Sistemi Operativi, Secondo Modulo, Canale A–L e Teledidattica
Riassunto della lezione del 03/05/2021

Igor Melatti

Le chiamate di sistema (*system call* o *syscall*)

- Considerazioni sul `man`
  - le system call sono quelle in `apropos -s 2 .`
  - quelle della libreria standard sono in `apropos -s 3 .` (sezione 3)
  - in entrambe le sezioni 2 e 3 viene mostrato il prototipo, cosa è necessario includere, ed una descrizione di argomenti e semantica della funzione
  - libreria standard (`printf, scanf, fopen`...): qualsiasi architettura/sistema operativo con un compilatore C vi permette di usare le funzioni ivi contenute (Linux, Windows, macosx...)
  - system call: sono specifiche di un sistema operativo, qui ci concentriamo su quelle di Linux
  - pertanto, le syscall *non* sono portabili: non funzionerebbero (o meglio, non è garantito che siano definite), ad esempio, sotto Windows
  - comunque, le system call di Linux sono sostanzialmente quelle di Unix, che hanno "ispirato" gli altri sistemi operativi oggi più in uso (Windows compreso)
- Le "vere" chiamate alle syscall andrebbero fatte come descritto in `man 2 syscall` e `/usr/include/x86_64-linux-gnu/sys/syscall.h`
  - andrebbero fatte in assembler perché, per motivi di sicurezza, le system call usano uno stack a parte
    - remember modulo 1: lo stack usato dalle system call è nella zona di memoria riservata al kernel
    - tanto, le system call sono eseguite in modalità sistema...
  - per semplificare queste chiamate, si usano dei wrapper, che sono quelli della sezione 2

  - alcuni sono wrapper di wrapper, nel senso che chiamano un wrapper anziché la syscall "vera"
    * per esempio, le varie funzioni `exec` (vedere `man 3 exec`; sì, sono tutte nella libreria standard) sono tutte basate su `execve` (che invece è una chiamata di sistema)
    * altro esempio: la `fork`, in realtà, è solo un wrapper per la system call `clone`
  - per lo più, hanno lo stesso nome, o quasi, della corrispondente syscall
  - in alcuni casi, i wrapper hanno un numero variabile di argomenti (in C, si può), per permettere di passare qualche argomento in meno
  - per esempio, la `open` dovrebbe prendere 3 argomenti, ma il wrapper ammette anche la versione con 2
  - gli argomenti non dati vengono, ovviamente, settati a degli opportuni default
  - le syscall "vere" hanno sempre il numero massimo di argomenti
  - ovviamente, sfruttano le syscall anche svariate funzioni della libreria standard, come quelle che hanno a che vedere con i file
  - sfruttano le syscall anche `malloc, calloc, realloc, free` ed `alloca`, che chiamano la syscall `brk`; inoltre, tail funzioni non sono solo wrapper, ma hanno anche delle strutture dati per permettere la `free` di singole variabili (non gestito dalle syscall!)

- In caso di errore, i wrapper settano anche `errno`, e una spiegazione dell'errore può essere ottenuta chiamando le funzioni (della libreria standard!) `strerror` e `perror` (vedere `man errno` e l'esempio `strerror.c` allegato)
  - attenzione: non fare la free del risultato di `strerror`! Si tratta di una string statica, e non allocata con `malloc` o simili
- Comando `strace`, permette di vedere tutte le syscall effettuate, sia di processi ongoing che di comandi da passare a `strace` stesso
  - vedere `man strace`

Le syscall per la gestione dei file

- Standard e portabili: `fopen`, `fclose`, `fseek`, `ftell`, `rewind`, `fread`, `fwrite`, `fscanf`, `fprintf` (capitoli 15 e 16)
- Syscall, disponibili solo sotto Linux: `open`, `close`, `lseek`, `read`, `write`
  - niente output o input formattato (a meno di non bypassare il tutto con `fdopen`)

  - niente `rewind` ed `ftell`, che vanno realizzati con `lseek`

- Altre syscall sui file, senza corrispettivo sulla libreria standard: `chown`, `chmod`, `stat`, `lseek`, `select`, `ioctl`, `unlink`, `fcntl`, `symlink`, `link`, `chdir`, `rename`, `mkdir`, `rmdir`, `chroot`
- Le syscall della libreria standard sono applicabili solo a file regolari e a directory; quelle di Linux tengono conto di tutti i tipi di file disponibili (in Linux, tutto ciò che non è un processo è un file...)
  - devices
  - pipe
  - fifo
  - socket
- Le syscall non includono solo apertura, chiusura, lettura, scrittura e posizionamento (come le standard), ma anche operazioni speciali
- Vedere gli esempi allegati (sono corredati di commenti ed esercizi), in quest'ordine: `open.c`, `open_read_write.c`, `lseek.c`, `dup.c`, `dup2.c`, `dup3.c`, `stat.c`, `fcntl.c`, `fcntl2.c`, `ioctl.c`, `select.c`, `readdir.c`, `chdir.c`, `chroot.c`, `mouse.c`
- Per ogni syscall menzionata, leggere il `man`
- Per `dup`, tenere sempre presente la Figura 1
- Qualche precisazione in più per la `select`
  - ci sono delle read e delle write che possono essere bloccanti
  - ad esempio, è bloccante la read sullo stdin, in quanto va in attesa dell'input da tastiera (a meno che non ci siano redirezioni...)
  - sono bloccanti, a meno di indicazioni contrarie al momento dell'apertura, i meccanismi di comunicazione tra processi (ci ritorneremo nelle prossime lezioni): pipe, fifo e socket
  - ad un certo punto, un'operazione bloccata può diventare disponibile, e occorre a quel punto servirla (leggendo i dati se è una read, scrivendoli se è una write)
  - ci sono 2 modi per fare tutto ciò:
    1. come abbiamo visto finora, ovvero effettuare nel codice la read-/write bloccante. Come conseguenza, il processo sarà in stato BLOCKED finché l'operazione non diventa possibile, e a quel punto l'operazione viene effettuata non appena lo scheduler riseleziona nuovamente il processo
    2. anziché rimanere bloccati, si usa la select per sapere se un'operazione su un certo file descriptor (già aperto...) è possibile oppure no. Questo permette:

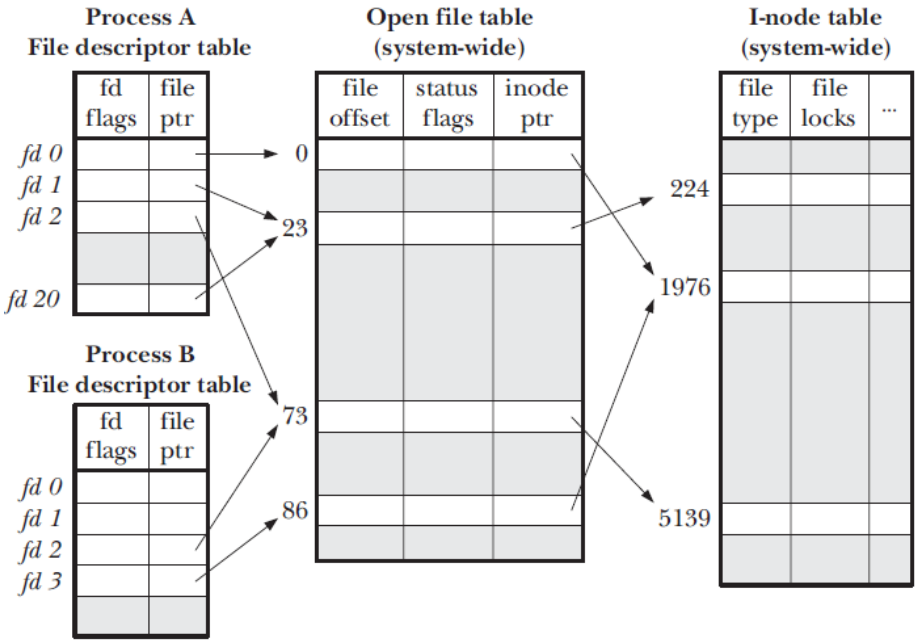

Figure 1: Gestione files in Linux

(a) di effettuare altre operazioni nel caso in cui una read/write sia bloccante

(b) rimanere in attesa finché almeno un file descriptor di un determinato insieme di file descriptor (tutti già aperti...) non diventa disponibile in lettura/scrittura

(c) una qualche combinazione dei due punti precedenti

– per l'opzione 2, si usa la select

– `select.c` dà un esempio della possibilità 2a

- **esercizio:** creare degli esempi per le seguenti syscall: `chmod`, `chown`, `link`, `symlink`, `unlink`, `rename`, `mkdir`, `rmdir`, di modo che si comportino esattamente (almeno per le opzioni principali viste nelle lezioni 2 e 3) come gli equivalenti comandi bash.