# LabSO A.A. 2016-2017

**Progetto modulo di Laboratorio**

*VERIFICARE SEMPRE EVENTUALI AGGIORNAMENTI PRIMA DELLA CONSEGNA NEGLI ANNUNCI ONLINE. QUESTO DOCUMENTO VALE COME RIFERIMENTO, MA LE INDICAZIONI UFFICIALI SONO QUELLE FORNITE COMUNQUE IN AULA.*

- Impostare una cartella con nome:
  `LABSO_PROGETTO_2016-2017--`*`MATRICOLA1_MATRICOLA2_MATRICOLA3`*

  *nota: salvo casi particolari i gruppi sono costituiti da 3 componenti (adattare ovviamente le indicazioni per gli altri casi). Riportare i numeri di matricola sempre in ordine crescente, sia nelle denominazioni delle risorse sia nella registrazione online.*

- La consegna deve avvenire inviando la cartella "zippata" - creando un file con lo stesso nome ed estenzione ".zip" in modo che all'interno ci sia la cartella stessa e tutto il contenuto (e non direttamente il contenuto) - via e-mail usando come oggetto lo stesso nome seguito dalla parola maiuscola CONSEGNA.

  es. nome file:
  `LABSO_PROGETTO_2016-2017--`*`MATRICOLA1_MATRICOLA2_MATRICOLA3`*`.zip`

  es. oggetto e-mail
  `LABSO_PROGETTO_2016-2017--`*`MATRICOLA1_MATRICOLA2_MATRICOLA3`* `CONSEGNA`

- La cartella deve contenere solo ed esattamente un file `Makefile` e una cartella src con i sorgenti "`.c`", "`.h`" ed eventuali altri files che si ritengano necessari

- Il progetto deve seguire le specifiche indicate nel testo prescelto e può opzionalmente disporre di una modalità di "test" (sempre secondo quanto indicato)

- Il Makefile contiene (oltre ad eventuali altre regole aggiuntive di appoggio che si ritengano utili) le regole:
  - `help` (default): deve mostrare a video i componenti del gruppo (nome, cognome, matricola), una descrizione brevissima del progetto e le regole utilizzabili
  - `clean`: deve cancellare la/e cartella/e build ed eventualmente assets se presente/i
  - `build`: deve creare una cartella "build" con dentro il/i file(s) eseguibile/i richiamando prima clean
  - `assets` (opzionale): deve creare una cartella "assets" con dentro il/i file(s) di supporto richiamando prima build (ad esempio per la modalità di "test")
  - `test` (opzionale): esegue il progetto in modalità "test" richiamando prima assets

- La compilazione deve avvenire con il tool gcc senza opzioni aggiuntive oltre al flag "-o" per scegliere il nome del file di output

- L'ambiente di riferimento è quello indicato nel testo del progetto

- Si possono usare solo le funzionalità di base discusse, le librerie standard C e fork/pipe/fifo/segnali per la comunicazione tra i processi (niente socket)

- La valutazione è orientativamente:
    - 18-22 per un'implementazione delle funzionalità "base" (v. testo del progetto)
    - 23-26 se si implementano le funzionalità "avanzate" (v. testo del progetto)
    - 27-29 se si implementa anche la modalità di test
    - 30-30L per una resa particolare (ad esempio output avanzato e/o opzioni extra)

- Registrarsi al form online https://goo.gl/forms/YKr00cGeZriyI3Yf1 entro il 04/05/2017 indicando il progetto scelto ed eventuali note aggiuntive, poi in sequenza i componenti del gruppo (il primo è ovviamente obbligatorio e poi solitamente dovrebbero essere indicati il secondo e il terzo, salvo casi particolari… è presente anche uno spazio per un quarto componente sempre per casi particolari. Lasciare "vuote" le sezioni non usate e poi salvare.

- Inviare il progetto finito all'indirizzo labso1@unitn.elementica.com esclusivamente nel periodo 30/04/2017 - 31/05/2017 comunque DOPO essersi registrati, attenendosi strettamente all'oggetto indicato in questo documento, allegando il file ".zip" come specificato e riportando nel corpo del messaggio i dati del form di registrazione (progetto scelto ed elenco componenti con tutti i dati)

- Dopo la consegna sarà fatta una pre-valutazione del progetto e si indicherà eventualmente una data per la discussione in aula dello stesso

**LabSO A.A. 2016-2017 - Progetto 1: Project Manager**

Realizzare un gestore di processi con eseguibile denominato `pmanager` che attraverso una shell interattiva custom accetti i seguenti comandi:

[livello base]
`phelp`: stampa un elenco dei comandi disponibili
`plist`: elenca i processi generati dalla shell custom
`pnew <nome>`: crea un nuovo processo con nome <nome>
`pinfo <nome>`: fornisce informazioni sul processo <nome> (almeno *pid* e *ppid*)
`pclose <nome>`: chiede al processo <nome> di chiudersi
`quit`: esce dalla shell custom

Nel caso in cui il processo da terminare abbia figli, devo restituire un errore. Altrimenti far ereditare i figli ad altri processi (complesso).

[livello avanzato]
`pspawn <nome>`: chiede al processo <nome> di clonarsi creando <nome_i> con i progressivo
`prmall <nome>`: chiede al processo <nome> di chiudersi chiudendo anche eventuali cloni
`ptree`: mostra la gerarchia completa dei processi generati attivi

[modalità test]
Se l'eseguibile è richiamato con un parametro, questo è interpretato come il nome di un file di testo contenente un elenco di comandi da eseguire come se fossero stati inseriti direttamente dalla shell custom. Tale file può essere presente nel progetto e copiato dentro la cartella di servizio "assets" o essere generato "al volo" (magari dinamicamente)

Ambiente di riferimento: Ubuntu 14.x

Il manager consente attraverso una shell custom interattiva e i comandi indicati di generare nuovi processi abbinandoli a un nome (stringa alfanumerica breve senza spazi) e interagire con essi. La chiusura deve avvenire da parte del processo interessato su richiesta di quello principale. Ogni comando deve stampare a video una breve informazione di cosa sta facendo (ad esempio `pspawn <nome>` può mostrare un testo tipo "*`Richiesta di clonazione inviata al processo <nome>`*") ed ogni processo deve stampare un testo informativo quando esegue qualche azione particolare (ad esempio all'avvio "*Processo <pid> avviato*" o quando si clona "*Clonazione avvenuta: proceso <pid> generato*" o simili).

**LabSO A.A. 2016-2017 - Progetto 2: Countdown LED**

Realizzare un sistema di countdown con eseguibile denominato `countdown` che attraverso una shell interattiva custom accetti i seguenti comandi:

[livello base]
`start <secs>`: avvia un countdown per i secondi indicati (massimo 59) se non già avviato
`elapsed`: mostra i secondi rimasti
`stop`: ferma il timer corrente se attivo
`tens`: mostra lo stato dei led delle decine
`units`: mostra lo stato dei led delle unità
`quit`: esce dalla shell custom

[livello avanzato]
`tensled info <n>`: mostra le info del led n-esimo delle decine (almeno il colore)
`tensled color <n> <color>`: imposta il colore del led n-esimo delle decine
`unitsled info <n>`: mostra le info del led n-esimo delle unità (almeno il colore)
`unitsled color <n> <color>`: imposta il colore del led n-esimo delle unità

[modalità test]
Se l'eseguibile è richiamato con un parametro, questo è interpretato come il numero dei secondi del timer che è avviato immediatamente e "stampato" su un file.

Ambiente di riferimento: Ubuntu 14.x o RASPBERRY

Il counter deve lavorare utilizzando almeno due sotto-processi indipendenti: uno per gestire le "decine" e uno le "unità" del tempo trascorso, da rappresentare tramite cifre con led standard a 7 segmenti (come nei classici orologi digitali). A loro volta ciascuno di tali sotto-processi deve utilizzare 7 sotto-sotto-processi, uno per ogni segmento, cui è abbinata una colorazione. Ad ogni segmento deve essere abbinato un file denominato `tens_led_`*`i`* o `units_led_`*`i`* rispettivamente per decine ed unità per il led i-esimo per un totale di 14 files (tens_led_1, tens_led_2, …, tens_led7 e units_led_1, units_led_2, …, units_led7) contenente il colore del led stesso (la stringa) oppure "off" se spento. I colori sono almeno "black", "red", "green", "blue", "yellow" e "white".
Nella versione RASPBERRY si può utilizzare l'output dei led per "pilotare" un display ad-hoc integrabile appositamente.