Sistemi Operativi, Secondo Modulo, Canale M–Z
Riassunto della lezione del 18/05/2016

Igor Melatti

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

- Considerazioni sul `man`
  - prototipo, sezione 2
  - le system call sono quelle in `apropos -s 2 .`
- 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
  - 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 lo più, hanno lo stesso nome, o quasi, della corrispondente syscall; in alcuni casi, ci sono più versioni della stessa syscall, essenzialmente la differenza è che viene settato un default per alcuni argomenti
  - le syscall vere hanno sempre il numero massimo di argomenti
  - contengono delle syscall anche alcune funzioni della libreria standard (che sono invece nella sezione 3 del `man`), come quelle che hanno a che vedere con i file
  - contengono delle syscall anche `malloc, calloc, realloc, free` ed `alloca`, che chiamano la `sbk`; inoltre, tail funzioni non sono solo wrapper, ma hanno anche delle strutture dati per permettere la `free` di singole variabili
  - diversamente dalle funzioni della standard library (es. printf, fopen, ...) le syscall *non* sono portabili: non funzionerebbero (o meglio, non è garantito che siano definite), ad esempio, sotto Windows
- In caso di errore, i wrapper settano anche `errno`, e una spiegazione dell'errore può essere ottenuta chiamando le funzioni (non di sistema) `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`, `fgets`, `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; quelle di Linux tengono conto di tutti i tipi di file disponibili (in Linux, tutto ciò che non è un processo è un file...)
  - directory
  - devices
  - pipe
  - fifo
  - socket
- A parte apertura e chiusura, le operazioni non includono solo lettura, scrittura e posizionamento, 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`
- Per ogni syscall menzionata, leggere il `man`
- 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 (vedere lezione 24): 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:
     (a) di effettuare altre operazioni che possono essere eseguite frattanto che le operazioni di read/write diventano possibili
     (b) di effettuare altre operazioni nel caso in cui una read/write sia bloccante, scartando eventuali dati quando diventano possibili
     (c) rimanere in attesa finché almeno un file descriptor di un determinato insieme di file descriptor (tutti già aperti...) non diventa disponibile in lettura/scrittura
     (d) una qualche combinazione dei due punti precedenti
- per l'opzione 2, si usa la select
- `select.c` dà un esempio della possibilità 2b

- **esercizio:** riconsiderare la struttura dati `datiutente` in `esempio_allocazione` di lezione 21. Modificarlo in modo che legga da file il suo input esattamente come in `esempio_allocazione`, stampi su stdout l'attuale contenuto della struttura dati `datiutente` (usando le normali funzioni standard) e poi la scriva anche su un file, usando delle syscall (niente funzioni standard con output formattato). Scrivere poi un secondo programma che legga la struttura stessa da file, e ne stampi nuovamente il risultato su stdout. Verificare con uno script che i due output siano coincidenti.

- **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) degli equivalenti comandi bash.