# Laboratorio di Sistemi Operativi
# 22 settembre 2014
### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (2 punti) Si supponga che uno script contenga i seguenti comandi

   ```
   var_prova='variabile di prova'
   echo $var_prova
   ```

   Ovviamente, quando viene lanciato in esecuzione visualizza sul terminale il messaggio `variabile di prova` perché tale è il valore con cui è stata inizializzata la variabile `var_prova` nello script.

   Se, dopo l'esecuzione dello script, lanciamo dalla shell il comando `echo $var_prova`, cosa otteniamo a video? Perché?

   **Soluzione:**

   A video non compare nulla, in quanto la variabile `var_prova` è definita nell'ambiente della sottoshell in cui viene eseguito lo script. Peranto al termine dell'esecuzione di quest'ultimo, torna in gioco l'ambiente della shell principale (in cui la variabile non è definita).

2. Si mostri qual è l'output prodotto dai seguenti comandi:
   (a) (2 punti) `echo abcdcd | sed '1,$y/dc/zy/'`
   (b) (2 punti) `echo abababab | sed '1,$s/ab/_/2'`
   (c) (2 punti) `echo abcdab | sed '1,$s/ab/_/g'`

   **Soluzione:**
   (a) `abyzyz`
   (b) `ab_ab`
   (c) `_cd_`

3. Classificare come vere o false le seguenti affermazioni (per quelle false giustificare le risposte):
   (a) (1 punto) Le modifiche apportate all'ambiente da parte di uno script della shell persistono al termine dell'esecuzione dello script anche nell'ambiente della shell chiamante.
   (b) (1 punto) Lo standard input (`stdin`), lo standard output (`stdout`) e lo standard error (`stderr`) sono normalmente collegati al terminale (il primo alla tastiera e gli altri due al video), ma possono essere rediretti facendo ricorso ad opportuni metacaratteri di redirezione.
   (c) (1 punto) La system call `fork()` può essere usata in combinazione con una delle system call della famiglia `exec` per sovrascrivere il processo figlio con il codice di un programma esterno.
   (d) (1 punto) Una `pipe` in C necessita di due file descriptor: uno per l'input e l'altro per l'output.
   (e) (1 punto) Un semaforo creato tramite un programma C non sopravvive all'esecuzione di quest'ultimo in nessun caso e non può essere visibile ad altri processi.
   (f) (1 punto) Un client non deve mai utilizzare la chiamata `bind` che è riservata al server, indipendentemente dal tipo di socket in uso.
   (g) (1 punto) Per la maggior parte dei segnali, se non vengono gestiti (tramite il cosiddetto *signal handling*), i processi eseguono una terminazione normale non appena li ricevono.

   **Soluzione:**
   (a) Falso: lo script viene eseguito in una sottoshell con un proprio ambiente (su cui le modifiche dello script hanno effetto). Quando la sottoshell si chiude (al termine dell'esecuzione dello script), tali modifiche vengono "perse".

(b) Vero.

(c) Vero.

(d) Vero.

(e) Falso: se non fosse visibile anche ad altri processi, non sarebbe di nessuna utilità per la loro sincronizzazione. Inoltre, per rimuovere un semaforo creato da un processo, deve essere invocata un'opportuna system call, altrimenti il semaforo continua a esistere nel sistema, anche dopo la terminazione del processo che lo ha originato.

(f) Falso: nel caso di socket connectionless, anche il client deve eseguire la `bind`.

(g) Vero.

4. (8 punti) Dato il seguente prototipo:

   ```
   int split(char *inputline, char *word, int size);
   ```

   si implementi la funzione `split` in modo che:

   - `inputline` punti ad un vettore di caratteri contenente una linea di testo di dimensione `size`, ovvero, il vettore contenga `size` caratteri +1 (il terminatore `\0`);
   - `word` punti ad un vettore di caratteri atto a ricevere una parola contenuta in `inputline` trovata dalla funzione `split`: per parola si intende una qualunque sequenza di caratteri diversi dai witespace character, ovvero, diversi da `' '` (spazio), `'\t'` (tab) e `'\n'` (newline);
   - il valore di ritorno rappresenti la lunghezza della parola estratta e memorizzata nel vettore `word`;
   - quando la funzione `split` viene richiamata la prima volta deve restituire la prima parola contenuta in `inputline`, quando viene richamata la seconda volta deve restituire la seconda parola contenuta in `inputline` e così via...
   - quando la funzione `split` viene richiamata con il primo parametro uguale a `NULL`, si resetta, ovvero, dalla prossima invocazione ricomincia a fornire nel vettore puntato da `word` la prima parola contenuta in `inputline`.

   Ad esempio se `inputline` puntasse alla stringa `The quick brown fox jumps ...`, la prima chiamata di `split` dovrebbe memorizzare in `word` la parola `The` (restituendo `3` come valore di ritorno). La seconda chiamata dovrebbe memorizzare in `word` la parola `quick` (di lunghezza `5`) ecc. Dopo una chiamata a `split(NULL,word,size)`, la successiva chiamata a `split` ricomincerebbe a memorizzare `The` in `word` e così via.

   **Suggerimento:** utilizzare una variabile globale oppure una variabile locale statica per tener traccia, fra una chiamata e l'altra, del carattere raggiunto nella stringa.

**Soluzione:**

```
1 int split(char *inputline, char *word, int size) {
2   int i,j=0,
3   /* flag per capire se è iniziata la scansione di una parola (1)
       oppure no (0) */
4   int word_begin=0;
5   /* last è l'indice del prossimo carattere da processare */
6   static int last=0;
7
8   /* Controllo se è il caso di resettare la scansione della stringa
       */
```

```
9   if(inputline==NULL) {
10    last=0;
11    return 0;
12  }
13
14  for (i=last; i<size; i++) {
15    /* salto i whitespace character */
16    if(inputline[i] == ' ' ||
17       inputline[i] == '\t' ||
18       inputline[i] == '\n') {
19        /* se word_begin vale 1, allora sono arrivato alla fine della
               parola, imposto il terminatore ed esco dal ciclo */
20        if(word_begin) {
21          word[j]='\0';
22          /* la prossima volta inizio la scansione dal carattere
                successivo */
23          last=i+1;
24          break;
25        }
26    }
27    else {
28      /* non è un carattere whitespace quindi inizia una parola e
             copio il carattere in word */
29      word_begin=1;
30      word[j++]=inputline[i];
31    }
32  }
33
34  return j;
35 }
```

5. (10 punti) Si completi il seguente frammento di programma C in cui è presente il codice che definisce il ciclo di servizio di un server che gestisce ogni nuova connessione tramite un processo figlio:

```
1   /* gestione delle connessioni dei client */
2   while (1) {
3     client_len = sizeof(client);
4     if ((fd = accept(sock, (struct sockaddr *)&client, &client_len)) <
          0) {
5       perror("accepting connection");
6       exit(1);
7     }
8
9     /* ogni volta che il server accetta una nuova connessione,
10     * quest'ultima viene gestita da un nuovo processo figlio
11     */
12    switch(fork()) {
13        case -1:
14            perror("Errore nella chiamata alla fork");
15            exit(2);
16        case 0:
17            fprintf(stderr, "Aperta connessione (PID %d).\n",getpid());
18            ...
19            fprintf(stderr, "Chiusa connessione (PID %d).\n",getpid());
20            exit(0);
```

```
21            default :
22                /* elimina eventuali figli zombie */
23                while(waitpid(-1, 0, WNOHANG)>0);}
24    }
```

Si completi il codice nel punto indicato (riga 18: è possibile inserire più righe di codice) in modo che, per ogni messaggio ricevuto da un client connesso, vengano inviate singolarmente a quest'ultimo le parole (intese come sequenze di caratteri delimitate dai whitespace character) contenute in tale messaggio.

Ad esempio, se un client inviasse il messaggio `The quick brown fox jumps ...`, il server dovrebbe rispondere al client quanto segue:

```
The
quick
brown
fox
jumps
...
```

**Suggerimento:** conviene utilizzare la funzione `split` definita nell'esercizio precedente.

**Soluzione:**

Al posto dei ...è possibile per esempio inserire il codice seguente:

```
word_list(fd, fd);
close(fd);
```

dove la funzione `word_list` è definita come segue:

```
1  void word_list(int in, int out) {
2    char inputline[LINESIZE],outputline[LINESIZE];
3    int  len,i;
4
5    /* Finché il client invia messaggi, eseguo il ciclo */
6    while ((len = recv(in, inputline, LINESIZE, 0)) > 0) {
7      /* Reset della scansione */
8      split(NULL,NULL,0);
9
10     /* Estraggo le parole dal messaggio corrente con la funzione
           split dell'esercizio precedente e le invio al client, ognuna
           seguita da un newline */
11     while((i=split(inputline,outputline,len))!=0) {
12       send(out, outputline, i, 0);
13       send(out, "\n", 1, 0);
14     }
15
16   }
17 }
```