Dynamische Bibliotheken in der Bash

Wunschbrunnen

Jan Bundesmann

Das Shell-Plug-in Ctypes.sh erlaubt den Zugriff auf beliebige Funktionen in dynamischen Bibliotheken von der Bash aus. Nutzer können damit in ihren Skripten beispielsweise auf C-Funktionen zugreifen.

Wer viel in der Linux-Konsole arbeitet, greift häufig zu Shell-Skripten. Sie bieten sich an, wenn komplexere Arbeitsschritte automatisiert stattfinden sollen. Die eingebauten Funktionen unterscheiden sich von Shell zu Shell. Bestimmte Aufgaben lassen sich jedoch mit keiner Variante erfüllen.

Die Bash hantiert zuverlässig mit Strings und mit Ganzzahlen von 0 bis 255. Das Bash-Plug-in Ctypes.sh spricht alle Nutzer an, die sich jenseits dieser Datentypen bewegen – vor allem solche, die mit Fließkommazahlen in der Bash arbeiten wollen oder komplexere mathematische Funktionen nutzen. Die Entwickler veröffentlichten die Version 0.1 am 24. August unter der MIT-Lizenz.

Ctypes.sh erweitert die Fähigkeiten der Bash, indem es Zugriff auf die Funktionen in dynamischen Bibliotheken erlaubt. Dafür stellt es die C-Schnittstellen *dlopen*, *dlsym* und *dlclose* in der Bash bereit und eine spezielle Routine *dlcall*, die den Funktionsaufruf durchführt. Neben den zusätzlichen Datentypen liefert Ctypes.sh die Zugriffsmöglichkeiten von C auf Sockets und das Dateisystem.

„Hello, World!“ über Sprachgrenzen hinweg

Das Plug-in steht als Tarball zum Download zur Verfügung (siehe „Alle Links“). Die Installation läuft über den klassischen Linux-Dreisprung *configure && make && make install*. Im Verzeichnis *PREFIX/bin* landet das Skript *ctypes.sh*, das die Erweiterung Ctypes.sh aktiviert. Damit ist der Weg geebnet für das erste Bash-Skript mit der neuen Funktion:

```
#!/bin/bash
source path/to/ctypes.sh
dlcall puts "Hello, World!"
```

Dynamische Bibliotheken laden Shell-Programmierer über den Befehl *dlopen libXYZ.so*. Dadurch erhalten sie Zugriff auf alle Symbole, die die Bibliothek XYZ enthält. Das Kommando *nm libXYZ.so* liefert deren Namen. Kompiliert man C-Code mit dem GCC, unterscheidet sich der Name nicht von dem im Quelltext. Andere Sprachen und Compiler dekorieren die Funktionen. Das sogenannte „Name Mangling“ trägt Eigenheiten der jeweiligen Sprache Rechnung: Module, Namespaces, Überladen.

Funktionen statt Operatoren

Folgender Quelltext in C dient als Schnittstelle zur Multiplikation von Fließkommazahlen, da Nutzer von Ctypes.sh Operatoren nicht direkt ansprechen können:

```
double mult(double a, double b)
{
    return (a * b);
}
```

Unter Verwendung der GCC-Compiler-Flags *–shared –fPIC* lässt sich dieser Codeschnipsel zu einer dynamischen Bibliothek kompilieren. Heißt die resultierende Datei *libmult.so*, kann man nun in der Shell eine Fließkomma-Multiplikation durchführen:

```
dlopen ./libmult.so
dlcall -r double mult double:2.5 double:5
```

Als Ausgabe erhält man *double:12.5*. Die Mächtigkeit dieses Werkzeugs offenbart sich bei der Verwendung in Skripten. Über die Option *–n FOO* schreibt *dlcall* den Rückgabewert in die Variable *FOO*. Vorsicht an dieser Stelle: Ist der Rückgabetyp der Funktion über die Option *–r* nicht korrekt definiert, kann die Bash abstürzen. Ähnlich verhält es sich, wenn die Typen der Parameter nicht korrekt angegeben wurden. Programmierer müssen alle Datentypen explizit ausweisen, nur Ganzzahlen und Strings erkennt Ctypes.sh implizit. Obacht auch bei Systemen, deren Standorteinstellung das Komma zur Dezimaltrennung vorsieht. In dem Fall schneidet Ctypes.sh die Information hinter dem Punkt einfach ab.

Da trotzdem in Bash-Skripten nur die wenigen nativen Datentypen zur Verfügung stehen, bedarf es des Umwegs über Zeiger, um Arrays oder Datenstrukturen zwischen Skript und externer Funktion über *dlcall* hin und her zu schieben. Zugriff auf den Speicher geben die Funktionen *pack* und *unpack*. Den Inhalt des Bash-Arrays *values* schreibt *pack $pointer values* in den Speicherbereich, den *$pointer* definiert. Der gegenteilige Aufruf *unpack $pointer values* schreibt den Inhalt von *$pointer* in die Variable *values*. Programmierer müssen auf beiden Seiten – Shell-Skript und C-Quelltext – definieren, wie sie den Speicherbereich bestücken. Schreibt die C-Routine in den Speicher den Wert *112*, gibt

```
values=(int16)
unpack $buffer values
echo ${values}
```

wie gewünscht *int:112* aus, während

```
values=(char)
unpack $buffer values
echo ${values}
```

statt der Zahl das Zeichen 112 aus der Zeichentabelle, für ASCII *char:p*, ausgibt.

In der Dokumentation des Projekts stellen die Entwickler einige Anwendungsfälle vor, in denen sie komplexere Funktionen verwenden. Sie demonstrieren beispielsweise den direkten Zugriff auf die Grafikbibliothek GTK+. Zur Demonstration, wie Ctypes.sh die Zugriffe auf Sockets und die Dateisystemschnittstelle von C verwendet, haben sie einen einfachen HTTP-Server ausschließlich aus Bash-Skripten geschrieben. Das etwa 20 KByte kleine Programm beherrscht zwar lediglich Grundfunktionen, nutzt aber auch nur native C-Funktionen. Vom produktiven Gebrauch raten die Macher allerdings ab. (jab)

Alle Links: www.ix.de/ix1512127