FH-OÖ Hagenberg/ESD
Embedded Operating Systems, WS 2014
Rainer Findenig, Florian Eibensteiner, Josef Langer © 2014

1. Übung: Einführung, Embedded Linux

Name(n): Punkte:

1. Aufgabe *Starten des embedded Systems*

In diesem Teil der Übung soll das System zum ersten Mal gestartet werden. Befolgen Sie dazu die folgenden Schritte:

1. Schließen Sie die Platine über ein Nullmodemkabel an Ihren PC an und starten Sie ein Terminalprogramm (9600N1).
2. Stellen Sie sicher, dass sich Linux auf dem Board befindet. Starten Sie es dazu und beachten Sie die Ausgaben auf der seriellen Schnittstelle – wird U-Boot geladen, ist Linux bereits installiert.
3. Starten Sie den TFTP-, den DHCP-, und den NFS-Server (Script `sourceme`). Wozu werden diese Server benötigt? Beschreiben Sie die Abfolge der Interaktionen zwischen dem Board und den Servern! □
4. Verbinden Sie die Platine und den PC über ein Netzwerkkabel. *Hinweis:* In dieser Übung werden Sie einen PC mit zwei Netzwerkkarten verwenden. Die Platine wird dabei nicht an einen Switch, sondern über ein gekreuztes Netzwerkkabel direkt an den PC angeschlossen.
5. Schalten Sie die Platine ein. Welches Programm ist für die ersten Ausgaben über die serielle Schnittstelle verantwortlich? Beschreiben Sie den weiteren Bootverlauf! □

Nach dem Bootvorgang sollte über die serielle Schnittstelle eine funktionierende Konsole zur Verfügung stehen. Versuchen Sie, Informationen über die CPU auszulesen (`cat /proc/cpuinfo`). □

2. Aufgabe *Entwicklungsumgebung für embedded Linux*

Spätestens seit *Visual Studio* und *Eclipse* ist ein Entwickler für Programme, die später auf einem PC laufen sollen, ständig von grafischen Benutzeroberflächen umgeben, die eine Integration der gesamten *tool chain* bieten. Diese sogenannten *integrated development environments* (IDEs) vereinfachen zwar den Entwicklungsprozess, sind jedoch nicht unbedingt für alle Plattformen vorhanden.

Grundsätzlich vereint eine solche Entwicklungsumgebung zumindest die folgenden Teile:

- den Compiler bzw. Cross-Compiler,
- einen Assembler, Linker und ähnliche Programme,
- einen Debugger und
- einen Editor.

Für die Entwicklung eines embedded Systems brauchen Sie alle diese Teile; ob diese in einer grafischen Benutzeroberfläche zusammengefasst oder als einzelne Programme sichtbar sind, ist jedoch irrelevant.

Für die vorliegende Übung werden Sie als Cross Compiler die *Gnu Compiler Collection* (`gcc`) verwenden. Die *GNU Binutils* stellen (neben vielen anderen nützlichen Programmen) den Linker (`ld`) und den Assembler (`as`) bereit. Als Debugger wird der *GNU Project Debugger* (`gdb`) verwendet. Die Frage nach dem richtigen Editor muss an dieser Stelle offen bleiben, diverse *flame wars* im Internet beschäftigen sich jedoch ausführlich damit.

Da auf Linux-Systemen meistens mehrere Installationen (zB. für verschiedene Plattformen) der *Gnu Compiler Collection* installiert sind, wird allen Programmnamen der *tool chain* ein Präfix vorangestellt; in unserem Fall ist dieses `arm-linux-`. Der Compiler heißt also `arm-linux-gcc`, der Linker `arm-linux-ld`, der Debugger `arm-linux-gdb`, usw.

Ihre erste Aufgabe bei der Verwendung einer neuen *tool chain* besteht meistens darin, diese Programme auszuprobieren und zu verstehen, welcher Teil der Aufgaben durch welches Programm durchgeführt wird. Alle dazu nötigen Informationen lassen sich den *man-pages* (zB. `man gcc`) entnehmen.

□ Beantworten Sie dazu folgende Fragen:

1. Welche Version von `arm-linux-gcc` wird eingesetzt?
2. Welche Version der *GNU Binutils* wird eingesetzt?
3. Mit Hilfe des Programms `file` kann in Linux der Typ einer Datei festgestellt werden. Schreiben Sie ein möglichst kurzes Programm (zB. “Hello World”), compilieren[1] Sie dieses einmal für den Host (`gcc`) und einmal für das Target (`arm-linux-gcc`). Geben Sie für beide Fälle an, welche Informationen `file` über das erstellte Programm ausgibt, und vergleichen Sie diese Informationen. Was bedeuten die Angaben “ELF executable”, “dynamically linked” und “not stripped”?
4. Compilieren Sie ein kurzes Programm mit `arm-linux-gcc -c`. Was bedeutet diese Option? Welche Art von Datei wird erzeugt?
5. Über den laufenden NFS-Server wird ein Netzwerk-Dateisystem als Root-Filesystem für das Toradex-Board bereitgestellt. Mit welcher Datei werden die exportierten Dateibäume konfiguriert. Welchen Inhalt hat diese Datei?

3. Aufgabe *"‘Sinnvolle"’ Programme mit embedded Linux*

Erstellen Sie ein C-Programm (nicht C++!), das 1000 Zufallszahlen zwischen 0 und 999 generiert und in einer Datei in `/tmp` ablegt. Compilieren Sie dieses Programm sowohl für Ihren Host-PC als auch für das Target. Führen Sie beide Programme aus, und messen Sie dabei die zur Ausführung benötigte Zeit (Tip: `man time`). Stellen Sie die Ergebnisse gegenüber. Erhöhen Sie gegebenenfalls die Anzahl der generierten Werte, um sinnvolle Messergebnisse zu erhalten.

[1] Der Aufruf von `gcc` kann dabei zB. so aussehen: `gcc -g hello.c -o hello`.

Hinweise:

- Verwenden Sie statt der Funktionen `fopen()`, `fprintf()` usw. die Funktionen `open()`, `write()` usw. Aufgrund eines Bugs in der am embedded System verfügbaren C-Bibliothek können die erstgenannten Funktionen zu unerklärlichen Abstürzen führen.
- Die Compiler-Flags `-Wall` und `-pedantic` erweisen sich beim Entwickeln größerer Programme oft als sehr hilfreich.

Debugging: Sie können die Software auf dem embedded System von Ihrem PC aus debuggen. Starten Sie dazu das zu debuggende Programm mit dem Programm `gdbserver`:

```
target # gdbserver localhost:1234 myprog
```

Am Host können Sie einen beliebigen Debugger mit `gdb`-Backend verwenden. Ein Beispiel dafür ist der *Data Display Debugger* (ddd). Starten Sie diesen:

```
host $ ddd --debugger arm-linux-gdb
```

Verbinden Sie dann die Debug-Sitzung am Host mit dem Target:

```
(gdb) target remote 192.168.1.199:1234
```

Danach öffnen Sie mit *File/Open* die entsprechende Datei (am Host!), setzen einen Breakpoint an den Beginn des Hauptprogramms und können das Programm mit *cont* (nicht *run*!) starten.

"panic("No CPUs found. System halted.\n");"
linux-2.4.3/arch/parisc/kernel/setup.c