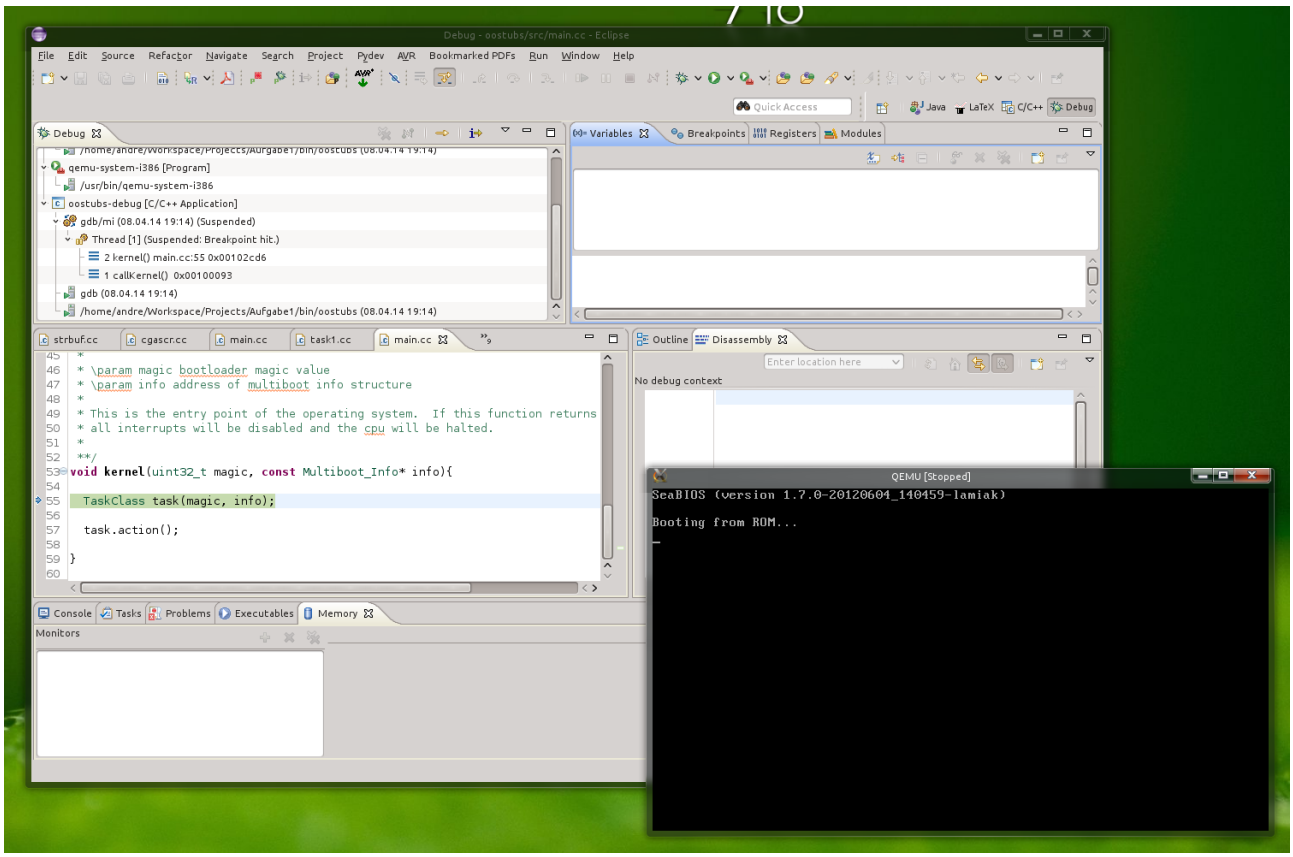
Das war es eigentlich alles, einfach auf „Apply“ und ihr habt alle Debug-Konfigurationen richtig eingestellt....

4.3 1xDebug

Damit ihr nicht immer zuerst qemo startet und dann separat den Debugger könnt ihr dass auch in einem Schritt tun.Es gibt noch den Punkt „Lauch Group“ wie im vorletzten Screenshot dargestellt. Hier könnt ihr ein neues Element erstellen, ich habe es „1xDebug“ genannt. Über „Add“ könnt ihr zusätzliche Elemente einfügen, die hintereinander ausgeführt werden sollen (beachtet den Modus). Wie gewohnt „Apply und Debug“ und der Debugger müsste automatisch anspringen.



Wenn ihr die Perspektive dann auf „Debug“ wechselt, müsste eine ähnliche Ansicht wie im letzten Screenshot zu sehen sein. Der graphische Debugger mit qemu zusammen. Ihr könnt nun komfortabel durch euren Code navigieren, euch die Inhalte von Variablen anzeigen lassen sowie den kompletten Speicher ....



Hoffentlich, viel Spaß beim Herumspielen ;)