Fachprojekt „Systemsoftwaretechnik“
A0 – Der Linux-Kern

0 Erste Schritte mit und im Linux Kernel

In dieser Aufgabe richtet Ihr Eure Arbeitsumgebung für das weitere Fachprojekt ein. Außerdem macht Ihr Euch mit `git` vertraut und könnt Eure Aufgaben dort verwalten. In einer virtuellen Maschine (`VM`) installiert Ihr Euch ein aktuelles Debian mit dem von Euch konfigurierten und gebauten Kernel.

Hierzu arbeitet Ihr Euch mit den einzelnen Aufgaben Stück für Stück durch die Werkzeuge: Beginned bei dem Umgang mit `git`, über das erstellen einer virtuellen Maschine und bis hin zum Konfigurieren und Übersetzen Eures eigenen Linux-Kerns. Am Ende des Aufgabenblatts verfügt Ihr über eine Entwicklungsumgebung, um selbst an dem Linux-Kern zu arbeiten.

0.1 Abgabe-Repository einrichten

Jede Gruppe erhält ein eigenes Projekt–Repository im Gitlab (`https://git.cs.tu-dortmund.de`). In diesem Repo sollen Eure Aufgaben verwaltet und zur Korrektur abgegeben werden.

Erstellt einen lokalen Checkout Eures Abgabe-Repositories in Eurer Arbeitsumgebung. Fügt einen Text der Datei `README.md` hinzu und markiert den Commit mit dem Tag „init-commit“.

0.2 Erstellen eines lauffähigen Linux-Systems

Um das Arbeiten am Linuxkern möglichst einfach und effizient zu gestallten, werden wir Linux in einer virtuellen Maschine (`VM`) ausführen. Im Rahmen des Praktikums nutzen wir `kvm/QEMU`. Zur einfachen Nutzung der `VM`, steht das Skript `boot.sh` zur Verfügung. Macht Euch mit dem Skript und den verwendeten `QEMU` Flags vertraut.

Da ein Linuxkern alleine nicht hilfreich ist, installiert im ersten Schritt eine Linux-Distribution. Verwendet bitte hierzu bitte das bereitgestellte ISO-Abbild von Debian[1]. Das ISO-Abbild findet Ihr unter `/fs/proj/sst/debian-12.5.0-amd64-netinst.iso`.

Für die `VM` wird außerdem eine virtuelle Festplatte (8 GB) benötigt. Diese könnt Ihr entweder in Euer Home legen oder unter `/fs/scratch/` ablegen. Bei letzterem müsst Ihr Euch aber jedes Mal an dem gleichen Rechner anmelden, um das Image zu verwenden.

0.3 Kernelquellen beschaffen, konfigurieren und übersetzen

Ziel des Praktikums ist es, einen selbstgebauten Kernel zu nutzen. Ladet Euch dazu zunächst den Quellcode des Linux-Kerns via `git` runter. Nutzt als Quelle für Euren Checkout eine offizielle Quelle für den Linux-Quellbaum, wie z. B. Github (`https://github.com/gregkh/linux`) oder `kernel.org` (`https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git`).

Achtung: Die Kernelquellen sind ein *sehr großes* Projekt. Ein Checkout des kompletten Repositories, inklusive Historie, kann demenstprechend einige Zeit in Anspruch nehmen und viel Speicherplatz belegen. Da im Praktikum ausschließlich mit Version 6.1.55 des Kernels gearbeitet werden soll, genügt es nur den entsprechenden Tag zu beziehen.

Bevor Ihr den Kern übersetzen könnt, müsst Ihr eine Kernel-Konfiguration erstellen. Die Konfiguration muss mindestens die passenden Treiber für die „Hardware“ und die Virtualisierung enthalten. Informiert Euch hierzu über die offizielle Kernel-Dokumentation im Internet über die verschiedenen `Makefile`-Targets,

[1] Ihr dürft auch eine Distribution Eurer Wahl einsetzen. Bei etwaigen Fehlern können wir Euch möglicherweise nur schwer helfen.

insbesondere die `default`-Targets[2]. Notiert Euch, welches `Target` Ihr letztlich zum Übersetzen genutzt habt. Achtet außerdem auf die letzte Zeile der Ausgabe des Übersetzungsvorgangs. Dieser hilft Euch beim Starten der `VM`.

0.4 Booten der `VM` mit eigenem Kernel und Teilen von Daten

Den selbstkonfigurierten und übersetzten Kern könnt Ihr mit der Umgebung, die Ihr in 0.2 erstellt habt, starten. Verwendet das bereitgestellte `boot.sh`-Skript und denkt an den allerletzten Teil der Aufgabe .
Um Dateien zwischen dem Host- und dem Gastsystem zu teilen, soll ein virtuelles Dateisystemgerät (`9pfs`) genutzt werden. Stellt sicher, dass Euer Kernel das Dateisystem `9p` unterstützt und richtet einen entsprechenden Ordner „vmshare“ in der `VM` ein, durch den Ihr auf das Gegenstück auf dem Hostsystem zugreifen könnt. Bindet hierzu das geteilte Verzeichnis in der `VM` wie folgt ein:

```
stud@sst: ~$ mkdir /mnt/DIR
stud@sst: ~$ mount -t 9p -o trans=virtio,version=9p2000.L vmshare /mnt/DIR
```

Challenge: Wer schafft den kleinsten Kernel?

Durch gezielte Konfiguration des Kernels kann die Größe – und damit auch die Übersetzungszeit – reduziert werden. Versucht einen kleinstmöglichen Kernel zu konfigurieren und zu bauen, mit dem die in 0.2 erstellte `VM` bootet und weiterhin Dateien mittels dem `9p` Dateisystem geteilt werden können.

Abgabe bis zum 25.04.2024:

1. Vorstellen der eingerichteten Arbeitsumgebung in der Rechnerübung:
   - ☐ Tag „init-commit“ im Abgabe-Repository
   - ☐ Funktionierende Debian `VM` mit selbstkonfiguriertem Kernel
   - ☐ Funktionierender Dateienaustausch via „vmshare“ zwischen Host und `VM`
2. Einchecken der Kernelconfig im Abgabe-Repository (Tag „a0“).
   Achtung: Achtet darauf, nur die Kernelkonfiguration und **nicht** das Kernel-Binary an sich oder die gesamten Kernelquellen einzuchecken.

Allgemeine Hinweise:

- Das `boot.sh`-Skript erstellt bei der Installation der `VM` einen „vmshare“ Ordner im Pfad der virtuellen Festplatte. Dieser soll für die Dateienübertragung zwischen Gast und Host genutzt werden.
- Das `boot.sh`-Skript leitet den Netzwerkport 22 an den Port `4711` des Hosts weiter. So könnt Ihr, sofern SSH installiert ist, Euch via SSH in der `VM` anmelden. Ihr müsst dem `ssh`-Kommando nur die richtigen Parameter für Host und Port angeben – siehe `man ssh`.
- Mithilfe der `make`-Option `'O=<directory>'` könnt Ihr ein extra Ausgabeverzeichnis für den Build-Prozess angeben. Somit kann man mehrere Konfigurationen „nebeneinander“ bauen, wenn man das `'O=<directory>'`-Flag auch bei `make menuconfig` mit angibt.
- Bitte baut einen 64-Bit Kernel (Architektur: `amd64`).
- Bei der Bewertung achten wir insbesondere auf selbständige Arbeitsweise und Problemlösung sowie die sichere Beherrschung der grundlegenden Tätigkeiten durch alle Gruppenmitglieder.

[2] https://www.kernel.org/doc/html/latest/kbuild/kconfig.html