Libraries für Parallele E/A

Seminar "Effiziente Programmierung"

Johannes Coym

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2019-12-10

Gliederung

VFS

- Abstraktionsschicht zwischen einer Anwendung und dem Dateisystem
- Gibt Vorgaben für Struktur und Schnittstelle von Dateisystemen
- Unterstützt und verwaltet verschiedenste Dateisysteme
- Bietet ein standardisiertes Interface (POSIX) zu den Anwendungen
  - Interface universell für alle Dateisysteme
  - Anwendungen bleiben portabel

Linux Storage Stack

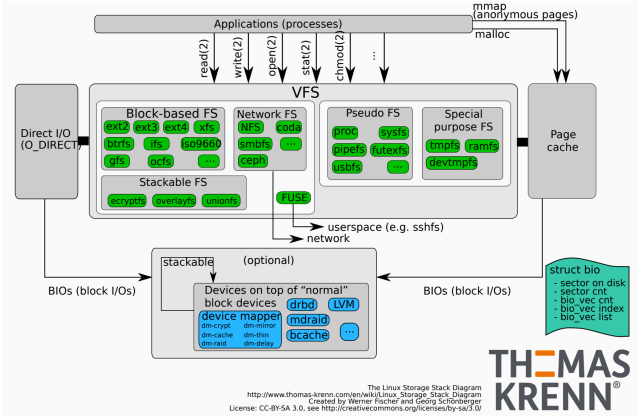

Abbildung: Ausschnitt des Linux Storage Stack [1]

System Calls

- Unter Linux geregelt durch das System Call Interface(SCI)
- Das SCI gibt Calls an das VFS weiter
- Das VFS schaut im Mountpoint das zugehörige Dateisystem nach und gibt diesem den Befehl weiter

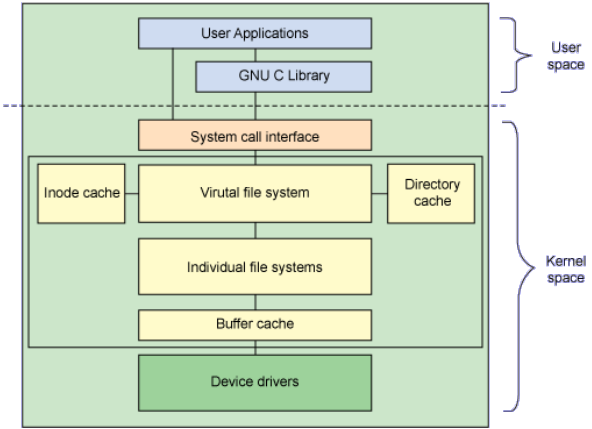

Abbildung: Linux Dateisystem Stack [2]

Portable Operating System Interface

- Standardisierte Schnittstelle für die Verbindung zwischen Anwendungen und Betriebssystemen
- Ab 1985 entwickelt um betriebssystemübergreifende Standards festzulegen
- 1988 das erste Mal als Standard festgelegt für Unix Systeme
- Neueste Revision des Standards von 2017
- Standards für Programme, Services und Tools, sowie z.B. Datei- und Netzwerk-E/A

POSIX Standards bis 1997

- POSIX.1: Core Services
  - Erstellung und Kontrolle von Prozessen, sowie Signale
  - Standard C Library und E/A Funktionen
- POSIX.1b: Real-Time Extensions
  - Nachrichten, Locking, Scheduling, etc.
  - Asynchrone und sychrone E/A
- POSIX.1c: Thread Extensions
  - POSIX Threads (pthreads)
- POSIX.2: Shell and Utilities
  - Command Line, sowie Tools (`echo`, `du`, etc.)

POSIX write

- `write`: Zum seriellen Schreiben
- Bewegt den File Descriptor beim Schreiben mit
- Mehrere Operationen auf die gleiche Datei gleichzeitig würden durch die Bewegung vom File Descriptor durcheinander schreiben

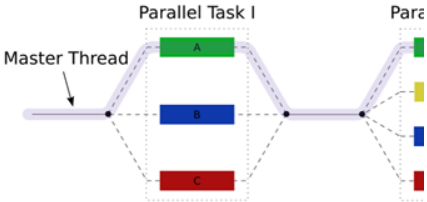

Abbildung: Serielles Schreiben in einem parallelen Programm [3]

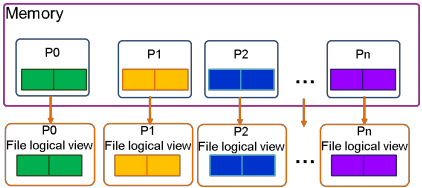

Abbildung: Paralleles Schreiben in mehrere Dateien [4]

POSIX pwrite

- `pwrite`: Schreiben ohne Bewegung vom File Descriptor
- Ein gemeinsamer File Descriptor beim Öffnen der Datei
- Explizite Angabe des Offsets, ausgehend vom File Desriptor
- Mehrere Threads können so an verschiedene Stellen schreiben

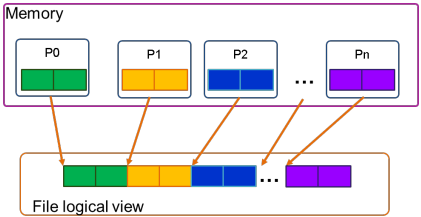

Abbildung: Paralleles Schreiben in eine Datei [4]

POSIX Semantiken

POSIX 2008 Spezifikation für die write Funktion

After a write() to a regular file has successfully returned:
– Any successful read() from each byte position in the file that was modified by that write shall return the data specified by the write() for that position until such byte positions are again modified.
– Any subsequent successful write() to the same byte position in the file shall overwrite that file data.

- Datei muss gelockt sein bis garantiert ist, dass jeder folgende read()-Aufruf richtig liest
- Problematischer für große verteilte Dateisysteme

POSIX Semantiken Beispiel

1. Node0 schreibt in Datei test.txt
2. Node1 liest von test.txt während die Daten noch im Schreibcache von Node0 liegen
3. Node1 liest die Daten die vor Schritt 1 in der Datei waren
4. POSIX Konsistenz wurde verletzt

POSIX Nachteile

- Strenge Konsistenzbedingungen, die oft nicht gewollt sind für HPC Anwendungen
- Bei HPC Performance oft wichtiger als vorgegebene Konsistenz der Daten
- Teilweise schwierigere Handhabung als optimierte High-Level E/A Libs
- Keine Threadsicherheit bei parallelen Zugriffen auf eine Datei

Software Stack für Parallele E/A

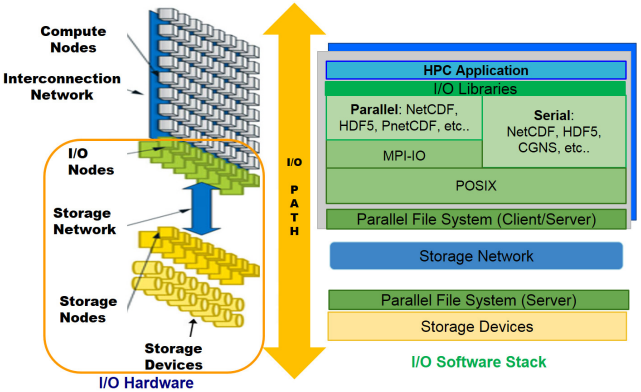

Abbildung: Software Stack für Parallele E/A [4]

MPI-IO

- Bietet ähnliches Interface für E/A wie POSIX
- Ist optimiert für parallele Anwendungen, auch verteilt über mehrere Knoten
- Baut auf dem selben Prinzip wie MPI für mehrere Prozesse auf
- Deutlich relaxiertere Konsistenzanforderungen als POSIX

POSIX vs. MPI-IO Code

```
1 open()
2 write()
3 close()
```

Listing 1: POSIX write dummy

```
1 MPI_File_open()
2 MPI_File_write()
3 MPI_File_close()
```

Listing 2: MPI-IO write dummy

MPI-IO Funktionen

- Open benutzt den Kommunikator wie im normalen MPI
- Write läuft unabhängig auf den einzelnen Knoten
- Close ist wieder gemeinsam über den Kommunikator
- Ebenfalls verfügbar:
  - `MPI_File_write_at`: Bietet Unterstützung für Offsets mit Threadsicherheit
  - `MPI_File_write_shared`: Benutzt gemeinsamen File Handle

MPI-IO Funktionen

- Kollektive E/A führt write parallel auf allen Prozessen aus und sammelt die Daten in größeren Blöcke
- Durch die größeren Blöcke können die Daten schneller geschrieben werden
- Jeder Prozess muss die Funktion selber aufrufen zur Synchronisierung
- Verfügbare Funktionen:
  - `MPI_File_write_all`: Normales synchrones write
  - `MPI_File_write_at_all`: Mit Integrierung des seeks für Threadsicherheit
  - `MPI_File_write_ordered`: Führt die writes nacheinander mit gemeinsamen File Handle aus

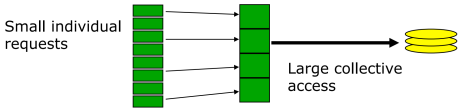

Abbildung: Kollektive E/A mit MPI-IO [5]

MPI-IO Semantiken

- Bei Standardsemantik:
  - Zwei gleichzeitige Writes an die gleiche Stelle geben ein nicht definiertes Ergebnis
  - Writes von anderen Prozessen müssen nicht direkt sichtbar sein
- Atomarer Modus:
  - Deutlich höhere Konsistenzanforderungen
  - Wieder größere Performanceeinbußen

MPI-IO Vor-/Nachteile

- Vorteile:
  - Threadsichere Zugriffe auf Dateien
  - Geringere Konsistenzanforderungen für höhere Performance als POSIX
- Nachteile:
  - Ebenfalls teilweise schwierigere Handhabung als optimierte High-Level E/A Libs

NetCDF

- Ursprünglich als eigenes Datenformat für wissenschaftliche Daten gestartet
- Bietet inzwischen Libraries für eine leichtere Benutzung von HDF5 über das NetCDF-4 Format
  - Klassisches NetCDF Format wird ebenfalls weiterhin unterstützt
- Basiert auf dem CDF Format der NASA von 1985
- Plattformunabhängig und bietet Interfaces für u.a. C, C++, Fortran und Python

HDF5

- Bietet ein standardisiertes Datenformat für Messdaten
- Primär für wissenschaftliche Nutzung gebaut
- Bietet die Möglichkeit Metadaten mit den Messdaten verknüpft zu speichern
- Plattformunabhängig mit Unterstützung für parallele I/O
- Unterstützung für Plugins, die andere Speichermöglichkeiten implementieren können

HDF5 Objekte

- File
  - Beschreibt eine HDF5-Datei
  - Beinhaltet keine eigenen Daten
  - root-Objekt von jeder HDF5-Struktur
- Group
  - Beschreibt eine Gruppe von Objekten
  - Beinhaltet keine eigenen Daten
  - Kann untergeordnet zu einer File oder Group sein

HDF5 Objekte

- Dataset
  - Homogenes, mehrdimensionales Array
  - Kann z.B. Messdaten enthalten
  - Ist immer untergeordnet zu einer Group
- Attribute
  - Kleine Daten, welche an ein Objekt angehängt werden
  - Kann z.B. Metadaten enthalten
  - Kann untergeordnet zu einer Group oder einem Dataset sein

HDF5 Dateistruktur

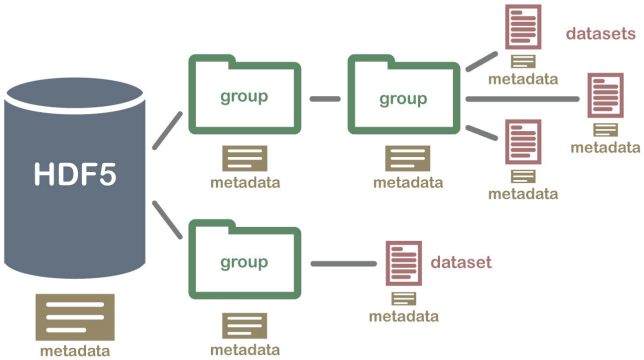

Abbildung: Aufbau einer HDF5 Datei [6]

NetCDF + HDF5

- NetCDF-4 baut HDF5 Dateien, welche vollständig dem HDF5 Standard entsprechen und von einer HDF5 Anwendung gelesen werden können
- HDF5 Dateien können hingegen nicht immer von NetCDF gelesen werden
- NetCDF-4 unterstützt einige Eigenarten von HDF5 nicht, wie zum Beispiel "looping groups"

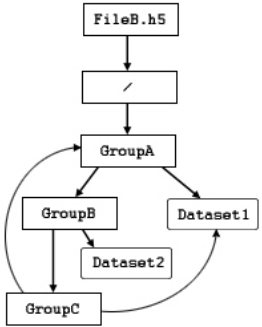

Abbildung: "Looping Group" in einer HDF5 Datei [7]

NetCDF Parallelisierung

- NetCDF schreibt standardmäßig seriell
- Parallel NetCDF(PnetCDF) bietet parallele I/O für die älteren Formate CDF-1, CDF-2 und CDF-5
- NetCDF-4 unterstützt die integrierte Parallelisierung von HDF5
- Beide Varianten bauen auf MPI-IO auf

NetCDF E/A Stack

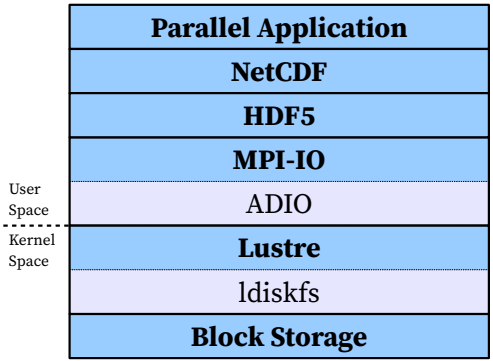

Abbildung: Typischer E/A Stack über NetCDF [8]

Zusammenfassung

- POSIX
  - Standard für E/A Operationen
  - Schwierig threadsicher zu parallelisieren
  - Hohe Konsistenz Anforderungen
- MPI-IO
  - Leicht threadsicher zu parallelisieren
  - Geringere Konsistenz Anforderungen
- NetCDF
  - Leicht zu verwendende High-level library
  - Plattformunabhängig und kompatibel zu HDF5

Quellen I

[1] W. Fischer, “Linux Storage Stack Diagram.” [Online]. Available: https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram

[2] IBM, “Architectural view of the Linux file system components.” [Online]. Available: https://developer.ibm.com/developer/tutorials/l-linux-filesystem/images/figure1.gif

[3] Guru99, “Threading Grafik.” [Online]. Available: https://career.guru99.com/wp-content/uploads/2017/11/thrading.png

[4] S. Mendez, S. Lührs, D. Sloan-Murphy, A. Turner, and V. Weinberg, “Best Practice Guide – Parallel I/O,” 02 2019. [Online]. Available: http://www.prace-ri.eu/best-practice-guide-parallel-i-o/

Quellen II

[5] W. Gropp, "Lecture 32: Introduction to MPI I/O," 2016. [Online]. Available: http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture32.pdf

[6] N. Science, "HDF5 Structure." [Online]. Available: https://www.battelleecology.org/sites/default/files/images/HDF5/hdf5_structure4.jpg

[7] H. Group, "HDF5 Looping Group." [Online]. Available: https://support.hdfgroup.org/HDF5/doc1.6/UG/Images/Group_fig2,8.jpg

[8] M. Kuhn, "Julea: A flexible storage framework for hpc," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham: Springer International Publishing, 2017, pp. 712–723.

Quellen III

[9] UCAR, “Interoperability with HDF5.” [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/docs/interoperability_hdf5.html

[10] Wikipedia, “NetCDF.” [Online]. Available: https://en.wikipedia.org/wiki/NetCDF

[11] ——, “POSIX.” [Online]. Available: https://en.wikipedia.org/wiki/POSIX

[12] A. N. Laboratory, “PnetCDF: A Parallel I/O Library for NetCDF File Access.” [Online]. Available: https://trac.mcs.anl.gov/projects/parallel-netcdf

[13] M. Kuhn, “Hochleistungs-Ein-/Ausgabe - Dateisysteme.” [Online]. Available: https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2019/hea-19-dateisysteme.pdf

Quellen IV

[14] G. Lockwood, “What’s So Bad About POSIX I/O?” [Online]. Available: https://www.nextplatform.com/2017/09/11/whats-bad-posix-io/