**ESERCIZI PRATICI Corso di Sistemi Operativi 1 A.A. 2018-2019 – Versione 2 (da consegnare dalla sessione autunnale in poi).**

**1)** Utilizzando soltanto gli strumenti presentati nel corso, modificare il semplice interprete di comandi "smallsh" presentato nelle esercitazioni in modo da:

1. ammettere la possibilità di lanciare comandi in background con la notazione:

   `comando &`

   (il riconoscimento di questo caso è già previsto nel codice, ma il programma si comporta come nel caso senza `&`);

2. per i comandi lanciati in background, stampare informazioni sul fatto che il comando è terminato; per i comandi (in bg o fg) terminati da un segnale, informare analogamente l'utente. A tal scopo può essere utile l'opzione `WNOHANG` della `wait` (vedere man);

3. ammettere la possibilità di interrompere un comando con il segnale di interruzione, senza però interrompere anche l'interprete (che è ciò che avviene nella versione fornita). L''interprete deve ignorare il segnale di interruzione solo quando è in corso un comando in foreground, mentre deve poter essere interrotto negli altri casi;

4. stampare il prompt della shell nel formato
   `%<nome_utente>:<home_utente>:`
   Ad esempio:
   *`%davide:/home/davide:`*
   le informazioni su nome e home dell'utente devono essere ricavate a tempo di esecuzione della shell leggendo le corrispondenti variabili di ambiente iniziali;

5. tenere traccia[1] tramite una variabile d'ambiente `BPID` dell'elenco dei processi attualmente in background. Tale variabile deve essere aggiornata quando si crea un nuovo processo in background e quando se ne cattura la sua terminazione (vedi punto 2).
   Il formato della variabile d'ambiente è:
   $BPID=pid_1:pid_2:...pid_n$
   Ad esempio:
   `BPID=12034:12045:13089`
   La shell deve prevedere anche un comando `bp` che stampi il contenuto della variabile `BPID`. Il comando `bp` è interno, ossia quando l'utente digita la stringa `bp`, la shell la riconosce e stampa il contenuto di `BPID` **senza creare un nuovo processo che esegua tale comando .**

**2) Utilizzando le system call POSIX per la gestione di semafori (sem_init, sem_wait e sem_post) e le funzioni per la gestione dei processi, risolvere il seguente problema:**

Una tribù di N selvaggi mangia in comune da una pentola che può contenere fino ad M porzioni di stufato, si assume che inizialmente la pentola sia piena. Quando un selvaggio ha fame controlla la pentola:
i) se non ci sono porzioni, sveglia il cuoco ed attende che questo abbia completamente riempito di

---

1 **Punto opzionale, ma necessario per ottenere la lode per la parte di Laboratorio.**

nuovo la pentola prima di servirsi;
ii) se la pentola contiene almeno una porzione, se ne appropria.
Il cuoco controlla che ci siano delle porzioni e, se ci sono, si addormenta, altrimenti cuoce M porzioni e le mette nella pentola. Ciascun selvaggio deve mangiare NGIRI volte prima di terminare.
Il numero di selvaggi N, di porzioni M e NGIRI dovranno essere richiesti come argomenti da linea di comando per facilitare esperimenti al variare di tali parametri. **I selvaggi ed il cuoco devono essere implementati come processi separati che lavorano su variabili e semafori condivisi.** Definire anche un parametro che permetta di contare complessivamente quante volte il cuoco riempie la pentola prima del termine del programma. Il programma termina quando tutti i selvaggi hanno completato il loro ciclo. Durante l'esecuzione stampare opportuni messaggi al fine di determinare il comportamento dei vari processi.