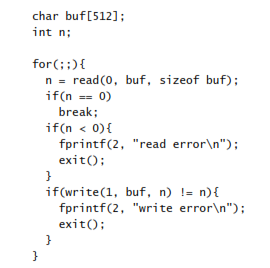
Deskriptor file adalah bilangan bulat kecil yang mewakili objek yang dikelola kernel tempat proses dapat membaca atau menulis. Suatu proses dapat memperoleh deskriptor file dengan membuka file, direktori, atau perangkat, atau dengan membuat pipa, atau dengan menduplikasi pendeskripsi yang ada. Untuk kesederhanaan, kita akan sering merujuk ke objek yang dirujuk oleh deskriptor file sebagai '' file ''; antarmuka deskriptor file memisahkan perbedaan antara file, pipa, dan perangkat, membuat semuanya tampak seperti aliran byte.

Secara internal, kernel xv6 menggunakan deskriptor file sebagai indeks ke tabel per-proses, sehingga setiap proses memiliki ruang privat deskriptor file yang dimulai dari nol. Menurut konvensi, proses membaca dari deskriptor file 0 (input standar), menulis output ke deskriptor file 1 (output standar), dan menulis pesan kesalahan ke deskriptor file 2 (kesalahan standar). Seperti yang akan kita lihat, shell mengeksploitasi konvensi untuk mengimplementasikan pengalihan dan pipeline I / O. Shell memastikan bahwa ia selalu memiliki tiga deskriptor file yang terbuka (8707), yang merupakan deskriptor file default untuk konsol.

Sistem baca dan tulis memanggil byte baca dari dan byte tulis ke file terbuka yang diberi nama oleh deskriptor file. Panggilan read (fd, buf, n) membaca paling banyak n byte dari deskriptor file fd, menyalinnya ke buf, dan mengembalikan jumlah byte yang dibaca. Setiap deskriptor file yang merujuk ke file memiliki kumpulan file yang terkait dengannya. Pembacaan membaca data dari kumpulan file saat ini dan kemudian memajukan yang ditetapkan dengan jumlah byte yang dibaca: pembacaan berikutnya akan mengembalikan byte setelah yang dikembalikan oleh pembacaan pertama. Jika tidak ada lagi byte untuk dibaca, read mengembalikan nol untuk menandai akhir dari file.

Panggilan tulis (fd, buf, n) menulis n byte dari buf ke deskriptor file fd dan mengembalikan jumlah byte yang ditulis. Kurang dari n byte ditulis hanya jika terjadi kesalahan. Seperti membaca, menulis menulis data pada kumpulan file saat ini dan kemudian memajukan yang ditetapkan oleh jumlah byte yang ditulis: setiap penulisan mengambil tempat yang ditinggalkan sebelumnya oﬀ.

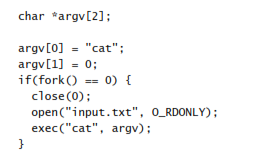
Fragmen program berikut (yang membentuk esensi cat) menyalin data dari input standarnya ke output standarnya. Jika terjadi kesalahan, itu menulis pesan ke kesalahan standar.



Hal penting yang perlu diperhatikan dalam fragmen kode adalah bahwa cat tidak tahu apakah ia membaca dari file, konsol, atau pipa. Demikian pula, kucing tidak tahu apakah itu mencetak ke konsol, file, atau apa pun. Penggunaan deskriptor file dan konvensi bahwa deskriptor file 0 adalah input dan deskriptor file 1 adalah output memungkinkan implementasi sederhana dari cat.

Close system call merilis deskriptor file, membuatnya gratis untuk digunakan kembali oleh panggilan sistem open, pipe, atau dup di masa depan (lihat di bawah). Deskriptor file yang baru dialokasikan selalu merupakan deskriptor tak terpakai bernomor paling rendah dari proses saat ini.

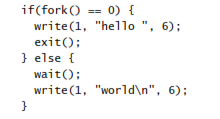
Deskriptor file dan fork berinteraksi untuk membuat pengalihan I / O mudah diterapkan. Fork menyalin tabel deskriptor file induk bersama dengan memorinya, sehingga anak memulai dengan file terbuka yang sama persis dengan file induknya. Eksekusi panggilan sistem menggantikan memori proses panggilan tetapi mempertahankan tabel filenya. Perilaku ini memungkinkan shell untuk mengimplementasikan pengalihan I / O dengan forking, membuka kembali deskriptor file yang dipilih, dan kemudian menjalankan program baru. Berikut adalah versi sederhana dari kode yang dijalankan shell untuk perintah cat <input.txt:



Setelah anak menutup deskriptor file 0, open dijamin untuk menggunakan deskriptor file itu untuk input.txt yang baru dibuka: 0 akan menjadi deskriptor file terkecil yang tersedia. Cat kemudian mengeksekusi dengan deskriptor file 0 (input standar) yang mengacu pada input.txt.

Kode untuk pengalihan I / O di shell xv6 bekerja persis seperti ini (8630). Panggil ulang bahwa pada titik ini dalam kode shell telah bercabang dengan shell anak dan runcmd akan memanggil exec untuk memuat program baru. Sekarang sudah jelas mengapa sebaiknya fork dan exec adalah panggilan yang terpisah. Karena jika terpisah, shell dapat melakukan fork pada anak, menggunakan open, close, dup pada anak tersebut untuk mengubah deskriptor file input dan output standar, dan kemudian exec. Tidak ada perubahan pada program yang sedang dijalankan (cat dalam contoh kita) diperlukan. Jika fork dan exec digabungkan menjadi satu panggilan sistem, beberapa skema lain (mungkin lebih kompleks) akan diperlukan agar shell dapat mengalihkan input dan output standar, atau program itu sendiri harus memahami cara mengalihkan I / O.

Meskipun garpu menyalin tabel deskriptor file, setiap kumpulan file yang mendasari dibagikan antara induk dan anak. Pertimbangkan contoh ini:



Di akhir fragmen ini, file yang dilampirkan ke deskriptor file 1 akan berisi data hello world. Tulisan di induk (yang, berkat menunggu, hanya berjalan setelah anak selesai) mengambil tempat tulis anak tersisa o ﬀ. Perilaku ini membantu menghasilkan keluaran berurutan dari urutan perintah shell, seperti (echo hello; echo world)> output.txt.

Panggilan sistem dup menduplikasi deskriptor file yang sudah ada, menghasilkan deskriptor baru yang merujuk ke objek I / O dasar yang sama. Kedua deskriptor file berbagi satu set file, sama seperti deskriptor file yang diduplikasi oleh fork. Ini adalah cara lain untuk menulis hello world ke dalam sebuah file:



Dua deskriptor file berbagi satu set file jika mereka diturunkan dari deskriptor file asli yang sama dengan urutan panggilan fork dan dup. Jika tidak, deskriptor file tidak membagikan kumpulan file, bahkan jika mereka dihasilkan dari panggilan terbuka untuk file yang sama. Dup memungkinkan shell untuk mengimplementasikan perintah seperti ini: ls existing-file non-existing-file> tmp1 2> & 1. 2> & 1 memberitahu shell untuk memberikan perintah deskriptor file 2 yang merupakan duplikat dari deskriptor 1. Baik nama dari file yang ada dan pesan kesalahan untuk file yang tidak ada akan muncul di file tmp1. Shell xv6 tidak mendukung pengalihan I / O untuk deskriptor file kesalahan, tetapi sekarang Anda tahu cara mengimplementasikannya.

Deskriptor file adalah abstraksi yang kuat, karena menyembunyikan detail dari apa yang terhubung dengannya: proses yang menulis ke deskriptor file 1 mungkin menulis ke file, ke perangkat seperti konsol, atau ke pipa.