System Programming Weekly Log 3

Nama : Mahardika Krisna Ihsani

Kelas : A

NPM : 1806141284

What I've learned in Week 3

Pada week 3, Saya mempelajari tentang system call yang berkaitan dengan I/O, di antaranya adalah stat, open, close, read, dan write. Selain itu, Saya juga mempelajari bagaimana I/O di UNIX bekerja. Pada log ini, Saya akan membahasnya satu per satu.

Untuk mendapatkan informasi suatu file, sistem operasi akan mengakses inode dari file tersebut. Inode-inode dari berbagai file akan tersimpan dalam suatu inode table. Informasi yang tersimpan bisa berupa waktu, permission, dll. Untuk melakukan pengaksesan informasi ini, diperlukan suatu system call yang bernama "stat". "stat" terbagi menjadi 3 jenis yaitu "fstat", "lstat", dan "stat". Ketiga hampir identik namun terdapat sedikit perbedaan. Pada "lstat", apabila path file yang diberikan berupa symbolic link, maka informasi yang dikeluarkan adalah berupa informasi dari symbolic link daripada file yang di-point oleh symbolic link. Pada "fstat" mengeluarkan informasi dari file berdasarkan file descriptor. Stat ini mempunyai structure yang dimana terkandung informasi dari file. Stat structure adalah sebuah struct yang berguna untuk menyimpan informasi mengenai file. Structure ini menyimpan informasi berupa: ID dari device yang menyimpan file, nomor inode, tipe file dan mode file, hard link, User ID pemilik file, Group ID pemilik file, total ukuran, ukuran blok filesystem I/O

Sebelumnya disinggung bahwa terdapat term "file descriptor". Sebenarnya apakah file descriptor itu? File descriptor adalah ID unik yang di-assign untuk suatu file yang dibuka. File descriptor ini pada dasarnya berfungsi sebagai pointer terhadap file description. File description sendiri memuat informasi-informasi yang berkaitan dengan file seperti inode pointer, status flag, dan file offset. Dalam UNIX, terdapat 3 file descriptor standard yaitu

- 0 (STDIN_FILENO): Untuk standard input
- 1(STDOUT_FILENO): Untuk standard output
- 2(STDERR_FILENO): Untuk standard error

Pada I/O terdapat 4 system call yang esensial:

- fd = open(pathname, flags, mode) : Membuka file

- numread = read(fd, buffer, count) : Membaca file sebanyak “count” bytes dari file descriptor hingga ke buffer yang dimulai dari “buffer”
- numwritten = write(fd, buffer, count) : Mengisi/Memodifikasi file sebanyak “count” bytes dari buffer yang ditunjuk oleh “buffer” ke file descriptor “fd”
- status = close(fd) : Menutup file descriptor sehingga file bisa digunakan oleh proses yang lain atau digunakan kembali

System call creat() ekuivalen dengan system call open(path, O_WRONLY|O_CREAT|O_TRUNC, mode). Perlu diperhatikan bahwa menggunakan standard file descriptor pada close akan menyebabkan race condition yang menyebabkan file menjadi corrupt sehingga hal tersebut tidak direkomendasikan untuk dilakukan. Selain itu apabila kita mengeksekusi open(), sebaiknya kita melakukan close() apabila selesai membuka file itu. Hal itu digunakan untuk mengurangi *resource waste* yang sebenarnya tidak diperlukan dan juga agar tidak terjadi *concurrency issue* seperti tidak bisa diaksesnya suatu file oleh proses lain, padahal file tersebut sudah selesai diproses namun lupa untuk ditutup dengan menggunakan close().

Pernah disinggung di log sebelumnya bahwa sistem UNIX menerapkan universality in I/O. Artinya bahwa kita bisa menerapkan operasi i/o tidak hanya ke file saja namun bisa juga diterapkan ke suatu device. Sebagai contoh kita bisa mengkopi konten dari satu file ke terminal maupun sebaliknya.

Salah satu library function yang menerapkan system call i/o adalah copy(). Yang dilakukan copy() pada dasarnya adalah dia membuat dua file descriptor. Satu file descriptor akan di-assign ke satu file input sebagai READ ONLY dan satu file descriptor lainnya akan di-assign sebagai file output dan akan di-assign sebagai WRITE ONLY. Dari sini yang dilakukan melakukan system call read pada file descriptor input lalu melakukan write() dengan data dari data pada buffer yang saat itu terambil ke file descriptor output hingga mencapai end of file atau hasil numread-nya=0. Setelah selesai mengkopi data, maka akan dipanggil system call close() pada kedua file descriptor tersebut.

Selain keempat system call tersebut, terdapat dua system call lainnya, yakni

- lseek(int fd, off_t offset, int whence): Mengatur reposisi offset pada file yang diacu oleh file descriptor “fd” dengan offset berdasarkan “whence” sebagai berikut:
    - SEEK_SET : Offset adalah letak absolut pada file itu
    - SEEK_CUR : Offset akan bersifat relatif terhadap offset yang saat ini di-point
    - SEEK_END : Offset akan bersifat relatif terhadap ukuran file
- Ioctl(): Berfungsi untuk memanipulasi *special device*

Something That's Still Unclear for Me in Week 3

Saya masih belum paham apakah proses daemon sendiri benar-benar tidak menggunakan file descriptor karena sebagian proses daemon masih menggunakan file descriptor berupa 3 standard file descriptor.

Referensi

- https://stackoverflow.com/questions/8175827/what-happens-if-i-dont-call-fclose-in-a-c-program
- https://stackoverflow.com/questions/8502005/what-is-the-difference-between-open-and-creat-system-call-in-c#:~:text=The%20creat%20function%20creates%20files,can%20only%20be%20written%20to.&text=creat()%20is%20equivalent%20to,files%20in%20the%20%2Fdev%20folder.
- https://medium.com/@copyconstruct/file-descriptor-transfer-over-unix-domain-sockets-dcbbf5b3b6ec
- https://lwn.net/Articles/357767/
- https://man7.org/linux/man-pages/
- http://www.cems.uwe.ac.uk/~irjohnso/coursenotes/lrc/system/daemons/d3.htm
- https://stackoverflow.com/questions/11095474/why-do-i-have-to-use-close-to-close-a-file
- https://www.computerhope.com/jargon/f/file-descriptor.htm
- https://www.slashroot.in/inode-and-its-structure-linux
- https://youtu.be/WV0rwKwcqc4
- https://youtu.be/O6pAyQwcCdc