Laboratorio di Sistemi Operativi
18 Luglio 2019
Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che stampi a video il numero di comandi contenuti nel file `.bash_history`, escludendo i duplicati. Ad esempio, se `.bash_history` contiene i seguenti comandi:

   ```
   cd /
   ls -al
   cp /etc/passwd ./copia_passwd
   ps ax
   ls -al
   cd
   ls -al
   ```

   l'output dovrà essere `5`.

   ```
   cat ~/.bash_history | sort | uniq | wc -l
   ```

2. (6 punti) si scriva uno script `somma.sh` della shell che calcoli e stampi a video la somma di tutti gli User ID contenuti nel file `/etc/passwd` (si ricorda che le linee del file sono organizzate in campi separati dai due punti `':'` e che il campo relativo allo User ID è il terzo).

   Ogni linea del suddetto file è simile alla seguente (lo User ID è il terzo valore, ovvero, `124`, nel caso della linea riportata):

   ```
   lightdm:x:124:116:Light Display Manager:/var/lib/lightdm:/bin/false
   ```

   Esempio di esecuzione:

   ```
   > ./somma.sh
   94012
   > _
   ```

   ```
   ids=`cat /etc/passwd | cut -d':' -f3`
   count=0

   for i in $ids
   do
     count=$[$count+$i]
   done

   echo $count

   exit 0
   ```

3. (8 punti) Il file `lista.txt` contiene una sequenza di interi separati fra loro da spazi bianchi. Ad esempio:

   ```
   4 -6 43  12 901
   13 20    -1 7
   ```

   Si specifichi una struttura dati ricorsiva per implementare una lista concatenata di interi e si scriva un programma C che effettui le seguenti operazioni:

1. legga i numeri interi dal file `lista.txt`;
2. per ogni intero letto dal file allochi un nodo della lista e vi memorizzi il numero intero.

Suggerimento: usare `fscanf()` che restituisce come valore il numero di conversioni effettuate con successo o `-1` se incontra `EOF`.

```
#include<stdio.h>
#include <stdlib.h>

struct node_t {
  int data;
  struct node_t *next;
};

typedef struct node_t Node;

Node *insert(Node *p,int d);

int main() {
  FILE *f=fopen("lista.txt","r");
  Node *head=NULL;
  int n;

  if(f!=NULL) {
    while(fscanf(f,"%d",&n)==1) {
      head=insert(head,n);
    }
  }

  fclose(f);

  return 0;
}

Node *insert(Node *p,int d) {
  Node *q=p;
  if(p==NULL) {
    p=(Node *)malloc(sizeof(Node));
    p->data=d;
    p->next=NULL;
  } else {
    while(q->next!=NULL) q=q->next;
    q->next=(Node *)malloc(sizeof(Node));
    q->next->data=d;
    q->next->next=NULL;
  }
  return p;
}
```

4. (6 punti) Si scriva un programma C che legga da standard input delle stringhe conformi al formato `n1 - n2` (dove `n1` e `n2` sono due interi) e stampi a video la differenza fra `n1` e `n2`. Ad esempio, se viene fornita la stringa `4 - 10` il programma deve stampare `-6`. Il programma termina quando incontra il carattere `EOF` oppure l'input viola il formato stabilito.

   Esempio di esecuzione (`[Ctrl-D]` simboleggia l'immissione di `EOF`):

```
> ./scansione
3 - 9
-6
43 - 87
-44
[Ctrl-D]
>
```

Suggerimento: usare `scanf()` che restituisce come valore il numero di conversioni effettuate con successo o `-1` se incontra `EOF`.

```c
#include <stdio.h>
#include <string.h>

int main() {
  int n1, n2;

  while(scanf("%d - %d",&n1,&n2)==2) {
    printf("%d\n",n1-n2);
  }

  return 0;
}
```

5. (8 punti) Si scriva un programma C (`fork_ps.c`) che generi un processo figlio e si metta in attesa della sua terminazione prima di terminare a sua volta. Il processo figlio deve eseguire il comando `ps` senza opzioni, se al padre non è stato passato nessun argomento sulla linea di comando. Nel caso invece in cui al padre venga passato un argomento sulla linea di comando, allora il figlio deve eseguire `ps` passando l'argomento come opzione del comando.

   Esempio:

```
> ./fork_ps      # il figlio esegue ps
> ./fork_ps -el  # il figlio esegue ps -el
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(int argc, char **argv) {
  pid_t pid;

  pid=fork();

  switch(pid) {
    case -1:
      fprintf(stderr,"fork failed!\n");
      return 1;
    case 0:
      if(argc==2)
        execlp("ps","ps",argv[1],NULL);
```

```
18        else
19          execlp("ps","ps",NULL);
20        fprintf(stderr,"execlp failed!\n");
21        return 1;
22      default:
23        waitpid(pid,NULL,0);
24    }
25
26    return 0;
27  }
```