Libfind – neue Paradigmen für UNIX Software

Jörg Schilling

berliOS

Fraunhofer FOKUS Institut für Offene Kommunikationssysteme

libfind

UNIX Systeme der 1970er Jahre

- Erstes UNIX ~ 1970 auf PDP-7
- UNIX V6 (1976) auf PDP-11 Adressraum 2x64 kB
- Kein „virtueller“ Speicher
- Programme können nur als Ganzes in den Swap
  - Adressraum bestimmt Programmgröße
  - Große Programme mit Fehlermeldungen unmöglich
  - Lösung durch Douglas McIlroy Pipes (1973)
    - Kombination kleiner Programme out1 -> in2
    - Steve Bourne: einen Tag nach der Erfindung der Pipes konnten wir uns nicht mehr vorstellen wie es ohne war

libfind

Wenig Speicher und Pipes bestimmen Regeln

- Regeln aus der Pipe Philosophie
  - Schreibe kleine Programme mit nur einer Aufgabe
  - Schreibe Programme mit Text Ein- und Ausgabe
  - Schreibe Programme die gut zusammenarbeiten
- Fazit:
  - Mach nur eine Sache aber die gut
- Extreme Beispiele: SCCS (1972)
  - „get s.file“ --> s.file nonexistant (ut4)
  - „help ut4“ --> The file and/or a directory in the pathname does not exist. Check for typos.

libfind

Stimmen die Regeln der 1970er Jahre noch?

- 1975: 2x64 kB Adressraum
- 2007: Hunderte von MB sind kein Problem
- Können alle Probleme durch die Kombination kleiner Programme gelöst werden?
- Können alle Probleme durch die Kombination kleiner Programme sinnvoll gelöst werden?
- Geben uns heutige Betriebssysteme vielleicht bessere Hilfsmittel für bessere Lösungen?

Untersuchungen am praktischen Beispiel

- TAR ist ein Dateiarchivierungprogramm
  - Es archiviert mehrere Dateien in eine neue Datei
  - Es extrahiert eine oder mehrere Dateien aus einem Archiv
  - Es listet den Inhalt eines Archivs auf
- Möglichkeiten der traditionellen Pipe Philosophie:
  - Erstellen einer Vorauswahl der zu archivierenden Dateien mit Hilfe von find(1) und anschließende Archivierung
  - Extrahieren des Archivs und anschließende Modifikation von Dateirechten und Dateieigentümern mit Hilfe von find(1)
  - Auflisten einer Auswahl des Inhaltes des Archivs mit Hilfe von grep(1)

Was die traditionelle Philosohie nicht kann

- Auflisten aller Dateien des Archivs die z.B. älter als 3 Tage sind ohne das Archiv dazu auszupacken
- Extrahieren nur der Dateien des Archivs, die einem bestimmten Benutzer gehören
- Regelbasiertes Archivieren von Dateien mit anderen Zugriffsrechten oder Eigentümern als in der Originaldatei
- Extrahieren von Dateien und Modifizieren der Rechte bzw. Eigentümer in einem Schritt

libfind

Schlüsse aus der historischen Methode

- Die historische Methode kombiniert bestehende Programme und ermöglicht so eine „Wiederverwendung“der Funktionen dieser Programme in einem größeren Projekt
- Wenn mehr als ein einfacher Datenfluß a -> b benötigt wird dann versagt die Pipe Methode
- Fazit: Die historische Methode ist beschränkt

Heutige Randbedingungen

- „Hoher“ Speicherbedarf ist kein Problem
- Alle modernen Plattformen bieten dynamische (shared) Bibliotheken
- Fast alle modernen Plattformen bieten zur Laufzeit ladbare Bibliotheken
- fork()/exec() ist relativ aufwendig
  - Aufruf von „Unterprogramm“-Prozessen ist teuer
- ARG_MAX ist zwar größer als früher aber nicht unendlich
  - `find . -name '*.[hc]' -exec wc {} +` nicht immer erfolgreich
  - Solaris 32 Bit: 1MB, 64 Bit: 2MB

libfind

Die Lösung: shared libraries intelligent nutzen

- Alt: kleine Programme sind kombinierbar und wiederverwendbar
- Neu: kleine shared libraries sind kombinierbar und wiederverwendbar
- Alt: Pipes ermöglichen Datenfluß in eine Richtung
- Neu: shared libraries ermöglichen komplexen Datenfluß
- Alt: Laden von Programmen ist „teuer“
- Neu: Laden von shared libraries ist weniger aufwendig
- Alt: Die Anzahl der Parameter bei exec() is begrenzt
- Neu: Datenmenge bei Parameterübergabe mit shared lib nur durch Speicher begrenzt

libfind

Existierende Implementierungen auf lib-Basis

- David Korn und Glenn Fowler
  - libcmd von ksh93(1) enthält eine rentrante Implementierung der wichtigsten UNIX Kommandos
    - basename, cat, chgrp, chmod, chown, cmp, comm, cp, cut, date, dirname, expr, fmt, fold, getconf, head, id, join, ln, logname, mkdir, mkfifo, mv, paste, rm, rmdir, stty, tail, tee, tty, uname, uniq, wc
    - Aufruf z.B. `b_cat(int argc, char **argv, void *context);`
  - Ksh93 ermöglicht das Laden von shared libs zur Laufzeit
- Jörg Schilling
  - libfind wird zur Zeit durch star(1) und mkisofs(1) genutzt
  - libfind ist eine vollständige reentrante find(1) Implementierung

Vergleich libcmd libfind

- libcmd hat einfache separat aufzurufende Kommandos
  - ksh93 ruft die Kommandofunktionen mit argc, argv
  - Jede Kommandofunktion hat ihren eigenen unabhängigen Optionsparser
- libfind wird typischerweise in Programme eingebettet
  - sfind(1) ist ein 10 Zeilen Programm um libfind
  - star/mkisofs find(1) Kommandosyntax wird eingebettet
  - SCCS (geplant) libfind wird „unerkannt“ genutzt
- libfind könnte auch durch ksh93 genutzt werden und dann statt fork()/exec() direkt wieder Kommandofunktionen aus libcmd rufen

libfind

Einbindung von libfind in star und mkisofs

- Im „Create-“ Modus ruft libfind eine Callbackfunktion von star zur Archivierung der Datei auf
- Im „List-“ und „Extrakt-“ Modus ruft star den Expression-Interpreter von libfind für jede Datei auf
- In beiden Fällen wird die Datei übersprungen wenn der libfind Interpreter FALSE liefert
- In beiden Fällen kann libfind den „struct stat“ modifizieren
- Nach der Option -find erfolgt die Übergabe an den Kommandozeilenparser von libfind

libfind

Sind shared libs die Lösung für alles?

- Zuviel Funktionen -> zuviele Optionen
  - Kleiner ist immer noch besser
- Kombination von vielen Funktionen nur dann wenn die notwendigen Optionen überschaubar bleiben
  - Schlecht: unübliche neue Optionen geringer Mächtigkeit z.B. GNU tar
  - Gut: zusätzlich die mächtige und bekannte find(1) Kommandozeilensyntax z.B. star

Regeln für für Verwendung von shared libs

- Nutzung von Funktionen aus shared librares nur wenn es sinnvoll ist:
  - Komplexer Datenfluß ist nicht durch Pipes zu erreichen
  - Grenzen durch fork()/exec() ARG_MAX können vermieden werden
  - Performance Gewinn ist erreichbar
  - Gewünschte Funktion ist in Reentranter lib verfügbar
- Nicht alle alten Regeln verwerfen sondern ergänzen
- Selbst sinnvolle Funktionen als shared lib implementieren

Aufwand und aktueller Zustand

- Das sfind(1) Programm wurde in einer Woche „von Scratch“ im Juli 2004 implementiert
- Seit Oktober 2005 entsteht daraus libfind
- Erste nutzbare Kombination sfind/star nach 1 Mannmonat
- Bislang mehr als 3 Mannmonate Aufwand bei der Konvertierung zu einer „sauberen“ Bibliothek
- libfind ist seit Ende 2006 ausreichend entwickelt
- Vollständige Implementierung im März 2007

libfind

Konvertierung Programm -> Biliothek

- Keine globalen modifizierbaren Daten sondern Funktions-parameter
- Kein exit() sondern return(error_code)
- Keine Speicherlecks, auch bei Fehlern
- free() aller allozierter Daten
- Ausgaben zu Fehlern vermeiden, stattdessen Fehlercodes
- Ausgaben sollen umlenkbar sein (z.B. FILE *std[3])
- Nachdenken über Interrupts vom Shell (Variable prüfen)

Der sfind(1) Sourcecode

```
#include <stdio.h>

main(int argc, char **av)
{
   FILE *std[3];

   std[0] = stdin;
   std[1] = stdout;
   std[2] = stderr;
   return (bfind(argc, argv, std));
}
```