Laboratorio di Sistemi Operativi
a.a. 2003-2004

Esercitazione 2

Renzo Davoli
Alberto Montresor

Esercitazione 2

- **Scopo**
  - Sviluppare codice C di una certa complessità
  - Prendere dimestichezza con segnali
  - Prendere dimestichezza con tool quali **`make`**
- **Note:**
  - Non è necessario utilizzare UML
- **Passi:**
  - Scrivere un programma, basato su segnali, che permette a due processi di comunicare tramite segnali (nota: non solo sincronizzarsi!)
  - Scrivere un make file per compilare, testare e rimuovere i file del programma

Esercitazione 2

- **Scrivere un programma diviso in due fasi:**
  - nella prima fase, ogni processo deve scoprire il pid dell'altro processo con cui comunicare, cercando nel file system /proc e individuando un processo che esegue lo stesso eseguibile ma con pid diverso dal proprio
  - nella seconda fase
    - ogni processo prende in input da stdin un testo ASCII
    - per ogni 64 bit di input:
      - codifica i 64 bit tramite DES (vedi lucidi successivi)
      - spedisce i 64 bit utilizzando i segnali USR1 e USR2
    - contemporaneamente:
      - riceve 64 bit dall'altro processo
      - decodifica tali bit
      - stampa la stringa così ottenuta
    - il processo termina quando riceve un segnale di EOF dall'altro processo

Esercitazione 2

- **Per estrarre i bit dall'input**
  - Utilizzate gli operatori logici &, |, ~
- **Per codificare i set di bit utilizzando DES**
  - Utilizzate la funzione **`encrypt()`** (**`man encrypt`** per ulteriori dettagli)
  - Per i più temerari: implementate DES!
- **Per spedire i bit:**
  - Utilizzate **`SIG_USR1`** per spedire un bit 0, **`SIG_USR2`** per spedire un bit 1
  - E' necessario che l'altro processo risponda con un segnale di acknowledgement (conferma), per evitare di perdere segnali
  - utilizzate **`sigsuspend()`**
  - non utilizzate **`pause()`** o **`sleep()`** (per carità!)

Esercitazione 2

- **Inizializzazione, comunicazione full-duplex, padding, terminazione**
  - Quando due processi p e q comunicano contemporaneamente:
    - ogni segnale da **p** a **q** viene utilizzato sia per trasportare un bit da **p** a **q** che come acknowledgement per un precedente segnale da **q** a **p**
    - il primo processo a spedire è quello con pid minore
  - Quando un processo non ha nulla da spedire (temporaneamente)
    - spedisce un carattere "\0" (null)
    - "\0" può essere utilizzato anche come padding
    - utilizzate read non bloccanti per evitare deadlock
  - Quando un processo vuole terminare
    - spedisce un carattere "\033" per segnalare all'altro processo la sua intenzione di terminare

Makefile

- **Il makefile deve contenere:**
  - un target principale
    per compilare il vostro programma
  - un target **`test`**
    per lanciare due processi concorrenti cahe si mandano due distinti file di testo
  - un target **`clean`**
    per eliminare tutti i file oggetto e eseguibili

Valutazione

- **Nella valutazione del lavoro verranno tenuti in considerazione i seguenti aspetti:**
  - Struttura (architettura, ingegnerizzazione) del programma:
    - divisione in moduli, mancanza di codice ripetuto, linearità e corretta definizione funzioni, scope delle variabili globali, mancanza di side-effect delle funzioni, architettura all'interno del file system, makefile
  - Leggibilità del codice:
    - documentazione (interna ed esterna), chiarezza dichiarativa del codice (uso di costanti, scelta dei nomi delle costanti, variabili, funzioni e macro....), indentazione del codice, ...
  - Coerenza con le specifiche di funzionamento
  - Gestione di casi particolari (e/o loro discussione nella documentazione)
  - Funzionamento, effiicienza

Commenti

- **In italiano o in inglese (italiano corretto o inglese leggibile)**
- **Dovete inserire la quantità minima di commenti che renda chiaro il funzionamento del programma.**
  - tanti commenti sono un costo per un'azienda:
    +commenti = +tempo per leggerli
  - pochi commenti sono un costo per un'azienda:
    -commenti = +tempo per comprendere il programma
- **I commenti non sono valutati in base alla quantità bensì in base alla qualità (chiarezza)**

Commenti

- **Commenti interni**
  - 1 commento all'inizio del file che indichi quale caratteristica accomuna le funzioni del modulo
  - 1 commento prima di ogni funzione per indicare cosa la funzione faccia
  - 1 commento per ogni punto criptico.
- **Commenti esterni**
  - in testo, html, pdf

Errori da evitare

- **GRAVISSIMI**
  - goto, break all'interno di cicli
  - file "dos" (con caratteri speciali come ^M)
- **GRAVI**
  - mancanza di makefile, main "che fanno tutto loro", side-effect,
- **altri errori**
  - mancanza di coerenza fra le parti realizzate dai vari componenti, parti sottocommentate, errori di ortografia
  - autori nominati col cognome in testa
  - parole inglesi menzionate in italiano con la s relativa al plurale o, peggio ancora, alla terza persona nei verbi

Cosa consegnare

- **Consegnate:**
  - con estensione .tgz e nome uguale al nome del gruppo
  - contenente una directory uguale al nome del gruppo
  - la directory deve contenere
    - tutti i file che avete scritto
    - file AUTHORS (lista autori)
    - file README (vedi dopo)

Esempio:

```
% ls
lso02a150.tgz
% tar zxvf lso02a150.tgz
% ls
lso02a150 lso02a150.tgz
% cd lso02a150
% ls
README AUTHORS file1.c
   file1.h file2.c Makefile
```

Cosa, come, quando consegnare

- **Cosa non consegnare:**
  - dischetti o materiale cartaceo
  - file di prova, materiale non correlato all'esercizio
- **Come consegnare:**
  - copiando il file .tgz in una directory del file system trusted che verrà indicata in seguito nel newsgroup
- **Quando consegnare**
  - entro 16 maggio 2004, ore 23.59