Sistemi Operativi, Secondo Modulo, Canale A–L e Teledidattica
Riassunto della lezione del 09/05/2022

Igor Melatti

Le syscall per la gestione dei processi

- Il primo processo è init, con PID 1 (esercizio: verificare che ci sia...)
- Da lì in poi, i processi sono tutti figli di un altro processo
  - quindi, tutti discendono da `init`
- La creazione avviene così:
  - un certo processo, già in esecuzione, deve effettuare una chiamata alla syscall `fork`
  - l'effetto è quello di creare una copia *quasi* esatta di sé stesso
    - questo ovviamente vale al momento stesso della creazione; dopodiché, con le sue prime istruzioni, il figlio tipicamente comincerà subito a distinguersi dal padre
    - il "quasi" è dovuto al fatto che le seguenti informazioni sono diverse anche al momento stesso della creazione:
      - il PID del processo: ovviamente, il PID del figlio è diverso dal PID del padre
      - il PID del padre: sia per il padre che per il figlio, si tratta del PID del processo che lo ha creati, ed ovviamente sono due processi diversi con PID diversi
      - nel caso tipico dei nostri esempi, il padre del padre è la shell, da cui il padre stesso viene avviato; nel caso del figlio, è per l'appunto il padre...
      - lock su file o su memoria detenuti dal padre: continuano ad essere mantenuti solo dal padre, altrimenti addio alla mutua esclusione
      - contatori risorse: ovviamente, ad esempio, il padre poteva essere in esecuzione da molto tempo prima di creare il figlio, per il quale però il tempo di esecuzione riparte da 0

        - i timer: sono quelli legati principalmente alla syscall `alarm(seconds)`, che fa sì che, dopo `seconds` secondi, venga inviato un `SIGALRM` al processo chiamante. Tali timer ripartono da zero per il processo figlio
  - il processo che chiama la `fork` diventa il *padre* del processo *figlio* appena nato
  - in modo un po' innaturale, ci si aspetta che il padre sopravviva al figlio: infatti l'exit status del figlio viene restituito al padre
    - il padre deve esplicitamente "ricevere" tale exit status, effettuando una chiamata alla syscall `wait`
    - fintantoché non lo fa, il figlio, pure se terminato, resta in parte in memoria, con lo stato di "zombie"
    - fintantoché è nello stato di "zombie", il figlio non può fare nessuna operazione, ma il suo process control block resta in memoria proprio per far sì che il suo exit status venga dato al padre quando quest'ultimo chiama la `wait`
    - se il padre, nel frattempo, termina senza chiamare `wait`, il processo verrà adottato dal processo init
    - periodicamente, il processo init effettua una `waitpid`, quindi a quel punto si risolve tutto
    - pertanto, per vedere un processo zombie occorre che il figlio sia terminato mentre il padre è ancora in esecuzione (ma senza aver chiamato la `wait`), o che il padre sia terminato senza chiamare `wait` e il processo `init` non l'abbia ancora fatto a sua volta

- I processi possono comunicare tra loro sia tramite segnali (in questa lezione) che tramite canali dedicati (nella prossima lezione)
  - i segnali sono per forza di cose limitati: essenzialmente comunicano solo un numero
  - con i canali si possono scambiare informazioni di qualsiasi tipo

- Vedere gli esempi allegati (sono corredati di commenti ed esercizi), in quest'ordine: `getppid.c`, `getgid.c`, `getuid.c`, `setgid.c`, `setuid.c`, `print_env.c`, `execve.c`, `getenv.c`, `setenv.c`, `signal.c`, `kill.c`, `sigaction.c`, `sigprocmask.c`, `sigsuspend.c`, `pause.c`, `abort.c`, `exit.c`, `fork.c`, `fork2.c`, `wait.c`, `waitpid.c`

- Per ogni syscall menzionata, leggere il `man`; non tutte sono nella sezione 2, ma tutte hanno a che vedere con le syscall