IF2230 Sistem Operasi

Semester 2 Tahun Ajaran 2014/2015

Tugas Besar 2

Jang Berikoetnja

Anggota :

Muhammad Nizami / 13512501

Ignatius Alriana Haryadi Moel / 13513051

Lucky Cahyadi Kurniawan / 13513061

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2015

I. Deskripsi Masalah

Pada tugas kali ini, FUSE API, API untuk implementasi filesystem pada user-space. dimanfaatkan untuk membuat filesystem bernama Poi-FS. Poi-FS ini diimplementasi dengan bahasa C ataupun C++, dengan compiler GCC ataupun G++, pada sistem operasi Linux.

Poi-FS yang dibuat adalah filesystem yang di-mount dari sebuah file dengan ekstensi .poi Setiap file dan folder pada volume Poi-FS sebenarnya disimpan dalam file ber-extension .poi tersebut.

Fungsi-fungsi yang diimplementaskan untuk tugas ini adalah sebagai berikut;

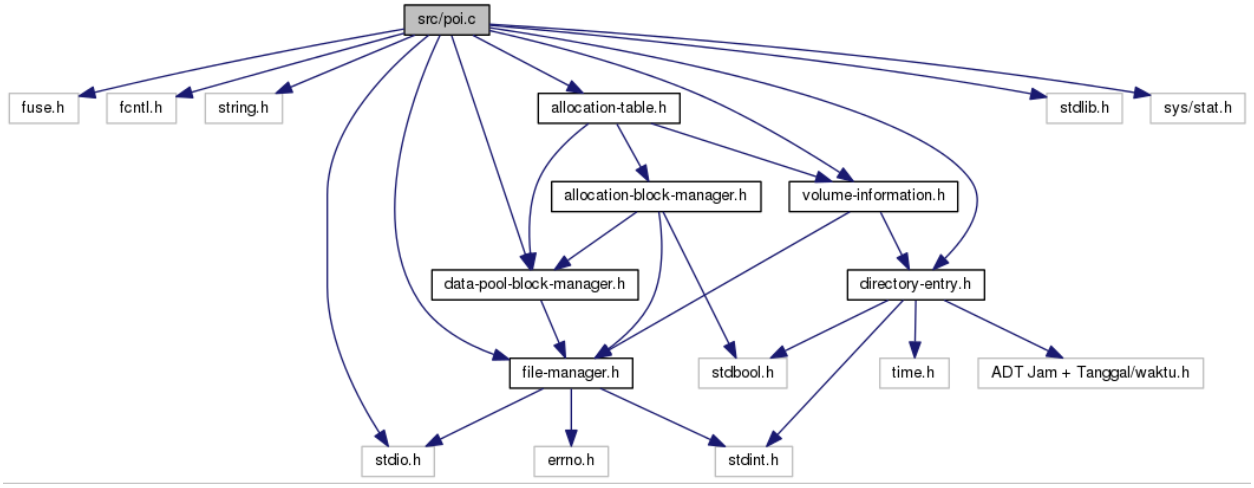
1. getattr
2. readdir
3. mkdir
4. mknod
5. read
6. rmdir
7. unlink
8. rename
9. write
10. truncate
11. chmod (bonus)
12. link (bonus)
13. open (bonus)

II. Analisis

Jelaskan apa yang diimplementasikan melalui FUSE ini. Bagaimana cara kerja FUSE dalam membangun suatu filesystem?

Dengan FUSE, maka Filesystem dapat diimplementasikan di userspace tanpa mengetahui isi dari filesystem dan programming kernel. Di dalam kernel, ada sebuah module yang mirip dengan module filesystem untuk menerima perintah-perintah yang diberikan dari userspace dan mentranslasikan perintah-perintah tersebut ke dalam bentuk yang bisa diterima oleh kernel. Di user space , FUSE mengimplementasikan library yang digunakan untuk berkomunikasi dengan modul di dalam kernel tadi,

Implementasi fungsi-fungsi ini disatukan dalam satu berkas, yang menggunakan modul-modul lain sebagai berikut:



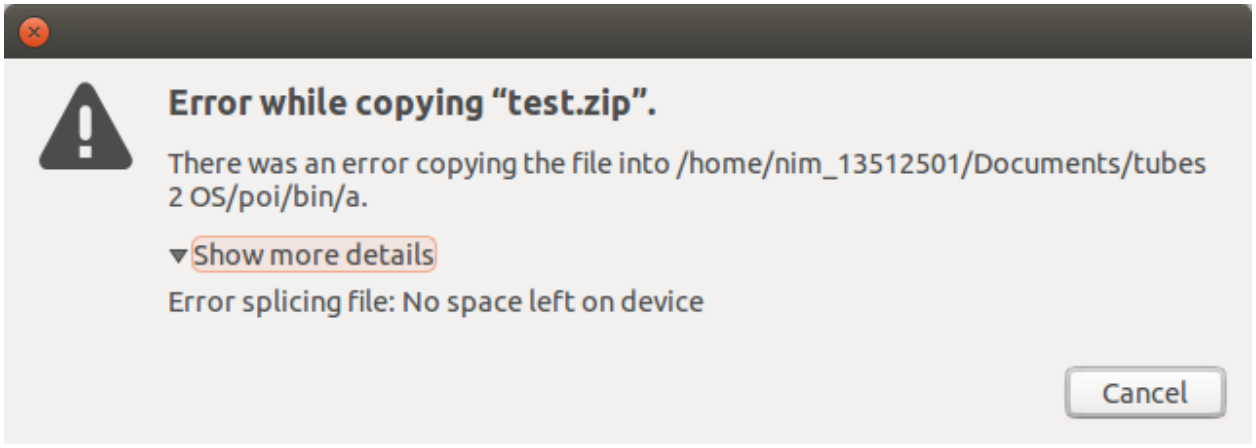
Dengan tugas tiap modul sebagai berikut:

Modul	Tugas
allocation-block-manager.h	Mengambil dan menyimpan pointer to next di allocation table. Tidak memperhitungkan free space.
allocation-table.h	Mengatur seluruh senarai berkait alokasi berkas Memperhitungkan juga ruang kosong. Bila ingin aman dengan perhitungan ruang kosong, panggil ini dan tidak langsung ke **allocation-block-manager.h** dalam hal manipulasi senarai. (dalam hal pengambilan saja, aman untuk mengambil langsung ke**allocation-block-manager.h**)
data-pool-block-manager.h	Mengatur pembacaan dan penulisan blok kantong data.

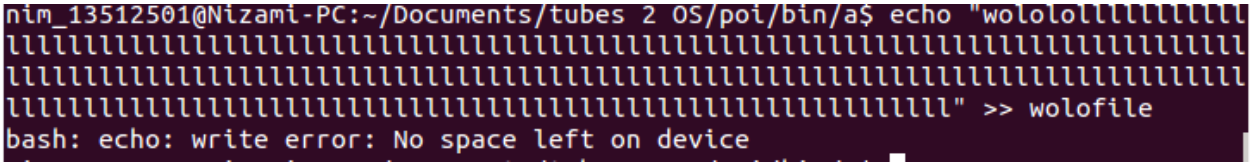
	Ini adalah struktur data entri direktori yang sudah dalam bentuk array of byte. Untuk menulis, tinggal disalin ke array of byte lain lalu ditulis. penyalinan array of byte dapat menggunakan memcpy langsung ke **directory_entry.bytearr**
directory-entry.h	Struktur data entri direktori. Modul ini mengatur seluruh masukan-luaran (i/o) langsung kepada berkas .poi yang ditunggangi. Seluruh modul lain HARUS menggunakan modul ini untuk masukan-luaran ke berkas .poi yang ditunggangi
file-manager.h	Manajer berkas dan blok. Modul ini bertugas untuk membaca dan menulis informasi volume pada blok pertama berkas .poi.
poi.c	Berisi implementasi interface ke fuse dan main program
volume-information.h	Mengatur informasi volume

Implementasi fungsi-fungsi yang dibutuhkan oleh FUSE terdapat pada berkas **poi.c**. Detilnya dapat dilihat pada berkas doxygen terlampir.

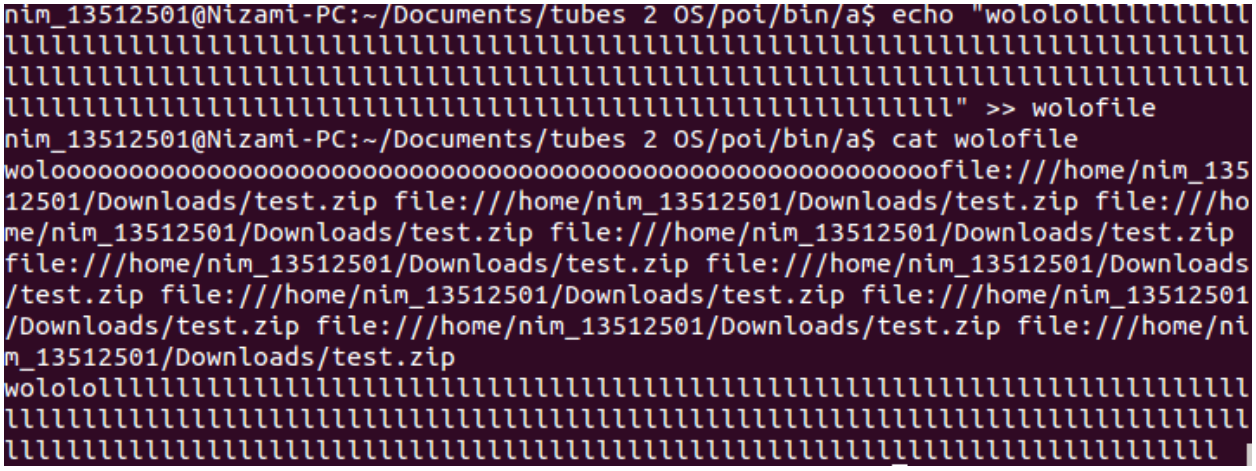
Untuk menguji perilaku yang terjadi ketika blok kantung data telah penuh, dilakukan penyalinan berkas sebesar 87.2 MB. Hasil yang terjadi adalah sebagai berikut:



Dilakukan pula pembubuhan terhadap berkas saat sistem berkas telah penuh, dengan hasil sebagai berikut:



namun, setelah test.zip dihapus, pembubuhan terhadap berkas menghasilkan luaran sebagai berikut:



III. Hal-hal yang menarik

Salah satu hal yang menarik dalam implementasi sistem berkas adalah propagasi galat. Ketika terjadi galat, karena persistensi sistem berkas, maka kesalahan tersebut akan menyebabkan kesalahan pada operasi-operasi berikutnya. Ketika operasi sistem berkas terhadap suatu berkas atau direktori di dalam sistem berkas mengalami galat, sangat mungkin berkas atau direktori lain ikut terpengaruh.
Propagasi galat ini menjadi lebih terlihat ketika diperhatikan bahwa setiap operasi memiliki percabangan yang sangat kompleks dan harus menangani perintah yang tidak dapat dilakukan karena berbagai batasan, seperti ukuran, nama, dan sebagainya.

Hal lain yang menarik adalah kompleksitas waktu penulisan berkas baru dalam ukuran besar. Pembacaan berkas dalam ukuran besar tidak memanggil write() dengan size seluruhnya, namun meminta pembacaan tersebut sedikit demi sedikit. Dalam suatu kasus, pembacaan berkas berukuran beberapa MB dilakukan dengan banyak perintah write dengan size 0x1000. Karena tabel alokasi diimplementasi dengan senarai lanjar dengan penunjuk hanya ke kepala (dan tidak ke ekor), maka setiap kali perintah write() dipanggil, dibutuhkan traversal ke ekor untuk menemukan tempat menulis, sehingga kompleksitas waktu penulisan menjadi kuadratik.