Linux Configuration V6.2

0.1 Einführung

Dieses Dokument dient zur Beschreibung von diversen Einstellungen bei der Konfiguration mittels `make menuconfig` unter Linux.
Es wird nicht näher darauf eingegangen, wie der Kernel kompiliert wird oder welche Voreinstellungen, Programme etc. zum Kompilieren benötigt werden.
Zu Beginn der jeweiligen Konfigurationszeile wird der Standardwert (Default) angezeigt. Mein Vorschlag folgt danach.
Z. B. bei CONFIG_WERROR [=n] **[Y]**
Hier ist der Standarwert ein Nein [n], meine Einstellung ein Ja [Y].

1 General setup →

1.1 Compile also drivers which will not load

CONFIG_COMPILE_TEST [=n] [ ]
Einige Treiber können auf einer anderen Plattform kompiliert werden als auf der, für die sie gedacht sind. Obwohl sie dort nicht geladen werden können (oder selbst wenn sie geladen werden können, können sie aufgrund fehlender Hardware-Unterstützung nicht verwendet werden), möchten Entwickler, im Gegensatz zu Distributoren, solche Treiber vielleicht trotzdem kompilieren und testen.

1.2 Compile the kernel with warnings as errors

CONFIG_WERROR [=n] **[Y]**
Ein Build sollte keine Compiler-Warnungen ausgeben, dies aktiviert die Flags '-Werror' (für C) und '-Dwarnings' (für Rust) um diese Regel standardmäßig zu setzen. Bestimmte Warnungen von anderen Tools z.B. der Linker könnte mit dieser Option Fehler generieren. Deaktivieren ist sinnvoll, wenn Sie einen neuen (oder sehr alten) Compiler bzw. Linker mit seltenen, ungewöhnlichen Warnungen haben. Haben Sie auf Ihrer Architektur Probleme, dann müssen Sie diese Konfiguration deaktivieren, um den Kernel erfolgreich zu bauen. Im Zweifelsfall sagen sie Y für Ja.

1.3 Local version – append to kernel release

CONFIG_LOCALVERSION [=] [ ]
Type: string
Hängen Sie eine zusätzliche Zeichenkette an das Ende Ihrer Kernelversion an.
Dies wird angezeigt, wenn Sie z. B. `uname` eingeben. Die hier angegebene Zeichenfolge wird an den Inhalt von einem Dateinamen mit `localverion*` als Objekt und im Quellbaum, in dieser Reihenfolge angezeigt. Die Zeichenkette darf maximal 64 Zeichen lang sein.

1.4 Automatically append version information to the version string

CONFIG_LOCALVERSION_AUTO [=y] [**Y**]
Dies versucht automatisch festzustellen, ob der aktuelle Baum ein Release-Tree ist, indem es nach **Git**-Tags sucht, die zur aktuellen Top-of-Tree-Revision gehören.
Eine Zeichenkette des Formats `-gxxxxxxxx` wird der lokalen Version hinzugefügt, wenn ein git-basierter Baum gefunden wird. Die so erzeugte Zeichenkette wird nach allen passenden „localversion*"-Dateien und nach dem in CONFIG_LOCALVERSION eingestellten Wert angehängt. (Die hier tatsächlich verwendete Zeichenkette sind die ersten 12 Zeichen, die durch die Ausführung des Befehls erzeugt werden:

```
$ git rev-parse --verify HEAD
```

der innerhalb des Skripts „scripts/setlocalversion" ausgeführt wird.)

1.5 Build ID Salt

CONFIG_BUILD_SALT [=] [ ]
Type: string
Dies wird verwendet, um die Binaries und ihre Debug-Infos zu verknüpfen. Wenn diese Option gesetzt ist, dann wird dieser Wert in die Berechnung der Build-ID einbezogen. Wird von Distributionen verwendet, die sicherstellen wollen, dass es eineindeutige IDs zwischen verschiedenen Builds gibt. Üblicherweise brauchen wir das nicht.

1.6 Kernel compression mode

Der Linux-Kernel ist eine Art selbstextrahierende ausführbare Datei.
Es stehen mehrere Kompressionsalgorithmen zur Verfügung, die sich in Effizienz, Kompressions- und Dekompressionsgeschwindigkeit unterscheiden. Die Komprimierungsgeschwindigkeit ist nur bei der Erstellung eines Kernels relevant. Die Dekomprimierungsgeschwindigkeit ist bei jedem Systemstart von Bedeutung. (Eine ältere Version dieser Funktionalität (nur bzip2) für 2.4 wurde von Christian Ludwig bereitgestellt) Hohe Komprimierungsoptionen sind vor allem für Benutzer nützlich, die wenig Festplattenplatz zur Verfügung haben (embedded systems), für die aber die Ram-Größe weniger wichtig ist.
Überblick: Gzip werden von den älteren Kernelversionen unterstützt,
Arch Linux (since Linux/x86 5.9.0) Standard: ZSTD (former: XZ since 4.14.4, predecessor GZIP,XZ)
Debian 11.6: XZ
@TODO Weitere Linux Distros

1.6.1 Gzip

CONFIG_KERNEL_GZIP [=n] [ ]
Die alte und bewährte gzip-Kompression. Sie bietet ein gutes Gleichgewicht zwischen Kompressionsrate und Dekompressionsgeschwindigkeit.

1.6.2 Bzip2

CONFIG_KERNEL_BZIP2 [=n] [ ]
Die Kompressionsrate und auch die Geschwindigkeit der ist durchschnittlich. Die Ge-

schwindigkeit der Dekomprimierung ist die langsamste. Größe des Kernels ist etwa 10 % kleiner im Vergleich zu GZIP. Es benötigt auch einen großen Speicherbereich, bei modernen Kerneln benötigt man zumindest 8 MB RAM oder mehr beim Booten.

1.6.3 LZMA

CONFIG_KERNEL_LZMA [=n] [ ]
Dieser Kompressionsalgorthmus hat die höchste Komprimierung. Die Geschwindigkeit der Dekomprimierung liegt zwischen GZIP und BZIP2. Komprimierung ist die langsamste. Kernelgröße beträgt etwa 33 % weniger als mit GZIP.

1.6.4 XZ

CONFIG_KERNEL_XZ [=n] [ ]
XZ verwendet den LZMA2-Algorithmus und befehlssatzspezifische BCJ-Filter, die das Komprimierungsverhältnis des ausführbaren Codes verbessern können. Die Größe des Kernels ist mit XZ im Vergleich zu GZIP etwa 30 % kleiner. Auf Architekturen, für die es einen BCJ-Filter gibt (i386, x86_64, ARM, IA-64, PowerPC und SPARC), erzeugt XZ einen um einige Prozent kleineren Kernel als einfaches LZMA. Die Geschwindigkeit ist in etwa die gleiche wie bei LZMA: Die Dekomprimierungsgeschwindigkeit von XZ ist besser als die von bzip2, aber schlechter als die von gzip und LZO. Die Komprimierung ist langsam.

1.6.5 LZO

CONFIG_KERNEL_LZO [=n] [ ]
Kompressionsrate ist die schlechteste aller anderen. Kernelgröße ist etwa 10 % größer als GZIP. Jedoch ist die Geschwindigkeit beim Komprimieren und Dekomrimieren die höchste.

1.6.6 LZ4

CONFIG_KERNEL_LZ4 [=n] [ ]
LZ4 ist eine LZ77-Typ-Komprimierung mit einer festen, byte-orientierten Enkodierung. Siehe auch http://code.google.com/p/lz4. Komprimierungsverhältnis ist noch schlechter als LZO. 8 % größere Kernelgröße als bei LZO. Dekomprimierung ist jedoch von der Geschwindigkeit her schneller als LZO.

1.6.7 ZSTD

CONFIG_KERNEL_ZSTD [=y] [**Y**]
ZSTD ist ein Komprimierungsalgorithmus, der auf eine Zwischenkomprimierung mit schneller Dekomprimierungsgeschwindigkeit abzielt. Er komprimiert besser als GZIP und dekomprimiert etwa so schnell wie LZO, ist aber langsamer als LZ4. Sie benötigen mindestens 192 KB RAM oder mehr zum Booten. Das Kommandozeilenprogramm `zstd` ist für die Komprimierung erforderlich.

1.7 Default init path

CONFIG_DEFAULT_INIT [=] [ ]
Diese Option legt den Standard-Init-Pfad für das System fest, wenn in der Kernel-Befehlszeile keine solche init=-Option übergeben wird. Wenn der angeforderte Pfad nicht vorhanden ist, wird trotzdem versucht, weitere Orte zu finden (z. B. /sbin/init usw.). Wenn dieser Pfad leer ist, wird einfach die Fallback-Liste verwendet, wenn init= nicht übergeben wird.

1.8 Default hostname

CONFIG_DEFAULT_HOSTNAME [=archlinux] [**=archlinux**]
Diese Option legt den Standard-Hostnamen des Systems fest, noch bevor der Userspace das Kommando sethostname(2) aufruft. Der Kernel verwendet hier traditionell "(none)", Sie möchten vielleicht eine andere Voreinstellung verwenden, um ein minimales System mit weniger Konfiguration benutzbar zu machen.

1.9 System V IPC

CONFIG_SYSVIPC [=y] [**Y**]
Die Inter-Prozess-Kommunikation IPC ist eine Zusammenstellung aus Bibliotheksfunktionen (libraries) und Systemaufrufen die Prozesse (laufende Programme) synchronisiert und Daten untereinander austauschen kann. Generell ist das eine gute Sache, einige Programme würden auch nicht funktionieren wenn Sie hier kein Y (ja) setzen.

1.10 POSIX Message Queues

CONFIG_POSIX_MQUEUE [=y] [**Y**]
Die POSIX-Variante der Nachrichtenwarteschlangen (message queues) ist ein Teil der IPC. In POSIX-Nachrichtenwarteschlangen hat jede Nachricht eine Priorität, die über die Reihenfolge des Empfangs durch einen Prozess entscheidet. Wenn Sie Programme kompilieren und ausführen wollen, die z.B. für Solaris geschrieben wurden und die POSIX-Warteschlangen (Funktionen mq_) verwenden, sagen Sie hier Y. POSIX-Nachrichtenwarteschlangen sind via Dateisystem als „mqueue“ sichtbar und können irgendwo eingehängt werden, wenn Sie Dateisystemoperationen auf Nachrichtenwarteschlangen durchführen wollen.

1.11 General notification queue

CONFIG_WATCH_QUEUE [=y] [**Y**]
Dies ist eine allgemeine Benachrichtigungswarteschlange für den Kernel, um Ereignisse an den Userspace weiterzuleiten, indem sie in Pipes gesplittet werden. Sie kann in Verbindung mit Watches für Schlüssel-/Schlüsseländerungsbenachrichtigungen (key/keyring) und Gerätebenachrichtigungen verwendet werden.
Bemerkung: Bei Debian Bullseye ist dies nicht gesetzt (N).

1.12 Enable process_vm_readv/writev syscalls

CONFIG_CROSS_MEMORY_ATTACH [=y] [**Y**]
Die Aktivierung dieser Option fügt die Systemaufrufe process_vm_readv und process_vm_writev

hinzu, die es einem Prozess mit den richtigen Rechten ermöglichen, direkt aus dem Adressraum eines anderen Prozesses zu lesen oder in diesen zu schreiben. Weitere Einzelheiten finden Sie in der Manpage.

1.13 uselib syscall (for libc5 and earlier)

CONFIG_USELIB [=n] **[N]**
Diese Option schaltet den uselib-Systemaufruf ein, der im dynamic-Linker von libc5 und früher verwendet wird. Das aktuelle glibc verwendet diesen Systemaufruf nicht mehr, deshalb kann man diese option ausschalten wenn sie keine Programme mehr verwenden, die auf libc5 (oder früher) compiliert wurden.
Bemerkung: Debian Bullseye verwendet dies noch (Y).

1.14 Auditing support

CONFIG_AUDIT [=y] **[Y]**
Aktivieren Sie eine Überwachungsinfrastruktur, die mit einem anderen Kernel-Subsystem verwendet werden kann, wie z.B. SELinux (das dies für die Protokollierung der Ausgabe von avc-Nachrichten benötigt). Die Systemaufrufüberprüfung ist auf Architekturen, die sie unterstützen, enthalten.