

# Seminar Dateisysteme – Large File System

Timo Fischer

1

## Inhalt

- Einführung
- Der Large File Support (LFS)
- Die Benutzung des LFS

Timo Fischer

2

Der Vortrag über Das Large File System soll vor allem Informationen geben um zu verstehen warum die Umstellung nötig war. Des Weiteren sollte man die Änderungen kennen lernen und den Umgang mit diesen Änderungen in den Unterschiedlichen Kompilierungsumgebungen sehen.

## Einführung

- Dateigrößen in 32bit Systemen
- Dateigrößen in 64bit Systemen
- Motivation

Timo Fischer

3

## Dateigrößen in 32bit Systemen

- Ursprünglich:
  - Festlegung der Dateigröße und Position durch 32bit integer.
    - Dadurch automatisch maximale Dateigröße durch  $2^{32}$  bit festgelegt (~4GB)
  - Problem durch benutzen des Integers als signed
    - Senkt maximale Dateigröße auf  $2^{31}$  bit (~2GB)
- → Definition: Large File
  - Ein „Large File“ ist ein File welches größer ist als 2GB und somit von normalen 32bit Systemen nicht behandelbar ist.

Timo Fischer

4

In der Einführung wird die allgemeine Seite der bisherigen Dateigrößen behandelt. Wie sahen diese in 32bit Rechnern aus, wo lagen die Einschränkungen, und warum sollte man das ganze überhaupt erweitern wenn doch die Möglichkeit bestand auf 64bit umzusteigen.

Nachtrag:

Die Beschränkung auf 31bit, also auf „signed“, erfolgte wohl aus dem Grund, dass die negativen Zahlen die dann existieren als Rückgabewerte bei Fehlern benutzt werden konnten. Man hatte somit zwar in der Gesamtgröße der Datei Einschränkungen hinzunehmen, konnte allerdings mit diesen Dateien dann sicherer Umgehen und spezielle Fehlercodes definieren und abfangen.

## Dateigrößen in 64bit Systemen

- Dateien haben gleiche Gestalt wie in 32bit Systemen
- Durch 64bit Technologie allerdings ein größerer Bereich adressierbar
  - Maximale Dateigröße unsigned:  $2^{64}$  bit
  - Maximale Dateigröße signed:  $2^{63}$  bit
- Hier also keine Notwendigkeit an der Dateigröße etwas zu ändern (noch nicht ☺)

## Motivation I

- Zu Beginn genügen 32bit Adressierungen
- Festplatten und Speichermedien hatten nur begrenzten Speicherplatz
  - Notwendigkeit für Große Dateien war nicht gegeben
- Im Laufe der Zeit kamen höhere Anforderungen an die Dateigrößen
  - Multimedia
  - Datenbank

## Motivation II

- **Komplette Umstellung auf 64bit Architektur wäre für die Industrie zu aufwendig**
  - Keine Rückwärtskompatibilität
  - Doppelte Belastung an Anwendungen
    - 2 Versionen ( 32bit und 64bit )
  - 32bit Architektur günstiger in der Herstellung
- **Lösung wurde gesucht um große Dateien benutzen zu können ohne gezwungenermaßen auf 64bit umstellen zu müssen**

Timo Fischer

7

Die Rückwärtskompatibilität war die Hauptschwierigkeit. Man wird später sehen dass diese auch nur teilweise gelöst wurde und meistens sehr umständlich, da man Inkompatibilitäten umgehen musste und mehr oder weniger Umwege in Kauf nehmen musste, was die Laufzeit der Programme teilweise stark beeinträchtigte.

- **Problem:**
  - Eine Möglichkeit um auch auf 32bit Dateisystemen mit Large Files umgehen zu können
- **Lösung:**
  - LARGE FILE SUPPORT (LFS)

Timo Fischer

8

## Der Large File Support

- Allgemeines
- Änderungen
- Resultate

Timo Fischer

9

Dieses Kapitel beschäftigt sich mit den allgemeinen Voraussetzungen die zum Large File Support geführt haben. Es betrachtet die wichtigsten Änderungen die vorgenommen worden bzw zur Verfügung gestellt worden sind. Die Resultate sind hierbei erst einmal nur die unterschiedlichen Kompilierumgebungen die für den Benutzer ab dann zur Verfügung standen.

## Der Large File Support (LFS)

- 1996 Bildung einer industriellen Initiative
  - Das *Large File Summit* (ebenfalls LFS) entsteht
- Ziel
  - *Einheitliche* Methode um Dateigrößen durch 64bit Zahlen darzustellen
- Problem
  - Keine 100%ige Rückwärtskompatibilität möglich
  - Manche Dateisysteme konnten nicht erweitert werden.  
→ Beispiel Microsoft Windows' FAT32 unterstützt maximal 4GB – Dateien (LFS erst ab NTFS)

Timo Fischer

10

Wichtig war dass die Methode die gesucht wurde einheitlich, also standardisiert war. Man musste vermeiden dass selbst nachdem man den LFS entwickelt hat doch jeder Nutzer, bzw jede Firma, wieder ihre eigene Methode entwickeln musste. Man musste also sehr beschränkt und allgemein bleiben bei den angestrebten Änderungen um sie mit so vielen Systemen wie nur möglich kompatibel zu halten. Einige Ausnahmen blieben allerdings nicht aus, denn im Gegensatz zu Unix war Windows etwas störrisch ☺

## Der Large File Support (LFS) II

### ■ Weitere Probleme:

- Kompatibilität alter Anwendungen
  - Manche Funktionen (Bsp. Betriebssystem) mussten alte 32bit – Dateigrößen beibehalten
  - Für Large Files mussten spezielle Funktionen entworfen werden
- Bei 32bit ist ein „long“ typischerweise 32bit groß
  - Man benötigt also einen neuen Datentyp für 64bit Länge

Timo Fischer

11

Auch wenn es Ausnahmen gibt in denen ein „long“ als 32+bit definiert ist, war die Notwendigkeit gegeben einen neuen Datentypen zu entwerfen bzw zu definieren.

## Änderungen durch LFS

### ■ Normale Interfaces wurden angepasst

- Fehlermeldungen werden richtig interpretiert. *Open()* wird nun *errno* auf *EOverflow* setzen bei Überschreiten des Offsets einer regulären Datei.
- 64bit Funktionen und Datentypen wurden eingeführt um große und normale Dateien zu behandeln
  - Für jede Funktion *xxx()* die auf dem Offset einer Datei arbeitet gibt es *xxx64()*
  - Für jeden Datentypen *xxx\_t* der mit Datei Offsets zu tun hat gibt es *xxx64\_t*

Timo Fischer

12

Durch die Änderung gaben die verschiedenen Operationen die mit Dateien zu tun hatten jetzt Fehlerwerte zurück. Setzen das „*errno*“-Flag auf den entsprechenden Fehlerwert und halfen somit eine Möglichkeit zu schaffen diese Fehler, welche allesamt aus Datei-Überlängen resultieren, abzufangen und entsprechend zu behandeln.

Die wichtigsten Fehler hierbei sind *EOverflow* und *EFBIG*.

## Geänderte Datentypen

### ■ Folgende Datentypen wurden auf 64bit Versionen erweitert:

- `ino_t` Serial Number
- `off_t` Offsets und Dateigröße
- `fpos_t` Position innerhalb der Datei
- `rlim_t` Ressourcen Limit
- `blkcnt_t` Anzahl der Blöcke
- `fsblkcnt_t` file system block counts
- `fsfilcnt_t` file system inode counts

## Resultate

### ■ Unterschiedliche Arten von Kompilierungs-Umgebungen

- Standard
  - Alle `xxx()` benutzen die ursprüngliche Implementierung
  - Small Files werden normal behandelt
  - Zugriff auf Large File resultiert in Fehler
- Gemischt
  - Sowohl `xxx()` als auch `xxx64()` sind verfügbar
- „Large File“ Umgebung
  - `xxx()` und `xxx_t` sind mit `xxx64()` bzw `xxx64_t` verknüpft

Die Standard-Umgebung ist in dem Sinne eigentlich nicht neu entstanden. Es ist eigentlich die normale Umgebung welche keine Änderung zu der Umgebung darstellt in der man sich ohne LFS befindet.

Die Verknüpfung in der Large File Umgebung ist keine direkte Verknüpfung oder Verlinkung. Im Endeffekt dienen hierbei die Namen der `xxx()` dem kompilier lediglich als macro. Er ersetzt also beim kompilieren alle stellen in denen `xxx()` steht durch `xxx64()`.

## Die Benutzung des LFS

- Fehlerbehandlung
- Interface Parameter und Rückgabewerte (Large File Umgebung)
- Gemischte Umgebung

Timo Fischer

15

In diesem Teil wird auf die Benutzung der Änderungen eingegangen. Worauf muss der Programmierer achten wenn er Anwendungen schreiben möchte die mit großen Dateien umgehen sollen.

Der Teil Interface Parameter und Rückgabewerte bezieht sich auf die Programmierung in der large file Umgebung. Es werden also nur xxx() benutzt. Diese stehen hier ja praktisch stellvertretend für ihre 64bit Versionen.

## Fehlerbehandlung

- Um Fehler während eines Programms zu vermeiden sollte man sich an folgende Richtlinien halten:
  - Alle Stellen finden bei denen mit Dateien gearbeitet wird
  - EOVERFLOW oder EFBIG Fehler entsprechend abfangen
  - Programmabsturz vermeiden oder abfangen

Timo Fischer

16

Unter Programmabsturz vermeiden oder abfangen versteht man, dass man eben falls ein Problem auftritt gezielt einen Absturz herbeiführen kann um zu vermeiden dass Datenverlust eintritt. Wenn zum Beispiel ein Programm eine Datei kopiert und diese Datei ist >2GB dann sollte das Programm zumindest abstürzen wenn die Datei nicht kopiert werden kann ab einer gewissen Stelle. Würde es das nicht tun würde das Programm die Datei kopieren aber ab einer Größe von 2GB wären keine Daten mehr geschrieben worden. Es kam also zu Datenverlust.

## Beispiel einer Änderung

### ■ Methode lseek()

```
if (lseek(fd, offset, 0) {  
    /* ohne Fehlerbehandlung für große Dateien */  
}
```

Sollte geändert werden in

```
if (lseek(fd, offset, 0) {  
    if (errno == EOVERFLOW) {  
        /* Fehlerbehandlung für große Dateien */ }  
}
```

Timo Fischer

17

Mit dieser Methode kann man Anwendungen large file save machen.

Large File Save bedeutet dass die Anwendung mit großen Dateien umgehen kann und nicht abstürzt falls es auf eine solche Datei trifft. I.A. gilt dass Anwendungen welche large file save sind zwar Dateien lesen können die größer sind als 2GB und auch mit diesen umgehen können. Als Output file wird aber normalerweise noch ein small file erstellt.

Wenn ein Programm auf den Input einer großen Datei auch eine große Datei als Output liefert (z.B. beim kopieren) so nennt man die Anwendung large file aware.

## Interface Parameter und Rückgabewerte I

### ■ Vorsicht bei der Verwaltung von Übergabe und Rückgabewerten

- Möglicherweise Fehler bei Interpretation dieser Werte durch den Compiler

```
long curpos;
```

```
curpos = lseek(fd, 0L, SEEK_CUR);
```

würde in Large File Umgebung den 64bit-Rückgabewert in einen 32bit-Wert wandeln ( → evtl Datenverlust)

besser:

```
off_t curpos;
```

```
curpos = lseek(fd, (off_t)0, SEEK_CUR);
```

Timo Fischer

18

## Interface Parameter und Rückgabewerte II

- nicht nur direkt betroffene Werte sollten entsprechend behandelt werden:

```
int delta;  
long curpos;  
...  
curpos = lseek(fd, 0L, SEEK_CUR);  
curpos += delta;
```

besser:  
`off_t delta;`  
`off_t curpos;`  
...

Timo Fischer

19

## Interface Parameter und Rückgabewerte III

- Weitere notwendige Änderungen:

- Ausgabe- und in-memory – Zeichenketten
  - Bsp.  
`off_t offset;`  
`printf(„%7ld“, offset);`

sollte geändert werden in  
`printf(„%7lld“, offset);`

um explizit einen „long long“ Wert anzugeben

Timo Fischer

20

Man sieht hierbei, dass es am einfachsten ist Variablen, die in irgendeiner Weise miteinander zu tun haben, von Vorneherein mit den neuen Datentypen zu initialisieren. Casts im Nachhinein gehen zwar auch, man läuft dabei aber eher Gefahr einzelne Casts zu vergessen und damit nicht 100% sicherzustellen dass kein Datenverlust eintritt.

Diese Änderung dient rein im Umgang mit einer großen Datei und sollte dementsprechend auch nur vorgenommen werden wenn man sicher ist dass es sich um eine solche handelt.

Auf der nächsten Folie sieht man eine Methode um die Änderung flexibel auf die Dateigröße bzw die Kompilierungsumgebung zu reagieren.

## Interface Parameter und Rückgabewerte IV

- Um sowohl small als auch large files zu behandeln sollte man `#ifdef` benutzen

```
off_t offset;  
#if _FILE_OFFSET_BITS == 64  
    printf(„ %7lld“, offset);  
#else  
    printf(„ %7ld“, offset);  
#endif
```

## Gemischte Umgebung

- genaue Planung notwendig auf welche Art von Datei zugegriffen wird

- Bsp.

- `open()` benutzen für Konfigurationsdateien
- `open64()` benutzen für Daten

- Merke:

- falls man mit `open()` auf ein large file zugreifen will wird dies einen Fehler geben
- entsprechende Deklarationen müssen direkt vorgenommen werden
  - `off64_t` muss bei 64bit benutzt werden, `off_t` behandelt hierbei nur 32bit

## Ergebnis

- durch das Large File Summit wurde ermöglicht auch auf 32bit Systemen mit großen Dateien umzugehen
- durch Zugriff auf neue Typen wurde ermöglicht fast jedes Dateisystem large file save zu machen
- durch gezielte Implementierung kann man bei Anwendungen gezielt den Umgang mit kleinen bzw. großen Dateien definieren
- Rückwärtskompatibilität teilweise gegeben
- Standard wurde erreicht dadurch dass man nur Methoden und keine kompletten Anwendungen zur Verfügung stellte

## erreichbare Datei(system)größen

Dateisystem	Max. Dateigröße	Max. Dateisystemgröße
Ext2 oder Ext3 (Blockgröße 1 kB)	$2^{34}$ (16GB)	$2^{41}$ (2TB)
Ext2 oder Ext3 (Blockgröße 2 kB)	$2^{38}$ (256GB)	$2^{43}$ (8TB)
Ext2 oder Ext3 (Blockgröße 4 kB)	$2^{41}$ (2TB)	$2^{44}$ (16TB)
Ext2 oder Ext3 (Blockgröße 8 kB)	$2^{46}$ (64TB)	$2^{45}$ (32TB)
ReiserFS v3	$2^{46}$ (64TB)	$2^{45}$ (32TB)
XFS	$2^{63}$ (8EB)	$2^{63}$ (8EB)
JFS (Blockgröße 512 Byte)	$2^{63}$ (8EB)	$2^{49}$ (512TB)
JFS (Blockgröße 4 kByte)	$2^{63}$ (8EB)	$2^{52}$ (4PB)
NFSv2 (clientseitig)	$2^{31}$ (2GB)	$2^{63}$ (8EB)
NFSv3 (clientseitig)	$2^{63}$ (8EB)	$2^{63}$ (8EB)

Größen in Byte

Quelle: <http://sman.informatik.htw-dresden.de/doc/manual.10.0/manual/sec.filesystems.lfs.html>

### Mysterium:

Bei manchen Beispielen ist die max. Dateigröße größer als die max. Dateisystemgröße. Woran das liegt ist mir ein Rätsel.

## Literatur

- Solaris OS group (March 1996). "*Large Files in Solaris: A White Paper*". Sun Microsystems, Inc..
- (August 14, 1996). "*Adding Large File Support to the Single UNIX® Specification*". X/Open Base Working Group.
- [http://www.llnl.gov/asci/platforms/bluepac/large\\_file\\_prog.html](http://www.llnl.gov/asci/platforms/bluepac/large_file_prog.html)
- [http://de.wikipedia.org/wiki/Large\\_File\\_Support](http://de.wikipedia.org/wiki/Large_File_Support)

Als weiterer link sollte dienen: <http://www.sas.com/standard/large.file/>

In diesem sollte die genauen Beschlüsse des Large File Summit von 1996 niedergeschrieben sein. Von vielen Artikeln über den LFS wird auf diesen Link verwiesen, allerdings kann das Dokument nicht mehr gefunden werden weshalb ich ihn nicht in die Literaturliste aufgenommen habe.

Beim link für Wikipedia ist am besten die englische Seite zu benutzen da der Artikel die Notizen aus dem Deutschen noch etwas ausführlicher darstellt.