

# Laboratorio di Sistemi Operativi

## 30 gennaio 2015

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Il comando `cp` può prendere come primo argomento una lista di file? In caso di risposta affermativa ed in quel caso, il secondo argomento deve sottostare a qualche vincolo o può essere un qualunque file od una qualunque lista di file?

#### Soluzione:

Sì, il comando `cp` può prendere come primo argomento una lista di file. In questo caso il secondo argomento deve essere obbligatoriamente una directory.

2. (3 punti) Qual è l'effetto del seguente comando?

```
tr abc ABC
```

Scrivere un comando equivalente usando `sed`.

#### Soluzione:

Il comando `tr abc ABC` predisponde la shell per accettare dell'input da parte dell'utente (infatti fa apparire il prompt secondario). Da quel momento fino alla pressione di `Ctrl+D` (fine file) tutte le linee inserite dall'utente verranno emesse sullo standard output con le lettere `a`, `b` e `c` convertite in maiuscolo.

Un comando `sed` equivalente è

```
sed y/abc/ABC/
```

3. (4 punti) L'output seguente mostra un frammento del contenuto del file `/etc/passwd`:

```
root::0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
mysql:x:124:134:MySQL Server,,,:/nonexistent:/bin/false
sync:x:4:65534:sync:/bin:/sync
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

Ogni linea contiene informazioni su un account del sistema in uso e tali informazioni sono formattate in campi separati dai due punti (`:`). Si scriva una pipeline che fornisca in output la lista dei soli nomi degli account (primo campo) ordinati numericamente in modo crescente in base allo user-id corrispondente (terzo campo). Ad esempio, per il frammento precedente l'output deve essere il seguente:

```
root
daemon
bin
sys
sync
mysql
```

#### Soluzione:

```
sort -t: -k3,4 -n /etc/passwd | cut -d: -f1
```

# Laboratorio di Sistemi Operativi

## 30 gennaio 2015

### Compito

4. (5 punti) Si predisponga uno script della shell che legga da riga di comando una successione di numeri interi positivi terminata da -1. Lo script deve poi produrre sullo standard output un istogramma (utilizzando ad esempio il carattere \*) tale che ogni linea sia lunga quanto il corrispondente numero della successione ricevuta in input (escluso il terminatore -1 ovviamente). Si ignori la gestione degli eventuali errori. Esempio:

```
> ./istog.sh 3 5 1 6 -1
***
*****
*
*****
```

#### Soluzione:

```
if test $# -eq 0
then
    echo "Usage: $0 n1 ... nk -1"
    exit 1
fi

n=$1

while ! test $n -eq -1
do
    i=0
    while test $i -lt $n
    do
        echo -n '*'
        i=${$i+1}
    done
    echo
    shift
    n=$1
done

exit 0
```

5. (5 punti) Sia data la seguente struttura ricorsiva in C:

```
struct elemento {
    int valore;
    struct elemento *prossimo;
};

struct elemento *lista=NULL;
```

Si scriva il codice per aggiungere alla lista dinamica vuota puntata da `lista` gli elementi 2, 12, 6.

#### Soluzione:

```
lista=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
    perror("Allocazione di memoria fallita!\n");
```

# Laboratorio di Sistemi Operativi

## 30 gennaio 2015

### Compito

```
    exit(1);
}

lista->valore=2;
lista->prossimo=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
    perror("Allocazione di memoria fallita!\n");
    exit(1);
}

lista->prossimo->valore=12;
lista->prossimo->prossimo=(struct elemento*)malloc(sizeof(struct elemento));

if(lista==NULL) {
    perror("Allocazione di memoria fallita!\n");
    exit(1);
}

lista->prossimo->prossimo->valore=6;
lista->prossimo->prossimo->prossimo=NULL;
```

6. (6 punti) Si scriva un programma C che crei un processo figlio e si metta in attesa della terminazione di quest'ultimo. Il figlio tuttavia decide di terminare il processo padre inviandogli il segnale SIGKILL.

**Suggerimento:** per ottenere il PID del padre si utilizzi la seguente system call:

#### SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

pid_t getppid(void);
```

DESCRIPTION: getppid() returns the process ID of the parent of the calling process.

#### Soluzione:

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

int main() {
    pid_t pid,ppid;

    switch(pid=fork()) {
    case -1:
        perror("Fork failed!\n");
        exit(1);
    case 0:
        printf("I am the son.\n");
        ppid=getppid();
```

# Laboratorio di Sistemi Operativi

## 30 gennaio 2015

### Compito

```
printf("Killing father (PID: %d)!\n",ppid);
kill(ppid,SIGKILL);
printf("Exiting (escaping :)\n");
break;
default:
    printf("I am the father.\n");
    waitpid(pid,NULL,0);
    printf("Son terminated!\n");
}

return 0;
}
```

7. (5 punti) Il programma seguente utilizza le funzioni sui semafori introdotte a lezione (i.e., p(), v() e initsem()) per creare NUM\_PROC processi figli e consentire ad ognuno di essi di scrivere il proprio PID nel file **registro.txt** nella linea corrispondente (il primo processo figlio nella prima linea, il secondo nella seconda ecc.). Ogni linea è lunga LENGTH caratteri: l'ultimo carattere è il newline.

Si completi il sorgente specificando i comandi mancanti da inserire al posto dei ... nei 5 punti indicati, affinché un solo processo per volta possa accedere in modo esclusivo al file **registro.txt**.

```
void scrivi(key_t semkey,int fd,int linea) {
    int i,semid;
    char buffer[LENGTH];

    if((semid=initsem(semkey))<0) {
        perror("Errore nell'apertura del semaforo!");
        exit(1);

    for(i=0;i<LENGTH-1;i++) buffer[i]=' '; // pulisce il buffer
    buffer[LENGTH-1]='\n'; // imposta il newline
    sprintf(buffer,"%d",getpid()); // scrive il PID nel buffer
    ... // <- inizio sezione critica: completare (1)
    ... // sposta il puntatore di lettura/scrittura sulla linea giusta: completare (2)
    ... // scrive nel file: completare (3)
    ... // <- fine sezione critica: completare (4)
}

int main() {
    key_t semkey=0x200;
    int fd,n,proc[NUM_PROC];

    fd=open("registro.txt",O_WRONLY | O_CREAT); // apre il file registro.txt in scrittura

    for(n=1;n<=NUM_PROC;n++) {
        switch(proc[n-1]=fork()) { // crea il figlio n-esimo
        case -1:
            perror("Fork fallita!");
            exit(1);
        case 0:
            scrivi(semkey,fd,n);
            exit(0);
        default:
            printf("Sono il padre %d: ho creato il figlio %d.\n",getpid(),proc[n-1]);
        }
    }
}
```

# Laboratorio di Sistemi Operativi

## 30 gennaio 2015

### Compito

```
for(n=1;n<=NUM_PROC;n++)  
    ... // <-- il padre attende la terminazione del figlio n-esimo: completare (5)  
  
close(fd);  
return 0;  
}
```

#### Soluzione:

Esempi di completamento dei punti indicati dall'esercizio:

1. p(semid);
2. lseek(fd,(linea-1)\*LENGTH,SEEK\_SET);
3. write(fd,buffer,LENGTH);
4. v(semid);
5. waitpid(proc[n-1],NULL,0);

# Laboratorio di Sistemi Operativi

## 06 Febbraio 2018

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) sapendo che il comando `date` produce un output come il seguente:

```
gio 1 feb 2018 15:45:26
```

si completi la pipeline seguente, aggiungendo quanto necessario (al posto dei ...) per estrarre l'anno:

```
date | tr -s ' ' | ...
```

A cosa serve il comando `tr -s ' '`?

Per completare la pipeline ed estrarre l'anno va aggiunto il comando `cut -d' ' -f4`. Il comando `tr -s ' '` serve a comprimere eventuali spazi multipli come per esempio quelli tra `gio` e `1` in modo da non sbagliare il riferimento numerico al campo da estrarre con il parametro `-f` del comando `cut`.

2. (5 punti) si scriva uno script `dim.sh` della shell che prenda come argomento sulla linea di comando il percorso di un file e, se quest'ultimo esiste, è un file regolare ed è leggibile dall'utente, ne stampi a video la dimensione in byte e stampi una serie di asterischi di lunghezza pari alla dimensione.

Esempio (si supponga che il file `indice.txt` sia lungo 7 byte):

```
> ./dim.sh indice.txt
Dimensione di indice.txt: 7 byte
*****
```

```
1 #!/bin/bash
2
3 if test $# -ne 1
4 then
5   echo "Utilizzo: $0 <file>"
6   exit 1
7 fi
8
9 if test -f $1 -a -r $1
10 then
11   dim=`wc -c < $1`
12   echo "Dimensione di $1: $dim byte"
13
14 i=0;
15
16 while test $i -lt $dim
17 do
18   echo -n '*'
19   i=$[i+1]
20 done
21
22 if test $dim -gt 0
23 then
24   echo
25 fi
26
27 else
```

# Laboratorio di Sistemi Operativi

## 06 Febbraio 2018

### Compito

```
28     echo "Il file passato da linea di comando deve esistere, essere
29         leggibile ed essere regolare."
30     exit 2
31 fi
32 exit 0
```

3. (8 punti) scrivere il codice di un programma C che riceva in input una lista di  $n$  parole e stampi a video tali parole in ordine lessicografico. Si gestisca a scelta (motivata) il formato di input/output.  
Ad esempio (se **es** è il nome del programma eseguibile):

```
$ ./es l3xixograph Ord3r WORDS aRe Ord3r3d bas3d on THE az order of their c0mp0nents
```

un possibile output è:

```
$ Ord3r, Ord3r3d, aRe, az, bas3d, c0mp0nents, l3xixograph, of, on, order, THE, their, WORDS
```

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

int main(int argc, char** argv)
{
    int i, j;
    char *temp;

    if(argc<2) {
        fprintf(stderr,"Utilizzo: %s stringa1 stringa2 ... \n",argv[0]);
        return 1;
    }

    for(i=1; i<argc-1; ++i){
        for(j=i+1; j<argc ; ++j){
            /* è accettabile anche l'uso di strcmp
             * (che distingue fra maiuscole e minuscole)
             */
            if(strcasecmp(argv[i], argv[j])>0) {
                temp=argv[i];
                argv[i]=argv[j];
                argv[j]=temp;
            }
        }
    }

    printf("\nIn lexicographical order: \n");
    for(i=1; i<argc; ++i){
        puts(argv[i]);
    }
    return 0;
}
```

4. (5 punti) Scrivere l'output del seguente programma C.

# Laboratorio di Sistemi Operativi

## 06 Febbraio 2018

### Compito

```
1 int main()
2 {
3     int levels=5;
4
5     for (int i = 0; i < levels; i++) {
6         for (int j = 0; j < levels - i; j++){
7             printf("-");
8         }
9         for (int k = 0; k < (2 * i + 1); k++){
10            printf("*");
11        }
12        printf("\n");
13    }
14    return 0;
15 }
```

```
-----*
-----**
---*****
--*****
-*****
```

5. (8 punti) Completare il codice (dove compaiono i ...) per trovare il numero più grande, utilizzando l'allocazione dinamica della memoria: calloc()

ricordate che: “calloc() Allocates space for an array elements, initializes to zero and then returns a pointer to memory”.

Sintassi di calloc():

```
ptr = (cast-type*)calloc(n, element-size);
```

**Nota bene:** al posto di una singola occorrenza dei ... è possibile inserire più linee di codice.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int i, num;
7     ...
8
9     printf("Enter total number of elements(1 to 100): ");
10    ...
11
12    // Allocates the memory for 'num' elements .
13    ...
14
15    if(data == ...)
16    {
17        printf("Error!!! memory not allocated.");
18        exit(0);
19    }
20
21    printf("\n");
```

# Laboratorio di Sistemi Operativi

## 06 Febbraio 2018

### Compito

```
23 // Stores the number entered by the user.
24 for(i = 0; i < num; ++i)
25 {
26     ...
27 }
28
29 // Loop to store largest number at address data
30 for(i = 0; i < num; ++i)
31 {
32     ...
33 }
34
35 printf("Largest element = ...", *data);
36
37 return 0;
38 }
```

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

int main()
{
    int i, num;
    float *data;

    printf("Enter total number of elements(1 to 100): ");
    scanf("%d", &num);

    // Allocates the memory for 'num' elements.
    data = (float*) calloc(num, sizeof(float));

    if(data == NULL)
    {
        printf("Error!!! memory not allocated.");
        exit(0);
    }

    printf("\n");

    // Stores the number entered by the user.
    for(i = 0; i < num; ++i)
    {
        printf("Enter Number %d: ", i + 1);
        scanf("%f", data + i);
    }

    // Loop to store largest number at address data
    for(i = 1; i < num; ++i)
    {
        // Change < to > if you want to find the smallest number
        if(*data < *(data + i))
            *data = *(data + i);
    }

    printf("Largest element = %.2f", *data);
```

**Laboratorio di Sistemi Operativi**  
**06 Febbraio 2018**  
**Compito**

```
    return 0;  
}
```

6. (4 punti) Dire se le seguenti operazioni sono corrette oppure errate, e motivare la risposta.

1 int c, \*pc;

- pc= c;
- \*pc = &c;
- pc = &c;
- \*pc = c;

(a) pc= c;

Sbagliato: si assegna un valore intero ad una variabile di tipo puntatore ad intero (ovvero, di tipo indirizzo). c non rappresenta un indirizzo.

(b) \*pc = &c;

Sbagliato: si assegna alla variabile puntata da pc (di tipo intero) l'indirizzo della variabile c.

(c) pc = &c;

Corretto: si assegna l'indirizzo della variabile c al puntatore pc. Dopo tale assegnamento pc punterà a c.

(d) \*pc = c;

Corretto: si assegna il valore intero memorizzato nella variabile intera c alla variabile di tipo intero puntata da pc (supponendo che pc sia stata correttamente inizializzata).

# Laboratorio di Sistemi Operativi

## 06 Luglio 2018

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Si scriva una pipeline per estrarre dal file `/etc/passwd` il terzo campo di ogni riga, ovvero, il numero intero che rappresenta lo user-ID dell'utente relativo a quella riga (si ricordi che il separatore di campo del file `/etc/passwd` è il carattere *due punti*).

La pipeline è la seguente:

```
cat /etc/passwd | cut -d: -f3
```

2. (5 punti) Sfruttando la soluzione dell'esercizio precedente, si scriva uno script della shell che produca in output la somma di tutti gli user-ID contenuti nel file `/etc/passwd`.

**Suggerimento:** si ricordi che `tail -n +k f` stampa le linee di `f`, a partire dalla `k`-esima, mentre `head -n +k f` stampa le prime `k` linee di `f`.

```
num_linee='wc -l < /etc/passwd'
i=1
somma=0

while test $i -le $num_linee
do
    linea='cat /etc/passwd | head -n +$i | tail -1'
    valore='echo $linea | cut -d: -f3'
    somma=$((somma+$valore))
    i=$((i+1))
done

echo $somma
```

3. (16 punti) Si scriva un programma C che:

- chieda all'utente di inserire il numero di righe e di colonne di una matrice;
- allochi dinamicamente lo spazio in memoria per la matrice;
- chieda all'utente di inserire gli elementi della matrice allocata;
- stampi la matrice a video;
- calcoli la trasposta della matrice;
- stampi la matrice trasposta a video.

Esempio di input/output:

```
1 Inserisci il n. di righe e di colonne della matrice: 2
2 3
3
4 Inserisci gli elementi della matrice:
5 Inserisci l'elemento a11: 2
6 Inserisci l'elemento a12: 3
7 Inserisci l'elemento a13: 4
8 Inserisci l'elemento a21: 5
9 Inserisci l'elemento a22: 6
10 Inserisci l'elemento a23: 4
11
12 Matrice inserita:
13 2 3 4
```

# Laboratorio di Sistemi Operativi

## 06 Luglio 2018

### Compito

```
14 5 6 4
15
16 Trasposta della matrice:
17 2 5
18 3 6
19 4 4
```

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

int main()
{
    int **a, **transpose;
    int r, c, i, j;
    printf("Enter rows and columns of matrix: ");
    scanf("%d %d", &r, &c);

    a=(int**)malloc(sizeof(int*)*r);
    transpose=(int**)malloc(sizeof(int*)*c);

    for(i=0; i<r; i++) {
        a[i]=(int *)malloc(sizeof(int)*c);
    }

    for(i=0; i<c; i++) {
        transpose[i]=(int *)malloc(sizeof(int)*r);
    }

    // Storing elements of the matrix
    printf("\nEnter elements of matrix:\n");
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
    {
        printf("Enter element a%d%d: ", i+1, j+1);
        scanf("%d", &a[i][j]);
    }

    // Displaying the matrix a[][] */
    printf("\nEnterd Matrix: \n");
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
    {
        printf("%d ", a[i][j]);
        if (j == c-1)
            printf("\n\n");
    }

    // Finding the transpose of matrix a
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j)
    {
        transpose[j][i] = a[i][j];
    }

    // Displaying the transpose of matrix a
```

# Laboratorio di Sistemi Operativi

## 06 Luglio 2018

### Compito

```
printf("\nTranspose of Matrix:\n");
for(i=0; i<c; ++i)
    for(j=0; j<r; ++j)
    {
        printf("%d ", transpose[i][j]);
        if(j==r-1)
            printf("\n\n");
    }

    return 0;
}
```

4. (4 punti) Si dica cosa stampa il seguente programma C (giustificando la risposta):

```
1 #include <stdio.h>
2
3 int main() {
4     int x;
5     int *p;
6
7     p=&x;
8     x=1;
9     printf("%d\n", (++x)+*p);
10    *p=1;
11    printf("%d\n", *p+(++x));
12    return 0;
13 }
```

Il programma C stampa

```
4
3
```

Infatti, la prima espressione incrementa `x` a 2, prima di valutarne il valore (grazie all'operatore di preincremento `++`), e poi ci somma nuovamente 2 in quanto `p` punta a `x`. La seconda espressione invece (dopo il reset di `x` a 1 per mezzo dell'assegnamento `*p=1`) valuta il contenuto della locazione di memoria puntata da `p`, ovvero, la locazione di `x`, ottenendo 1 a cui somma il valore di `x`, dopo aver incrementato quest'ultimo a 2 (grazie, nuovamente, all'operatore di preincremento).

5. (5 punti) Il programma seguente dichiara una variabile `visit_log` (un array di caratteri) e lancia in esecuzione un numero `MAX` di thread. Ognuno di essi deve accedere a `visit_log`, stamparne il valore corrente ed aggiornarlo con la stringa ricevuta tramite la chiamata `pthread_create`, ovvero, `msg[i]` per il thread `i`-esimo.

Si completi il sorgente specificando i comandi mancanti da inserire al posto dei ... nei 5 punti indicati, affinché un solo thread per volta possa accedere in modo esclusivo al vettore di caratteri `visit_log`.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <string.h>
5 #define MAX 10
6 #define STRLEN 81
7
```

# Laboratorio di Sistemi Operativi

## 06 Luglio 2018

### Compito

```
8 ...
9 char visit_log[STRLEN] = "vuoto";
10 void *update_log(void *ptr);
11
12 int main() {
13     pthread_t thread[MAX];
14     char msg[MAX][STRLEN];
15     int i;
16
17     for(i=0; i<MAX; i++) {
18         sprintf(msg[i], "%s%d", "Thread n.", i+1);
19         printf("%s\n", msg[i]);
20
21         if(pthread_create(&thread[i], NULL,
22             (void *)&update_log, ...) != 0) {           // <- completare (punto 2)
23             fprintf(stderr, "Errore nella creazione del thread n. %d/%d.\n",
24                 i+1, MAX);
25             exit(1);
26         }
27     }
28
29     for(i=0; i<MAX; i++) {
30         pthread_join(thread[i], NULL);
31     }
32
33     printf("Ultimo visitatore rilevato: %s\n", visit_log);
34     return 0;
35 }
36
37 void *update_log(void *ptr) {
38     printf("%s - in attesa di accedere al log\n", (char *)ptr);
39     ...                                         // <- completare (punto 3)
40     printf("%s - accesso al log; precedente visitatore rilevato: %s\n"
41             ,(char *)ptr, visit_log);
42     strcpy(visit_log, ...);                   // <- completare (punto 4)
43     printf("%s - log aggiornato\n", (char *)ptr);
44     ...                                         // <- completare (punto 5)
45     printf("%s - rilascio del log\n", (char *)ptr);
46 }
```

I punti vanno completati come segue:

1. pthread\_mutex\_t log\_mutex=PTHREAD\_MUTEX\_INITIALIZER;
2. (void \*)msg[i]
3. pthread\_mutex\_lock(&log\_mutex);
4. (char \*)ptr
5. pthread\_mutex\_unlock(&log\_mutex);

# Laboratorio di Sistemi Operativi

## 6 Luglio 2022

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si consideri il file `/etc/passwd` ed una variabile di ambiente `user`: si scriva una pipeline che stampi a video la riga (se esiste) del file `/etc/passwd` corrispondente al nome utente memorizzato nella variabile `user`. Ad esempio, se `user` ha come valore la stringa `root`, la pipeline deve stampare a video la linea di `/etc/passwd` relativa all'utente `root`:

```
root:x:0:0:root:/root:/bin/bash
```

Esempio di soluzione:

```
cat /etc/passwd | grep "^\$user"
```

2. (6 punti) Si realizzi uno script della shell Bash che prenda come parametro sulla linea di comando un percorso di un file di testo. Quest'ultimo deve essere composto da numeri interi separati da spazi bianchi (i.e., spazi, tabulazioni, newline), ad esempio:

```
2 4 10      -80  
7 5  
8  
  
1
```

Lo script deve leggere i numeri contenuti nel file e stampare a video il risultato della loro somma, ad esempio per il caso precedente:

```
-43
```

Si gestiscono eventuali errori dovuti al parametro mancante o alla non esistenza/accessibilità in lettura del file.

Esempio di soluzione:

```
1 if ! test $# -eq 1  
2 then  
3     echo "Utilizzo: $0 <file>"  
4     exit 1  
5 fi  
6  
7 if ! test -f $1 -a -r $1  
8 then  
9     echo "Errore nell'accesso al file $1"  
10    exit 2  
11 fi  
12  
13 s=0  
14  
15 for n in `cat $1`  
16 do  
17     s=$((s + n))  
18 done  
19  
20 echo $s  
21  
22 exit 0
```

# Laboratorio di Sistemi Operativi

## 6 Luglio 2022

### Compito

3. (8 punti) Scrivere il codice di un programma C che si comporti come lo script della shell dell'esercizio 2. Si gestiscano gli eventuali errori (parametro non presente, file non esistente o non accessibile in lettura, ecc.).

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     if(argc!=2) {
5         fprintf(stderr,"Uso: %s pathname\n",argv[0]);
6         return 1;
7     }
8
9     FILE *f=fopen(argv[1],"r");
10    if(f==NULL) {
11        fprintf(stderr,"Il file %s non è accessibile.\n",argv[1]);
12        return 2;
13    }
14
15    int s=0,x;
16    while(fscanf(f,"%d",&x)==1)
17        s+=x;
18
19    fclose(f);
20    printf("%d\n",s);
21    return 0;
22 }
```

4. (5 punti) Si consideri la seguente dichiarazione in C per rappresentare un elemento di una struttura a pila (stack):

```
1 struct stack {
2     int n;
3     struct stack *next;
4 };
```

Si scriva il codice necessario per implementare le operazioni push (per inserire in testa alla pila un nuovo elemento) e pop (per eliminare l'elemento di testa della pila, restituendo come valore di ritorno l'intero contenuto in quest'ultimo) del tipo di dati astratto stack.

Risposte:

```
1 struct stack *push(struct stack *p, int n) {
2     struct stack *q=malloc(sizeof (struct stack));
3     if(q) {
4         q->n=n;
5         q->next=p;
6     }
7     return q; // q punta alla nuova cima dello stack
8 }
9
10 int pop(struct stack **p) {
11     int r=-1;
12     struct stack *q=(*p);
13     if(*p) {
```

# Laboratorio di Sistemi Operativi

## 6 Luglio 2022

### Compito

```
14     (*p)=(*p)->next;
15     r=q->n;
16     free(q);
17 }
18 return r;
19 }
```

5. (10 punti) Si scriva un programma C che chieda all’utente quanti numeri interi pseudo-casuali vuole generare. Dopodiché il programma alloca memoria sufficiente per un array che possa contenere tali numeri ed inizia a generare un numero pseudo-casuale ogni 3 secondi. Alla fine stampa tutti i numeri generati, il minimo ed il massimo fra questi e termina. Tuttavia, prima di giungere a terminazione, se l’utente preme Ctrl-C (ovvero, preme contemporaneamente i tasti Ctrl e C), il programma stampa i numeri generati fino a quel momento ed il minimo ed il massimo fra questi.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
                     attuale , assicurandosi di generare sequenze diverse ad ogni
                     esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
              tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <signal.h>
5 #include <unistd.h>
6
7 int compute_min_max=0;
8
9 void catchint(int signo) {
10     compute_min_max=1;
11 }
12
13 void min_max(long int *min, long int *max, int index, long int *
               buffer) {
14     for(int i=0;i<index;i++) {
15         printf("%ld\n",buffer[i]);
16         if(*max== -1 || *max<buffer[i])
17             *max=buffer[i];
18         if(*min== -1 || *min>buffer[i])
19             *min=buffer[i];
20     }
21 }
22
23 int main() {
```

# Laboratorio di Sistemi Operativi

## 6 Luglio 2022

### Compito

```
24     int current_index, n;
25     long int *buf, max, min;
26     current_index=n=0;
27     buf=NULL;
28     max=min=-1;
29     srand(time(NULL));
30     signal(SIGINT, catchint);
31     printf("Inserisci il numero di interi da generare: ");
32     scanf("%d",&n);
33     buf=malloc(n*sizeof(long int));
34     while(current_index<n) {
35         if(compute_min_max) {
36             printf("Numeri casuali generati finora:\n");
37             min_max(&min, &max, current_index, buf);
38             printf("Minimo numero generato finora: %ld\n",min);
39             printf("Massimo numero generato finora: %ld\n",max);
40             compute_min_max=0;
41         }
42         else {
43             buf[current_index]=random();
44             current_index++;
45         }
46         sleep(3);
47     }
48
49     printf("Numeri casuali generati:\n");
50     min_max(&min, &max, n, buf);
51     printf("Minimo numero generato: %ld\n",min);
52     printf("Massimo numero generato: %ld\n",max);
53
54     free(buf);
55     return 0;
56 }
```

# Laboratorio di Sistemi Operativi

## 06 Settembre 2018

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Si scriva una pipeline per estrarre dal file `/etc/passwd` il terzo campo di ogni riga, ovvero, il numero intero che rappresenta lo user-ID dell'utente relativo a quella riga (si ricordi che il separatore di campo del file `/etc/passwd` è il carattere *due punti*) e si ordini i valori in modo *crescente*.

La pipeline è la seguente:

```
cat /etc/passwd | cut -d: -f3 | sort -n
```

2. (5 punti) Sfruttando l'idea alla base della soluzione dell'esercizio precedente, si scriva uno script della shell che produca in output soltanto gli user-ID contenuti nel file `/etc/passwd` con un valore maggiore dell'argomento passato allo script sulla linea di comando.

**Suggerimento:** si ricordi che `tail -n +k f` stampa le linee di `f`, a partire dalla `k`-esima, mentre `head -n +k f` stampa le prime `k` linee di `f`.

Nel seguente esempio di soluzione si utilizza l'opzione `-r` di `sort` per ordinare in modo decrescente gli user-ID. In tal modo non sarà necessario scorrere tutte le linee del file (non appena il valore corrente risulterà minore od uguale al parametro passato su linea di comando, si potrà uscire dal ciclo, terminando l'esecuzione dello script).

```
1 num_linee='wc -l < /etc/passwd'
2
3 if test $# -ne 1
4 then
5   echo "Utilizzo: $0 n"
6   exit 1
7 fi
8
9 if test $1 -lt -1
10 then
11   echo "Utilizzo: $0 n # n>=-1"
12   exit 2
13 fi
14
15 i=1
16 somma=0
17
18 while test $i -le $num_linee
19 do
20   linea='cat /etc/passwd | sort -n -r -t: -k3,3 | head -n +$i | tail
21     -1'
22   valore='echo $linea | cut -d: -f3'
23   if test $valore -gt $1
24   then
25     echo $valore
26   else
27     exit 0
28   fi
29   i=$((i+1))
30 done
31 exit 0
```

# Laboratorio di Sistemi Operativi

## 06 Settembre 2018

### Compito

3. (13 punti) Si scriva un programma C che:

- chieda all'utente di inserire il numero di righe e di colonne di una matrice quadrata;
- allochi dinamicamente lo spazio in memoria per la matrice (usando `malloc()`);
- chieda all'utente di inserire gli elementi della matrice allocata;
- stampi la matrice a video;
- calcoli e stampi a video il prodotto degli elementi della diagonale principale della matrice.

Esempio di input/output:

```
1 Inserisci il n. di righe e di colonne della matrice: 3
2
3 Inserisci gli elementi della matrice:
4 Inserisci l'elemento a11: 2
5 Inserisci l'elemento a12: 3
6 Inserisci l'elemento a13: 5
7 Inserisci l'elemento a21: 5
8 Inserisci l'elemento a22: 6
9 Inserisci l'elemento a23: 3
10 Inserisci l'elemento a31: 9
11 Inserisci l'elemento a32: 1
12 Inserisci l'elemento a33: 7
13
14 Matrice inserita:
15 2 3 5
16 5 6 3
17 9 1 7
18
19 Prodotto degli elementi della diagonale principale della matrice:
20 84
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int **a;
7     int r, p, i, j;
8     printf("Enter rows and columns of a square matrix: ");
9     scanf("%d", &r);
10
11    a=(int**)malloc(sizeof(int*)*r);
12
13    for(i=0; i<r; i++) {
14        a[i]=(int *)malloc(sizeof(int)*r);
15    }
16
17    // Storing elements of the matrix
18    printf("\nEnter elements of matrix:\n");
19    for(i=0; i<r; ++i)
20        for(j=0; j<r; ++j)
21        {
22            printf("Enter element a%d%d: ", i+1, j+1);
23            scanf("%d", &a[i][j]);
24        }
```

# Laboratorio di Sistemi Operativi

## 06 Settembre 2018

### Compito

```
25 // Displaying the matrix a[][] */
26 printf("\nEnterd Matrix: \n");
27 for(i=0; i<r; ++i)
28     for(j=0; j<r; ++j)
29     {
30         printf("%d ", a[i][j]);
31         if (j == r-1)
32             printf("\n");
33     }
34
35
36 p=1;
37 for(i=0; i<r; ++i) {
38     p=p*a[i][i];
39 }
40
41 printf("Prodotto degli elementi della diagonale principale
42      della matrice:\n%d\n",p);
43 return 0;
44 }
```

4. (4 punti) Si dica cosa stampa il seguente programma C (giustificando la risposta):

```
1 #include <stdio.h>
2
3 int main() {
4     int x;
5     int *p;
6
7     p=&x;
8     x=1;
9     printf("%d\n", (++x)*(*p));
10    *p=1;
11    printf("%d\n", (*p)*(++x));
12    return 0;
13 }
```

Il programma C stampa

```
4
2
```

Infatti, la prima espressione incrementa `x` a 2, prima di valutarne il valore (grazie all'operatore di preincremento `++`), e poi lo moltiplica per 2 in quanto `p` punta a `x`. La seconda espressione invece (dopo il reset di `x` a 1 per mezzo dell'assegnamento `*p=1`) valuta il contenuto della locazione di memoria puntata da `p`, ovvero, la locazione di `x`, ottenendo 1 per cui moltiplica il valore di `x`, dopo aver incrementato quest'ultimo a 2 (grazie, nuovamente, all'operatore di preincremento).

5. (8 punti) Il codice seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) e di sincronizzazione (condition variable), per implementare una soluzione al problema classico dei produttori e consumatori con memoria limitata. Ci sono `NUM_P` thread produttori e `NUM_C` thread consumatori, che accedono in modo concorrente al vettore condiviso `buffer` con `LENGTH` posizioni per altrettanti interi positivi (che assumono valori da 1 a `MAX`). Per convenzione il valore -1 indica

# Laboratorio di Sistemi Operativi

## 06 Settembre 2018

### Compito

che la posizione del vettore è libera (vuota). Un thread produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera. Un thread consumatore (se il buffer non è vuoto) preleva un elemento dalla prima posizione non vuota.

Supponendo di avere a disposizione

- la funzione `full()` che restituisce 1 se il buffer è pieno e 0 altrimenti,
- la funzione `empty()` che restituisce 1 se il buffer è vuoto e 0 altrimenti,

si completi il sorgente, specificando i comandi mancanti da inserire al posto dei ... negli 8 punti indicati, affinché il codice risultante rappresenti una soluzione corretta del problema dei produttori e consumatori con memoria limitata.

```
1 int buffer[LENGTH];           // buffer condiviso di lunghezza LENGTH
2 pthread_t threadP[NUM_P];    // vettore che contiene gli ID dei thread produttori
3 pthread_t threadC[NUM_C];    // vettore che contiene gli ID dei thread consumatori
4
5 // mutex per l'accesso esclusivo
6 pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
7 // condition variable: buffer non vuoto
8 pthread_cond_t not_empty_buffer=PTHREAD_COND_INITIALIZER;
9 // condition variable: buffer non pieno
10 pthread_cond_t not_full_buffer=PTHREAD_COND_INITIALIZER;
11
12 void *producer(void *n) {
13     int elemento, i;
14     while(1) {
15         printf("Thread produttore (ID %lu)\n",threadP[((int)n)-1]);
16         ... // <-- inizio sezione critica: completare (1)
17         ... // <-- controllo se posso inserire un nuovo elemento: completare (2)
18         for(i=0; i<LENGTH; i++)
19             if(buffer[i]==-1) {
20                 elemento=random()%MAX+1; // genero l'elemento
21                 buffer[i]=elemento;      // inserisco l'elemento
22                 break;
23             }
24         ... // <-- quale evento devo segnalare qui? completare (3)
25         ... // <-- fine sezione critica: completare (4)
26     }
27 };
28
29 void *consumer(void *n) {
30     int elemento, i;
31     while(1) {
32         printf("Thread consumatore (ID %lu)\n",threadC[((int)n)-1]);
33         ... // <-- inizio sezione critica: completare (5)
34         ... // <-- controllo se posso prelevare un elemento (6)
35         for(i=0; i<LENGTH; i++)
36             if(buffer[i]!=-1) {
37                 elemento=buffer[i]; // prelevo l'elemento
38                 buffer[i]=-1;       // segnalo che la posizione è libera
39                 break;
40             }
41         ... // <-- quale evento devo segnalare qui? completare (7)
42         ... // <-- fine sezione critica: completare (8)
43     }
44 };
```

1. `pthread_mutex_lock(&mutex);`

**Laboratorio di Sistemi Operativi  
06 Settembre 2018  
Compito**

```
2. if(full()) pthread_cond_wait(&not_full_buffer,&mutex);  
3. pthread_cond_signal(&not_empty_buffer);  
4. pthread_mutex_unlock(&mutex);  
5. pthread_mutex_lock(&mutex);  
6. if(empty()) pthread_cond_wait(&not_empty_buffer,&mutex);  
7. pthread_cond_signal(&not_full_buffer);  
8. pthread_mutex_unlock(&mutex);
```

# Laboratorio di Sistemi Operativi

## 9 Settembre 2021

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `del_files.sh` che prenda come argomento sulla linea di comando una stringa ed un percorso, controlli che quest'ultimo corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, cancellando tutti i file incontrati che abbiano come estensione la stringa fornita come primo argomento. Durante la cancellazione deve stampare a video il percorso dei file cancellati e, alla fine, deve stampare il numero totale di file rimossi.

Esempio:

```
./del_files.sh bak .
./a.bak
./b/c.bak
Numero di file cancellati: 2
```

Esempio di soluzione:

```
1 if ! test $# -eq 2
2 then
3     echo "Utilizzo $0 extension pathname"
4     exit 1
5 fi
6
7 num_deleted_file=0
8 files='find $2 -name "*.$1"'
9
10 for f in $files
11 do
12     if test -f $f
13     then
14         echo $f
15         rm -f $f
16         num_deleted_file=$((num_deleted_file+1))
17     fi
18 done
19
20 echo "Numero di file cancellati: $num_deleted_file"
21
22 exit 0
```

2. (6 punti) Si supponga che sia stata impostata una variabile di ambiente di nome `utente`. Si scriva un unico comando o pipeline, utilizzando i vari metacaratteri di composizione di comandi della shell, che stampi a video la stringa `ok` (e niente altro) se esiste un utente del sistema con nome di login uguale al valore della variabile `utente`. In caso contrario, il comando o pipeline non deve stampare nulla.

**Suggerimento:** si ricorra al file `/etc/passwd` dove ogni linea corrisponde ad un account del sistema ed è una successione di campi separati dai due punti (:). Il campo relativo al nome di login è il primo.

Esempio di soluzione:

```
1 cat /etc/passwd | cut -d: -f1 | grep "$utente" /etc/passwd 2>&1 >/dev/null && echo ok
```

**Laboratorio di Sistemi Operativi**  
**9 Settembre 2021**  
**Compito**

3. (8 punti) Scrivere il codice di un programma C che prenda come argomento sulla linea di comando il percorso di un file di testo e stampi a video il suo contenuto, applicando il seguente filtro: devono essere rimossi dal testo tutti i caratteri eccetto le lettere dell'alfabeto (maiuscole e minuscole). Si gestiscano inoltre gli eventuali errori (numero di argomenti errato, file non leggibile, ecc.).

```
#include <stdio.h>

#define SIZE 150

void filter(char *line) {
    int i, j;

    for(i = 0; line[i] != '\0'; ++i)
    {
        while (!( (line[i] >= 'a' && line[i] <= 'z') ||
                  (line[i] >= 'A' && line[i] <= 'Z') ||
                  (line[i] == '\0'))
               ))
        {
            for(j = i; line[j] != '\0'; ++j)
            {
                line[j] = line[j+1];
            }
            line[j] = '\0';
        }
    }

    puts(line);
}

int main(int argc, char **argv) {
    char line[SIZE];
    int i, j;

    if(argc!=2) {
        fprintf(stderr, "Usage: %s file-path\n", argv[0]);
        return 1;
    }

    FILE *f=fopen(argv[1],"r");

    if(f==NULL) {
        fprintf(stderr, "file %s not found!\n", argv[1]);
        return 2;
    }

    while(fgets(line,SIZE,f)!=NULL) {
        filter(line);
    }

    fclose(f);

    return 0;
}
```

# Laboratorio di Sistemi Operativi

## 9 Settembre 2021

### Compito

4. (4 punti) Si considerino le seguenti dichiarazioni in C:

```
1 struct lista {
2     int n;
3     struct lista *next;
4 };
5 struct binary_tree {
6     int n;
7     struct binary_tree *left;
8     struct binary_tree *right;
9 };
10 struct lista *l;
11 struct binary_tree *t;
12 struct lista l2={5, NULL};
13 struct binary_tree t2={1, NULL, NULL};
```

Si dica, per ognuna delle seguenti sequenze di comandi, se è corretta oppure no (motivando la risposta e correggendo gli eventuali errori):

Sequenza 1:

```
1 l=0;
2 t=0;
```

Sequenza 2:

```
1 t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
2 if(t!=NULL) {
3     t->n=l2.n;
4     t->left=t->right=0;
5 }
```

Sequenza 3:

```
1 l=(struct lista *)malloc(sizeof(struct lista));
2 if(l!=NULL) {
3     (*l).n=l2.n+1;
4     l->next=l2;
5 }
```

Sequenza 4:

```
1 t2.n=l2.n+1;
```

Risposte:

1. sequenza corretta: viene assegnata la costante 0 ad entrambi i puntatori **l** e **t** che così denotano, rispettivamente, la lista vuota e l'albero binario vuoto (senza nodi);
2. sequenza corretta: viene allocata memoria per un nodo di un albero binario, assegnando al puntatore **t** il suo indirizzo; in seguito viene assegnato l'intero contenuto nel campo **n** del nodo **l2** al campo **n** del nodo appena allocato (i campi **left** e **right** si vedono assegnare la costante 0, ad indicare che il nodo è privo di figli);
3. sequenza scorretta: a **l->next** bisogna assegnare un puntatore di tipo **struct lista \***.  
Correzione:

```
1 l=(struct lista *)malloc(sizeof(struct lista));
2 if(l!=NULL) {
3     (*l).n=l2.n+1;
4     l->next=&l2;
5 }
```

4. sequenza corretta: all'intero **t2.n** viene assegnato il successore dell'intero **l2.n**.

# Laboratorio di Sistemi Operativi

## 9 Settembre 2021

### Compito

5. (10 punti) Si scriva un programma C `dice.c` che simuli il lancio di un dado. Il programma prende da linea di comando il numero `n` (maggiore o uguale a 1) di thread che deve lanciare in esecuzione. Ogni thread inizia a generare un numero pseudo-casuale compreso tra 1 e 6 ogni 5 secondi. Il primo thread che genera il punteggio massimo provoca la terminazione di tutto il processo. Prima di terminare, il processo salva nel file `log.txt` il messaggio `Punteggio massimo raggiunto dal thread con ID: x.`, dove `x` è l'ID del thread che ha generato il punteggio massimo.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
    attuale , assicurandosi di generare sequenze diverse ad ogni
    esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
    tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di esecuzione:

```
1 $ ./dice 8
2 2 (139833826105088)
3 4 (139833809319680)
4 5 (139833817712384)
5 2 (139833724888832)
6 6 (139833716496128)
7 $ cat log.txt
8 Punteggio massimo raggiunto dal thread con ID: 139833716496128.
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5 #include <unistd.h>
6 #include <pthread.h>
7
8 int stop=0;
9 pthread_t winner;
10 pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
11
12 void *dice(void *ptr) {
13     while(!stop) {
14         long int r=random();
15         int score=r%6+1;
16         printf("%d (%lu)\n",score,*((pthread_t *)ptr));
17         if(score==6) {
18             pthread_mutex_lock(&mutex);
19             stop=1;
20             winner=*((pthread_t *)ptr);
21             pthread_mutex_unlock(&mutex);
22         }
23         sleep(5);
```

**Laboratorio di Sistemi Operativi**  
**9 Settembre 2021**  
Compito

```
24     }
25 }
26
27 int main(int argc, char** argv) {
28     if(argc!=2) {
29         fprintf(stderr,"Uso: %s n\n",argv[0]);
30         return 1;
31     }
32
33     int num_thread=atoi(argv[1]);
34     winner=0;
35     pthread_t *thread_id=NULL;
36
37     if(num_thread>=1) {
38         thread_id=(pthread_t*)malloc(sizeof(pthread_t)*num_thread);
39
40         if(thread_id!=NULL) {
41             for(int i=0; i<num_thread; i++) {
42                 if(pthread_create(&thread_id[i],NULL,dice,&thread_id[i])
43                     !=0) {
44                     fprintf(stderr,"Errore nella creazione del thread n. %d
45                         .\n",i+1);
46                     exit(1);
47                 }
48             }
49         } else {
50             perror("Memoria insufficiente.\n");
51             exit(1);
52         }
53
54         for(int i=0; i<num_thread; i++)
55             pthread_join(thread_id[i],NULL);
56
57         FILE* log_file=fopen("log.txt","w");
58         char buffer[80];
59         sprintf(buffer,"Punteggio massimo raggiunto dal thread con ID:
60             %lu.\n",winner);
61         fwrite(buffer,sizeof(char),strlen(buffer),log_file);
62         fclose(log_file);
63         free(thread_id);
64     }
65
66     return 0;
67 }
```

# Laboratorio di Sistemi Operativi

## 7 luglio 2016

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Si specifichi il comando per creare un link **hard** di nome `link1` nella directory corrente al file `/home/pippo/documento.txt`
- (b) Si specifichi il comando per creare un link **simbolico** di nome `link2` nella directory corrente al file `/home/pippo/documento.txt`
- (c) A livello dell'implementazione nel filesystem, c'è differenza fra `link1` e `link2` creati nei due punti precedenti? Se sì, quale?
- (d) È possibile creare un link simbolico ad un file risiedente su un dispositivo fisico diverso da quello in cui risiede il link? Ed un link hard? Motivare le risposte.

#### Soluzione:

- (a) `ln /home/pippo/documento.txt link1`
- (b) `ln -s /home/pippo/documento.txt link2`
- (c) Sì, c'è differenza fra `link1` e `link2` creati nei due punti precedenti: nel primo caso infatti si tratta di un alias per lo stesso numero di inode, ovvero, sia `documento.txt` che `link1` puntano allo stesso elemento dello stesso vettore di inode. Invece nel secondo caso `link2` punta ad un numero di inode diverso rispetto a quello puntato da `documento.txt`; infatti `link2` altro non è che un file di testo (trattato in modo speciale dal sistema operativo) contenente il pathname del file `documento.txt`.
- (d) Sì, è possibile creare un link simbolico ad un file risiedente su un dispositivo fisico diverso da quello in cui risiede il link in quanto, come detto precedentemente, un link simbolico è un file di testo contenente un pathname. Quindi è sufficiente seguire il percorso per arrivare al file puntato. Al contrario non è possibile creare un link hard ad un file risiedente su un dispositivo fisico diverso da quello in cui risiede il link in quanto ogni dispositivo fisico ha un proprio vettore di inode (e non è possibile puntare ad inode di un dispositivo diverso da quello di origine).

2. Si scrivano i comandi necessari per risolvere ognuno dei task seguenti:

- (a) fornire il conteggio dei processi dell'utente `pippo`;
- (b) fornire la dimensione in byte (caratteri) del file `/etc/passwd` e memorizzarla nella variabile `len`;
- (c) estrarre il primo (user name) ed il terzo (user ID) campo dal file `/etc/passwd` e salvarli nel file `estratto.txt` in modo che i campi corrispondenti su ogni linea siano separati dal carattere underscore (`_`). Quindi l'output finale deve essere del tipo seguente:

```
root_0  
daemon_1  
bin_2  
...  
...
```

#### Soluzione:

- (a) `ps -u pippo --no-headers | wc -l`
- (b) `len='wc -c < /etc/passwd'`
- (c) `cat /etc/passwd | cut -d: -f1 > f1.txt`  
`2 cat /etc/passwd | cut -d: -f3 > f3.txt`  
`3 paste -d_ f1.txt f3.txt > estratto.txt`

3. Si scriva il codice di uno **script della shell** che prenda in input come parametro della linea di comando il percorso di un file e ne stampi a video la linea più lunga.

Ad esempio (se `longest` è il nome dello script eseguibile):

# Laboratorio di Sistemi Operativi

## 7 luglio 2016

### Compito

```
> ./longest /etc/passwd
```

deve produrre in output la linea più lunga del file `/etc/passwd`. Si gestiscano gli eventuali errori (numero di argomenti errato, file non leggibile).

#### Soluzione:

Esempio di soluzione:

```
1 if test $# -ne 1
2 then
3     echo "utilizzo : $0 <file>"
4     exit 1
5 fi
6
7 if ! test -e $1 -a -r $1
8 then
9     echo "il file $1 non è accessibile"
10    exit 2
11 fi
12
13 linee='wc -l < $1'
14 i=0
15 max=0
16 lunga=
17
18 while test $i -lt $linee
19 do
20     linea_corrente='cat $1 | tail -n +$i | head -1'
21     num_car='echo $linea_corrente | wc -c'
22     if test $num_car -gt $max
23     then
24         max=$num_car
25         lunga=$linea_corrente
26     fi
27     i=$((i+1))
28 done
29
30 echo $lunga
```

4. Si consideri il seguente programma `logger_server.c` (per semplicità sono state omesse le direttive di inclusione) che implementa un server multithread che accoda nel file `log.txt` i messaggi inviati dai client a lui connessi. Ogni connessione viene gestita da un nuovo thread che esegue la funzione `logger()`, dove avviene la ricezione del messaggio ed il suo accodamento nel file `log.txt`.

```
1 #define SERVER_PORT 8888 /* porta di ascolto del server */
2 #define LINESIZE 255      /* dimensione dei buffer */
3
4 struct channel {
5     int fd; /* file descriptor del canale di comunicazione */
6     int num; /* numero intero identificante il thread */
7 };
8
9 void *logger(void *c) {
10     char inputline[LINESIZE];
11     char buffer[2*LINESIZE];
12     int len;
13     int fd=((struct channel *)c)->fd; /* recupero il socket file descriptor */
14     int num=((struct channel *)c)->num; /* recupero il numero del thread */
15     ... /* se necessario, è possibile aggiungere altre variabili */
16
17     while ((len = recv(fd, inputline, LINESIZE-1, 0)) > 0) {
18         ... /* completare (1) */
19     }
```

# Laboratorio di Sistemi Operativi

## 7 luglio 2016

### Compito

```

20     close(fd); /* chiudo la connessione con il client */
21 }
22
23 int main() {
24     int sock, client_len, fd, i=0;
25     struct sockaddr_in server, client;
26     pthread_t t;
27     struct channel ch;
28
29     if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
30         perror("chiamata alla system call socket fallita");
31         exit(1);
32     }
33     server.sin_family = ...; /* completare (2) */
34     server.sin_addr.s_addr = ...; /* completare (3) */
35     server.sin_port = ...; /* completare (4) */
36
37     /* binding dell'indirizzo al transport end point */
38     if (bind(...) == -1) { /* completare (5) */
39         perror("chiamata alla system call bind fallita");
40         exit(2);
41     }
42
43     /* impostiamo il server in modo che possa gestire 15 richieste
44      contemporaneamente */
45     ...; /* completare (6) */
46
47     while (1) {
48         client_len = sizeof(client);
49         if ((fd = accept(sock, (struct sockaddr *)&client, &client_len)) < 0) {
50             perror("accepting connection");
51             exit(3);
52         }
53
54         /* ogni volta che il server accetta una nuova connessione,
55          quest'ultima viene gestita da un nuovo thread
56          */
57         ch.fd=fd;
58         ch.num=++i;
59
60         if(pthread_create(&t, NULL, logger, (void *)&ch)!=0) {
61             fprintf(stderr, "Errore nella creazione del thread n. %d!", i);
62             return 1;
63         }
64     }
65 }

```

- (a) Si completi il codice relativo alle system call delle socket (dove compaiono i puntini).
- (b) Si completi il codice della funzione `logger()` in modo che la linea accodata nel file `log.txt` abbia il formato seguente:
- ```
Thread n. i: <stringa>
```
- dove `i` è l'intero identificante il thread (memorizzato nel campo `num` di `struct channel`) e `<stringa>` è il messaggio ricevuto dal client (si faccia attenzione al fatto che `recv` non mette nel buffer in cui scrive il carattere terminatore: va messo a mano).
- (c) Una volta completato il punto precedente, si modifichi la funzione `logger()` ed eventualmente il programma in generale (dove necessario) per garantire che il server non registri in modo disordinato i messaggi provenienti dai client, ma faccia in modo che le linee di ogni client appaiano contigue. Per esempio, nel caso di tre client connessi, il file `log.txt` non deve contenere una sequenza del genere:

```
Thread n. 1: abc
Thread n. 2: bla bla bla
Thread n. 1: def
Thread n. 3: 1111
Thread n. 1: ghi
thread n. 3: 2222
```

Deve invece fare in modo che la successione sia la seguente:

# Laboratorio di Sistemi Operativi

## 7 luglio 2016

### Compito

```
Thread n. 1: abc
Thread n. 1: def
Thread n. 1: ghi
Thread n. 2: bla bla bla
Thread n. 3: 1111
thread n. 3: 2222
```

facendo eventualmente attendere gli altri thread, mentre il thread correntemente in esecuzione sta scrivendo nel file.

#### Soluzione:

- (a) Codice relativo alle system call delle socket:

```
Linea 33: server.sin_family = AF_INET;
Linea 34: server.sin_addr.s_addr = inet_addr("127.0.0.1");
oppure
server.sin_addr.s_addr = INADDR_ANY;
Linea 35: server.sin_port = htons(SERVER_PORT);
Linea 38: if (bind(sock, (struct sockaddr *)&server, sizeof server) == -1){
Linea 45: listen (sock, 15);
```

- (b) Codice della funzione logger:

```
1 void *logger (void *c) {
2     char inputline [LINESIZE];
3     char buffer [2*LINESIZE];
4     int len;
5     int fd=((struct channel *)c)->fd ;
6     int num=((struct channel *)c)->num;
7     FILE *logfd ;
8
9     while ((len = recv(fd , inputline , LINESIZE-1, 0)) > 0) {
10         inputline [len]='\0';
11         sprintf(buffer , "Thread %d: " ,num);
12         strcat(buffer , inputline);
13         logfd=fopen ("log.txt" , "a"); /* apro il file log.txt in append */
14         fputs(buffer , logfd); /* scrivo nel file */
15         fclose(logfd); /* chiudo il file log.txt */
16     }
17
18     close(fd);
19 }
```

- (c) Vi sono vari modi di risolvere il problema. Ad esempio si può fare in modo che il padre attenda la terminazione di ogni thread (mediante una `pthread_join()`) prima di creare il successivo. Oppure è possibile aggiungere un mutex, dichiarandolo ed inizializzandolo dopo le direttive `#define`:

```
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
ed utilizzarlo per realizzare una sezione critica all'interno della funzione logger():
```

```
1 void *logger (void *c) {
2     char inputline [LINESIZE];
3     char buffer [2*LINESIZE];
4     int len;
5     int fd=((struct channel *)c)->fd ;
6     int num=((struct channel *)c)->num;
7     FILE *logfd ;
8
9     pthread_mutex_lock(&mutex); /* inizio sezione critica */
```

# Laboratorio di Sistemi Operativi

## 7 luglio 2016

### Compito

```
10
11  while ((len = recv(fd , inputline , LINESIZE-1, 0)) > 0) {
12      inputline [len]='\0';
13      sprintf(buffer , "Thread\u2022n.\u2022%u:\u2022" ,num);
14      strcat(buffer ,inputline);
15      logfd=fopen( " log .txt " , "a" );
16      fputs(buffer ,logfd );
17      fclose(logfd );
18  }
19
20  close(fd );
21  pthread_mutex_unlock(&mutex);          /* fine sezione critica */
22 }
```

# Laboratorio di Sistemi Operativi

## 8 Luglio 2021

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `check_file_tree.sh` che prenda come argomento sulla linea di comando un percorso, controlli che corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, stampando il percorso dei soli file (quindi non delle sottodirectory) incontrati, la relativa dimensione in byte e, alla fine, il totale dei byte occupati da essi.

Esempio:

```
./check_file_tree.sh .
./check_file_tree.sh, 374 byte
./draw.c, 197 byte
./test, 16864 byte
./dice.c, 1031 byte
./test.c, 738 byte
./draw, 16696 byte
./dice, 17272 byte
Numero totale di byte: 53172
```

Esempio di soluzione:

```
1 if ! test $# -eq 1
2 then
3     echo "Utilizzo $0 pathname"
4     exit 1
5 fi
6
7 num_bytes=0
8
9 if test -e $1 -a -d $1
10 then
11     lista='find $1 -print 2>/dev/null'
12     for i in $lista
13     do
14         if test -f $i -a ! -l $i
15         then
16             dim=$(cat $i | wc -c)
17             echo $i, $dim byte
18             num_bytes=$[num_bytes+$dim]
19         fi
20     done
21 fi
22
23 echo "Numero totale di byte: $num_bytes"
24
25 exit 0
```

2. (6 punti) Si scriva una pipeline che stampi a video l'elenco (senza ripetizioni e ordinato lessicograficamente *al contrario*) dei nomi di login degli utenti di sistema.

**Suggerimento:** si ricorra al file `/etc/passwd` dove ogni linea corrisponde ad un account del sistema ed è una successione di campi separati dai due punti (:). Il campo relativo al nome di login è il primo.

# Laboratorio di Sistemi Operativi

## 8 Luglio 2021

### Compito

Esempio di soluzione:

```
1 cat /etc/passwd | cut -d ':' -f1 | sort -r | uniq
```

3. (6 punti) Si considerino le seguenti dichiarazioni in C:

```
1 struct lista {
2     int n;
3     struct lista *next;
4 };
5 struct binary_tree {
6     int n;
7     struct binary_tree *left;
8     struct binary_tree *right;
9 };
10 struct lista *l;
11 struct binary_tree *t;
12 struct lista l2={5, NULL};
13 struct binary_tree t2={1, NULL, NULL};
14 int a[10]={1,1,1,1,1,1,1,1,1,1};
```

Si dica, per ognuna delle seguenti sequenze di comandi, se è corretta oppure no (motivando la risposta e correggendo gli eventuali errori):

Sequenza 1:

```
1 l=NULL;
2 t=NULL;
```

Sequenza 2:

```
1 t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
2 if(t!=NULL) {
3     t->n=a[5];
4     t->left=t->right=NULL;
5 }
```

Sequenza 3:

```
1 a[1]=l2->n;
```

Sequenza 4:

```
1 l=(struct lista *)malloc(sizeof(struct lista));
2 if(l!=NULL) {
3     (*l).n=l2.n+1;
4     l->next=l2;
5 }
```

Sequenza 5:

```
1 t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
2 if(t!=NULL) {
3     t->n=a[5];
4     t->left=&t2;
5     t->right=&l2;
6 }
```

Sequenza 6:

```
1 t2.n=l2.n+1;
```

# Laboratorio di Sistemi Operativi

## 8 Luglio 2021

### Compito

Risposte:

1. sequenza corretta: viene assegnata la costante NULL ad entrambi i puntatori `l` e `t` che così denotano, rispettivamente, la lista vuota e l'albero binario vuoto (senza nodi);
2. sequenza corretta: viene allocata memoria per un nodo di un albero binario, assegnando al puntatore `t` il suo indirizzo; in seguito viene assegnato l'intero contenuto nella sesta posizione dell'array `a` al campo `n` del nodo appena allocato (i campi `left` e `right` si vedono assegnare la costante NULL, ad indicare che il nodo è privo di figli);
3. sequenza scorretta: l'operatore `->` si applica solo ai puntatori a strutture (correzione: `a[1]=l2.n;`);
4. sequenza scorretta: a `l->next` bisogna assegnare un puntatore di tipo `struct lista *`.  
Correzione:

```
1 l=(struct lista *)malloc(sizeof(struct lista));
2 if(l!=NULL) {
3     (*l).n=l2.n+1;
4     l->next=&l2;
5 }
```

5. sequenza scorretta: il tipo di `&l2` è `struct lista *`, mentre dovrebbe essere `struct binary_tree *`. Correzione:

```
1 t=(struct binary_tree *)malloc(sizeof(struct binary_tree));
2 if(t!=NULL) {
3     t->n=a[5];
4     t->left=&t2;
5     t->right=NULL; // oppure: t->right=&t2;
6 }
```

6. sequenza corretta: all'intero `t2.n` viene assegnato il successore dell'intero `l2.n`.

4. (4 punti) Si dica qual è l'output generato dal seguente programma C:

```
1 #include <stdio.h>
2
3 int main() {
4     int i,j;
5     for(i=2; i>=0; i--) {
6         for(j=0; j<i; j++)
7             printf(" ");
8         for(j=0; j<(5-i)*2-1; j++)
9             printf("*");
10        printf("\n");
11    }
12
13    return 0;
14 }
```

```
*****
*****
*****
```

# Laboratorio di Sistemi Operativi

## 8 Luglio 2021

### Compito

5. (10 punti) Si scriva un programma C `dice.c` che simuli il lancio di un dado. Il programma prende da linea di comando il numero `n` (maggiore o uguale a 1) di thread che deve lanciare in esecuzione. Ogni thread inizia a generare un numero pseudo-casuale compreso tra 1 e 6 ogni 5 secondi. Il primo thread che genera il punteggio massimo provoca la terminazione di tutto il processo, stampando a video il messaggio `E' stato raggiunto il punteggio massimo!`.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
    attuale, assicurandosi di generare sequenze diverse ad ogni
    esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
    tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <unistd.h>
5 #include <pthread.h>
6
7 int stop=0;
8 pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
9
10 void *dice(void *ptr) {
11     while(!stop) {
12         long int r=random();
13         int score=r%6+1;
14
15         if(score==6) {
16             pthread_mutex_lock(&mutex);
17             printf("Punteggio massimo raggiunto.\n");
18             stop=1;
19             pthread_mutex_unlock(&mutex);
20         }
21
22         sleep(5);
23     }
24 }
25
26 int main(int argc, char** argv) {
27     if(argc!=2) {
28         fprintf(stderr, "Uso: %s n\n", argv[0]);
29         return 1;
30     }
31     int num_thread=atoi(argv[1]);
32     pthread_t *thread_id=NULL;
33
34     if(num_thread>=1) {
35         thread_id=(pthread_t*)malloc(sizeof(pthread_t)*num_thread);
```

# Laboratorio di Sistemi Operativi

## 8 Luglio 2021

### Compito

```
36
37     if(thread_id!=NULL) {
38         for(int i=0; i<num_thread; i++) {
39             if(pthread_create(&thread_id[i],NULL,dice,NULL)!=0) {
40                 fprintf(stderr,"Errore nella creazione del thread n. %d
41                         .\n",i+1);
42                 exit(1);
43             }
44         } else {
45             perror("Memoria insufficiente.\n");
46             exit(1);
47         }
48
49         for(int i=0; i<num_thread; i++)
50             pthread_join(thread_id[i],NULL);
51         free(thread_id);
52     }
53
54     return 0;
55 }
```

# Laboratorio di Sistemi Operativi

## 08 settembre 2014

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. L'utente pippo esegue il comando `ls -l` nella sua home directory, ottenendo il seguente output:

```
...
drw-r--r--    2 pippo  users   68 Sep  2 11:42 tmp
...
```

Successivamente tenta di eseguire il comando `cd tmp`, ma questo fallisce, riportando il messaggio d'errore `-bash: cd: tmp: Permission denied`.

- Perché il comando `cd tmp` fallisce?
- Cosa deve fare l'utente per riuscire a spostarsi nella directory `tmp`?

#### Soluzione:

- L'errore è dovuto al fatto che anche per l'owner (l'utente `pippo`) manca il permesso di esecuzione.
- Per rimediare è sufficiente digitare (ad esempio) il seguente comando:

```
chmod u+x tmp
```

2. Si completi il seguente script della shell `bash` che deve leggere dallo standard input un flusso di testo (linea per linea), producendo in output un istogramma di asteriski (\*), in modo che la lunghezza di ogni linea di quest'ultimo rappresenti il numero di caratteri contenuto nella linea corrispondente del testo passato in input. Ad esempio, se il file di testo `prova.txt` contiene 4 linee di 7, 2, 1, 5 caratteri rispettivamente, l'istogramma prodotto deve essere il seguente:

```
*****
**
*
*****
```

Il codice da completare è il seguente:

```
IFS='\'n\' # imposta il separatore di linea per il comando read
while read -r linea # -r considera eventuali backslash come normali caratteri
do
  ...
done
```

#### Soluzione:

Esempio di soluzione:

```
IFS='\'n\''
while read -r linea
do
  lunghezza='echo $linea | wc -c'
  #echo $lunghezza
  i=0
  while test $i -lt $lunghezza
  do
    echo -n '*'
    i=$[i+1]
  done
  echo
done
```

# Laboratorio di Sistemi Operativi

## 08 settembre 2014

### Compito

3. Si mostri qual è l'output prodotto dai seguenti comandi:

- (a) echo ababab | sed '1,\$y/ab/yz/'
- (b) echo ababab | sed '1,\$s/ab/\_/'
- (c) echo ababab | sed '1,\$s/ab/\_/g'

#### Soluzione:

- (a) yzyzzyz
- (b) \_abab
- (c) ---

4. Si progetti uno script della shell `rect.sh` che prenda in input sulla linea di comando due interi positivi e stampi un rettangolo di asterischi secondo lo schema dell'esempio seguente (prendendo quindi le misure dei lati come argomenti sulla linea di comando):

```
$ ./rect.sh 5 3
*****
*   *
*****
```

#### Soluzione:

Esempio di soluzione:

```
if test $# -ne 2
then
    echo 'Utilizzo dello script: rect.sh <m> <n>'
    exit 1
fi

if ! test $1 -gt 0 -a $2 -gt 0
then
    echo 'Le dimensioni del rettangolo devono essere degli interi >0'
    exit 2
fi

x=$1
y=$2

while test $y -gt 0
do
    while test $x -gt 0
    do
        if test $y -gt 1 -a $y -lt $2
        then
            if test $x -gt 1 -a $x -lt $1
            then
                echo -n " "
            else
                echo -n "*"
            fi
        else
            echo -n "\n"
        fi
    done
done
```

# Laboratorio di Sistemi Operativi

## 08 settembre 2014

### Compito

```
echo -n "*"
fi
x=${x-1}
done
x=$1
y=${y-1}
echo
done

exit 0
```

5. Classificare come vere o false le seguenti affermazioni (per quelle false giustificare le risposte):
- (a) Nel linguaggio C si definisce una struttura ricorsiva, specificando un membro di tipo puntatore, che fa riferimento ad una struttura dello stesso tipo di quella in cui è contenuto.
  - (b) La system call `fork()` viene usata per generare *nuovi thread* all'interno del processo che la invoca.
  - (c) Le system call della famiglia `exec` consentono di lanciare in esecuzione un nuovo processo a partire da un programma esterno, dopodiché il processo originale (ovvero, colui che ha invocato la system call di tipo `exec`) continua la propria esecuzione.
  - (d) Una `pthread_cond_signal()` va eseguita soltanto se c'è un thread in stato di attesa (ovvero, che ha eseguito una `pthread_cond_wait()`) sulla corrispondente condition variable. Altrimenti la segnalazione viene "persa", dato che non c'è nessun thread in attesa di essa.
  - (e) Le socket possono essere utilizzate come un meccanismo di comunicazione tra processi (sia che questi risiedano sulla stessa macchina che su macchine diverse connesse in rete).
  - (f) La comunicazione stabilita tramite socket è di tipo bidirezionale.
  - (g) Una socket in uso è solitamente legata ad un indirizzo e la natura di quest'ultimo dipende dal dominio di comunicazione della socket.

#### Soluzione:

- (a) Vero.
- (b) Falso. La system call `fork()` crea un nuovo processo, copiando il chiamante (padre).
- (c) Falso. Questa famiglia di system call sovrascrive il processo chiamante, sostituendone il codice con quello del programma esterno.
- (d) Vero.
- (e) Vero.
- (f) Vero.
- (g) Vero.

6. Scrivere un programma C che esegua ad intervalli di 5 secondi il comando `ps -ef`. Il programma deve continuare la sua esecuzione fintanto che l'utente non prema la combinazione di tasti CTRL-C (pressione contemporanea di CTRL e C). A quel punto prima di terminare l'esecuzione il programma deve visualizzare su standard output il numero di cicli compiuti.

#### Soluzione:

Esempio di soluzione:

**Laboratorio di Sistemi Operativi**  
**08 settembre 2014**  
Compito

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

int stop;

/* prototipo della funzione per la gestione del segnale SIGINT */
void catchint(int);

int main() {
    pid_t pid;
    int count=0;
    stop=0;
    static struct sigaction act;
    /* registrazione dell'handler */
    act.sa_handler = catchint;
    /* eventuali altri segnali saranno ignorati
       durante l'esecuzione dell'handler */
    sigfillset(&(act.sa_mask));
    sigaction(SIGINT, &act, NULL);
    while(!stop) {
        sleep(5);
        switch(pid=fork()) {
            case 0:
                execl("/bin/ps","ps","-ef",(char *)0);
                perror("Exec fallita!");
                break;
            case -1:
                perror("Fork fallita!");
                return 1;
            default:
                waitpid(pid,NULL,0);
                count++;
        }
    }
    printf("Terminazione; sono stati eseguiti %d cicli\n",count);
    return 0;
}

void catchint(int signo) {
    stop=1;
}
```

# Laboratorio di Sistemi Operativi

## 9 febbraio 2016

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

- Il comando `date` produce in output la data ed ora attuali: ad esempio,

`Tue 9 Feb 10:23:04 CET 2016`

Si scriva uno script che produca in output una stringa nel formato

`giorno/mese/anno - ore_minuti_secondi` (ad esempio: `9/Feb/2016 - 10_23_04`).

#### Soluzione:

Esempio di soluzione:

```
1 #!/bin/bash
2 dataora='date | tr -s " "
3 giorno='echo $dataora | cut -d ":" -f2'
4 mese='echo $dataora | cut -d ":" -f3'
5 anno='echo $dataora | cut -d ":" -f6'
6 ora='echo $dataora | cut -d ":" -f4 | tr ":" "_"'
7 echo "$giorno/$mese/$anno$ora"
```

- Qual è l'effetto dei seguenti comandi?

- `ls -l | grep '^d...r.x'`
- `n='wc registro.txt | sed 's/ /:/g' | cut -d':' -f4'`  
si supponga che l'output di `wc registro.txt` sia la stringa '`14 27 415 registro.txt`' (senza apici, ma con gli spazi evidenziati);
- `echo $n` (dove `n` è la variabile inizializzata nel punto precedente).

**Attenzione:** nel punto 2 gli apici esterni sono dei *backquote* (apici rovesciati).

#### Soluzione:

- L'output di questa pipeline consiste nella visualizzazione in long format delle subdirectory della directory corrente che sono leggibili ed attraversabili dagli utenti del gruppo. Infatti il pattern usato con il comando `grep` seleziona le linee prodotte da `ls -l` che iniziano con una `d` (directory) e che hanno i permessi `r` e `x` del gruppo attivi.
- L'output prodotto dal primo comando della pipeline ('`14 27 415 registro.txt`') viene passato al comando `sed` che sostituisce tutti gli spazi con dei due punti (:), producendo la stringa '`:14::27:415: registro.txt`'. Quindi il comando `cut` seleziona il quarto campo (usando i due punti come separatore) producendo come output finale della pipeline 27. Questo è quanto viene assegnato alla variabile `n`, dato che tutta la pipeline si trova all'interno dei backquote, inducendo la shell a compiere una *command substitution*.
- viene stampato il valore della variabile `n`, ovvero, 27.

- Sia data la seguente struttura ricorsiva in C per la rappresentazione di elementi di una coda di interi:

```
struct int_queue {
    int val;
    struct int_queue *next;
};
```

dove `val` rappresenta il valore dell'elemento della lista, mentre `next` punta al prossimo elemento della lista (se l'elemento in questione è l'ultimo, allora `next` punta a `NULL`). La lista vuota è così rappresentabile da un puntatore di tipo `struct int_queue *` inizializzato a `NULL`.

Si scriva il codice di una funzione avente il seguente prototipo:

# Laboratorio di Sistemi Operativi

## 9 febbraio 2016

### Compito

```
void append(struct int_queue *head, int n);
```

che inserisca il valore intero `n` in fondo alla coda puntata da `head`, allocando la memoria necessaria ed aggiornando opportunamente i puntatori.

#### Soluzione:

Esempio di soluzione iterativa:

```
1 void append(struct int_queue *head, int n) {
2     while(head->next!=NULL) {
3         head=head->next;
4     }
5
6     head->next=(struct int_queue *)malloc(sizeof(struct int_queue));
7     if(head->next==NULL) {
8         perror("Errore di allocazione della memoria!\n");
9         return;
10    }
11    head->next->val=n;
12    head->next->next=NULL;
13 }
```

4. Si consideri il seguente programma `ordered_vector.c` (per semplicità sono state omesse le direttive `include`):

```
1 int *buf; /* la variabile è globale, ovvero, accessibile a tutti i thread
2          in modo concorrente! */
3 void *generate(void *n) { /* funzione che genera un elemento intero
4                          e lo inserisce in modo ordinato in buf */
5     int i, t, r;
6     r=random();           // genero il valore intero casuale
7     printf("Valore generato: %d\n", r);
8
9     for(i=0; i<((int)n); i++) {
10        if(buf[i]==-1) {      // la posizione corrente è libera
11            buf[i]=r;
12            break;
13        }
14        else {
15            if(buf[i]<=r) /* posizione occupata da un elemento minore od uguale:
16                           prosegua con l'iterazione successiva */
17                continue;
18            else {
19                sleep(random()%2+1); // il thread attende da 1 a 2 secondi
20                /* posizione occupata da un elemento maggiore: scambio i valori di r e
21                   di buf[i] */
22                t=buf[i];
23                buf[i]=r;
24                r=t;
25            }
26        }
27    }
28
29    int i, n=atoi(argv[1]);
30    pthread_t *thr;
31    thr=(pthread_t *)malloc(n*sizeof(pthread_t)); /* alloco memoria per un array
32   di n thread id */
```

# Laboratorio di Sistemi Operativi

## 9 febbraio 2016

### Compito

```
32     buf=(int *) malloc(n*sizeof(int)); /* alloco memoria per un array di n interi
33             */
34     for( i=0;i<n; i++) { /* gli elementi del vettore sono inizializzati a -1 */
35         buf[ i]=-1;      /* la posizione i-esima è libera */
36     }
37
38     srand( time(NULL) ); /* inizializzo il generatore di numeri casuali */
39     for( i=1; i<=n; i++) {
40         if( pthread_create(&thr[ i-1],NULL, generate,(void *)n)!=0) { // creo il
41             thread i-esimo
42             perror("Errore nella creazione del thread.\n");
43             exit(1);
44         }
45
46         for( i=1;i<=n; i++) {
47             pthread_join( thr[ i-1],NULL); /* attendo la terminazione dell'i-esimo
48             thread figlio */
49         }
50         free( thr); // libero la memoria allocata per il vettore di thread id
51
52         for( i=0;i<n; i++) { // stampo il contenuto di buf
53             printf("buf[%d]=%d\n",i,buf[ i]);
54         }
55         free( buf); // libero la memoria allocata per il vettore buf
56     }
57 }
```

Una volta compilato, l'eseguibile accetta un intero **n** sulla linea di comando e genera un numero **n** di thread figli, ognuno dei quali (tramite la funzione **generate**) produce un valore intero casuale, memorizzandolo nel vettore **buf** in modo ordinato crescente:

```
> ./ordered_vector 5
Valore generato: 1807604932
Valore generato: 38608569
Valore generato: 1799968236
Valore generato: 1322540687
Valore generato: 1094827781
buf[0]=38608569
buf[1]=1094827781
buf[2]=1322540687
buf[3]=1799968236
buf[4]=1807604932
```

- Il programma riportato funzionerà correttamente, ovvero, produrrà in output il vettore ordinato? Perché?
- Se si è risposto negativamente alla precedente domanda, modificare (si può cancellare/modificare/aggiungere codice a piacimento) il programma in modo che rispetti la consegna.

#### Soluzione:

- Il programma non funzionerà correttamente in quanto i vari thread potranno accedere e modificare in modo concorrente il vettore condiviso **buf** con operazioni non atomiche. Lo stato finale del vettore sarà quindi dipendente da una corsa critica. Per evitare ciò si può far ricorso ad un mutex in questo modo:

- dichiarare ed inizializzare un mutex globalmente accessibile, ad esempio, aggiungendo la linea seguente fra le righe 2 e 3:

```
1     pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
```

**Laboratorio di Sistemi Operativi**  
**9 febbraio 2016**  
**Compito**

2. isolare la sezione critica della funzione `generate` introducendo in linea 8 la chiamata seguente:

```
1     pthread_mutex_lock(&mutex);
```

ed introducendo una nuova linea fra le righe 25 e 26 con la chiamata duale:

```
1     pthread_mutex_unlock(&mutex);
```

Un'altra possibilità è quella di sequenzializzare l'esecuzione dei thread, facendo in modo che il padre attenda la terminazione del thread appena creato, prima di procedere alla creazione del successivo. Per far ciò è sufficiente eliminare il ciclo `for` da riga 46 a riga 48 e spostare l'istruzione in riga 47 in una nuova linea fra le righe 43 e 44.

# Laboratorio di Sistemi Operativi

## 9 luglio 2014

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (2 punti) Si scriva un comando/pipeline per visualizzare tutte le directory contenute *ricorsivamente* in `/home` *scrivibili* dagli utenti appartenenti al *gruppo* delle directory in questione (*suggerimento*: l'opzione `-R` del comando `ls` consente di eseguire un listing ricorsivo).

#### Soluzione:

```
ls -lR /home | grep '^d....w....'
```

oppure

```
ls -lR /home | grep '^d....w'
```

2. (2 punti) Cos'è un *here document*? Si faccia un esempio di utilizzo di questo concetto nella shell bash.

#### Soluzione:

Un *here document* è un modo per redirigere dell'input multilinea dalla console ad un programma. Sostanzialmente è come se l'input venisse fornito al programma sotto forma di contenuto di un file. Tuttavia il file non esiste fisicamente nel filesystem, ma viene creato "al volo" tramite redirezione dello standard input. Un esempio classico è il seguente:

```
> wc <<delim      # here document
? queste linee formano il contenuto
? del testo
? delim
2    7    44
```

dove il testo fornito in input al comando `wc` viene digitato alla console fintanto che non si incontra una linea contenente il delimitatore `delim` che indica la fine dell'input, ovvero, la fine del documento temporaneo.

3. (3 punti) Sapendo che il comando `date`, quando invocato sulla linea di comando, produce in output una stringa come la seguente:

```
Mon Jul  7 16:46:11 CEST 2014
```

si spieghi qual è l'effetto dei seguenti comandi:

```
val='date | cut -d': -f2'                      # gli apici esterni sono dei backtick
echo $val
```

In particolare, se l'output di `date` è proprio `Mon Jul 7 16:46:11 CEST 2014`, qual è il valore stampato dall'ultimo comando?

#### Soluzione:

I backtick permettono di realizzare la cosiddetta *command substitution*, ovvero, di "catturare" l'output del comando `date | cut -d': -f2` (che normalmente verrebbe visualizzato sullo schermo del terminale) e di assegnarlo alla variabile `val`. In particolare, nel caso menzionato, l'output del comando `cut` riceve dalla pipeline la data ed ora correnti e la suddivide in campi usando il carattere *due punti* (opzione `-d':`) come separatore, estraendo il secondo campo (opzione `-f2`), ovvero, i minuti. L'output finale sarà quindi il valore assegnato a `val` in tal modo, ovvero, 46.

# Laboratorio di Sistemi Operativi

## 9 luglio 2014

### Compito

4. (4 punti) Si progetti uno script della shell `triangle.sh` che prenda in input sulla linea di comando un intero positivo e stampi un triangolo di asterischi secondo lo schema dell'esempio seguente:

```
$ ./triangle.sh 5
*
**
***
****
*****
```

#### Soluzione:

```
i=0

while test $i -lt $1
do
    j=0
    while test $j -le $i
    do
        echo -n '*' # stampo gli asterischi di una riga senza andare a capo
        j=$[j+1]
    done
    echo # vado a capo alla fine della riga di asterischi
    i=$[i+1]
done
```

5. Classificare come vere o false le seguenti affermazioni (per quelle false giustificare le risposte):
- (a) (1 punto) La system call `fork()` restituisce al processo figlio il proprio PID, mentre al padre restituisce NULL.
  - (b) (1 punto) Le system call della famiglia `exec` consentono di sovrascrivere la memoria di un processo, iniziando l'esecuzione di un altro programma.
  - (c) (1 punto) Le pipe sono un meccanismo di comunicazione tra processi.
  - (d) (1 punto) Le pipe possono essere gestite con diverse politiche (ad esempio, FIFO, LIFO ecc.) e sono bidirezionali (ovvero, con un singolo descrittore di file è possibile leggere e scrivere contemporaneamente nel canale).
  - (e) (1 punto) Tramite l'apposita system call è possibile creare insiemi di semafori con un'unica chiamata.
  - (f) (1 punto) Le socket possono essere utilizzate soltanto con il dominio Internet, ovvero, fra processi residenti su macchine distinte collegate in rete. Non è possibile utilizzarle per far comunicare processi in esecuzione sulla stessa macchina.

#### Soluzione:

- (a) Falso: la chiamata `fork()` restituisce il PID del figlio al padre e 0 al figlio.
- (b) Vero.
- (c) Vero.
- (d) Falso: le pipe possono essere gestite soltanto con politica FIFO e sono unidirezionali (infatti servono due descrittori di file per leggere e scrivere in una pipe).
- (e) Vero.
- (f) Falso: usando il dominio UNIX, ad esempio, si possono utilizzare le socket anche fra i processi di una singola macchina, senza necessitare di connessioni di rete.

# Laboratorio di Sistemi Operativi

## 9 luglio 2014

### Compito

6. Completare i seguenti frammenti di codice (i ... indicano le parti mancanti):



**Soluzione:**

- (a) int n;  
printf("Inserisci un numero decimale intero: ");  
scanf("%d",&n);  
printf("Il numero inserito e': %d\n",n);

(b) char \*s1="Ciao, mondo!";  
char \*s2=(char \*)malloc(strlen(s1)+1); // +1 per il terminatore di stringa  
strcpy(s2,s1); // copia s1 in s2  
printf("s2: %s\n",s2);

7. Individuare l'errore nei seguenti frammenti di codice:

- (a) (2 punti) `filedes = open("nomefile", O_RDONLY);  
write(filedes, "prova", strlen("prova"));`
  - (b) (2 punti) `filedes=open("prova.txt",O_RDONLY);  
lseek(filedes, (off_t)-40, SEEK_SET);`

**Soluzione:**

- (a) Avendo aperto il file `nomefile` in sola lettura (`O_RDONLY`), la successiva `write` fallirà.
  - (b) La chiamata a `lseek` cerca di applicare un offset negativo (-40) al puntatore alla posizione corrente del file aperto. Ciò non è permesso in quanto equivarrebbe a spostarsi a prima dell'inizio del file stesso (`SEEK_SET`).

8. Si scriva un programma C che prenda in input (come argomento sulla linea di comando) il percorso di un file e stampi a video le seguenti informazioni:

- (a) (1 punto) la dimensione (logica) in byte del file;
  - (b) (1 punto) la dimensione del blocco logico del filesystem;
  - (c) (3 punti) il numero di byte “sprecati” a causa della frammentazione interna.

Si ignori la gestione degli eventuali errori. *Suggerimento:* si utilizzi la struttura **struct stat** e le relative system call (i membri della struttura da utilizzare sono: **st\_size** per la dimensione logica in byte e **st\_blksize** per la dimensione del blocco logico; se si preferisce utilizzare il membro **st\_blocks**, si faccia attenzione al fatto che esso rappresenta il numero di blocchi *fisici*, non logici, da 512 byte allocati).

**Soluzione:**

```
#include <stdio.h>
#include <sys/stat.h>
```

**Laboratorio di Sistemi Operativi**  
**9 luglio 2014**  
**Compito**

```
#include <sys/types.h>

int main(int argc, char *argv[]) {
    struct stat buf;
    long int dim, blk;

    stat(argv[1], &buf);
    dim=buf.st_size;
    blk=buf.st_blksize;
    printf("Dimensione del file %s: %ld byte\n", argv[1], dim);
    printf("Dimensione del blocco: %ld byte\n", blk);
    printf("Frammentazione interna: ");
    /* Se la dimensione del file (dim)
     * e' un multiplo del blocco (dim%blk==0),
     * allora la frammentazione interna e' 0.
     * Altrimenti e' pari a blk-(dim%blk), ovvero,
     * il residuo di byte non utilizzati (sprecati)
     * nell'ultimo blocco.
     */
    printf("%ld byte\n", ((dim%blk==0) ? 0 : blk-(dim%blk)));
    return 0;
}
```

# Laboratorio di Sistemi Operativi

## 11 febbraio 2014

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Illustrare un modo per individuare qual è la propria shell di login.

**Risposta:** Digitare al prompt della shell il comando seguente:

```
echo $SHELL
```

2. Si supponga che la directory *corrente A* (contenente il file **f1**) si trovi nel filesystem del disco rigido, mentre la directory *B* si trovi nella radice del filesystem di una chiavetta USB (formattata con filesystem Unix) montata in /mnt/usbmedia0. Cosa succede se l'utente digita i comandi seguenti (si supponga che l'utente abbia tutti i privilegi necessari)?

1. **ln f1 /mnt/usbmedia0/B/f1\_link**
2. **ln -s f1 /mnt/usbmedia0/B/f1\_link**

**Risposta:**

1. Il tentativo di creazione del link hard fallisce in quanto link e file si troverebbero su due dispositivi fisici distinti (con array di inode distinti).
2. Viene creato il link simbolico **f1\_link** (all'interno della directory *B* nella radice della chiavetta USB) al file **f1** che risiede nel filesystem del disco rigido.
3. Qual è l'effetto dei seguenti comandi (nella sequenza fornita)?

```
cd  
ls -al | grep '^d'
```

**Risposta:** il primo comando (**cd**) sposta l'utente nella sua home directory. La pipeline successiva mostra in output su schermo soltanto le directory contenute nella home directory dell'utente. Infatti l'output di **ls -al** viene filtrato dal comando **grep** che "lascia passare" soltanto le linee che iniziano con il carattere **d**, ovvero, le linee corrispondenti alle directory.

4. Si predisponga uno script della shell che legga dallo standard input una serie di numeri interi, fermandosi quando incontra la stringa **stop** (su una linea da sola) e stampandone la somma su standard output. Si ignori la gestione degli eventuali errori.

**Risposta:**

```
read linea  
  
somma=0  
  
while test $linea != 'stop'  
do  
    somma=$((somma + linea))  
    read linea  
done  
  
echo $somma
```

5. Spiegare qual è l'effetto delle seguenti dichiarazioni di variabili in C:

1. **int i;**
2. **int \*ip;**
3. **int a1[10];**
4. **int a2[]={0,1,2,3,4,5,6,7,8,9};**
5. **int m[10][20];**
6. **int \*b[10];**

# Laboratorio di Sistemi Operativi

## 11 febbraio 2014

### Compito

**Risposta:** L'effetto è il seguente:

1. viene dichiarata la variabile intera `i`;
2. viene dichiarato il puntatore `ip` ad interi;
3. viene dichiarato un vettore `a1` di 10 interi con indici da 0 a 9.
4. viene dichiarato un vettore `a2` di 10 interi: il vettore viene anche inizializzato con i valori 0, 1, ..., 9;
5. viene dichiarato un vettore `m` di  $10 \times 20$  elementi interi (ovvero, una matrice di interi di 10 righe e 20 colonne);
6. viene dichiarato un vettore `b` di dieci puntatori ad interi;
7. Si scriva il codice C necessario per leggere dallo standard input delle quadruple di interi, memorizzando soltanto il primo ed il terzo di questi nelle variabili `x` e `y`, rispettivamente. La lettura avrà termine al momento in cui verrà rilevato l'end-of-file (`EOF`). Si ignori la gestione degli eventuali errori.

**Risposta:**

```
int x,y;  
  
while(scanf("%d %*d %d %*d",&x,&y)!=EOF);
```

7. Individuare l'errore nel seguente frammento di codice C:

```
int *ip;  
printf("Il valore puntato da ip e': %d\n",*ip);
```

**Risposta:** il puntatore `ip` viene utilizzato in un deriferimento senza essere stato inizializzato (errore logico).

8. Si assumano le seguenti direttive e dichiarazioni in C:

```
#define MAX 81  
...  
struct dati {  
    int lunghezza;  
    char testo[MAX];  
};  
struct dati *d;  
char s[]="Ciao, mondo!";
```

Scrivere il codice per memorizzare (tramite il *puntatore d*) il valore di `s` e la sua lunghezza nei campi `testo` e `lunghezza`, rispettivamente.

**Risposta:**

```
d=(struct dati *)malloc(sizeof(struct dati));  
strcpy(d->testo,s);  
d->lunghezza=strlen(d->testo);
```

9. Il programma seguente dichiara una variabile `visit_log` (un array di caratteri) e lancia in esecuzione un numeroMAX di thread. Ognuno di questi ultimi deve accedere a `visit_log`, stamparne il valore corrente ed aggiornarlo con la stringa ricevuta tramite la chiamata `pthread_create`, ovvero, `msg[i]` per il thread `i`-esimo.

(a) Si completi il sorgente specificando i comandi mancanti da inserire al posto dei ... nei 5 punti indicati, affinché un solo thread per volta possa accedere in modo esclusivo al vettore di caratteri `visit_log`.

# Laboratorio di Sistemi Operativi

## 11 febbraio 2014

### Compito

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

#define MAX 10
#define STRLEN 81

...
char visit_log[STRLEN] = "vuoto"; # <- completare (punto 1)

void *update_log(void *ptr);

main() {
    pthread_t thread[MAX];
    char msg[MAX][STRLEN];
    int i;

    for(i=0; i<MAX; i++) {
        sprintf(msg[i], "%s%d", "Thread n.", i+1);
        printf("%s\n", msg[i]);

        if(pthread_create(&thread[i], NULL,
            (void *)&update_log, ...)!=0) { # <- completare (punto 2)
            fprintf(stderr, "Errore nella creazione del thread n. %d/%d.\n", i+1, MAX);
            exit(1);
        }
    }

    for(i=0; i<MAX; i++) {
        pthread_join(thread[i], NULL);
    }

    printf("Ultimo visitatore rilevato: %s\n", visit_log);
    exit(0);
}

void *update_log(void *ptr) {
    printf("%s - in attesa di accedere al log\n", (char *)ptr);
    ...
    printf("%s - accesso al log; precedente visitatore rilevato: %s\n", (char *)ptr, visit_log); # <- completare (punto 3)
    strcpy(visit_log, ...); # <- completare (punto 4)
    printf("%s - log aggiornato\n", (char *)ptr); # <- completare (punto 5)
    ...
    printf("%s - rilascio del log\n", (char *)ptr);
}
```

(b) Cosa potrebbe succedere nel caso in cui nel programma precedente venissero eliminate le seguenti linee del `main`?

```
for(i=0; i<MAX; i++) {
    pthread_join(thread[i], NULL);
}
```

#### Risposta:

(a) I punti vanno completati come segue:

**Laboratorio di Sistemi Operativi**  
**11 febbraio 2014**  
**Compito**

1. `pthread_mutex_t log_mutex=PTHREAD_MUTEX_INITIALIZER;`
2. `(void *)msg[i]`
3. `pthread_mutex_lock(&log_mutex);`
4. `(char *)ptr`
5. `pthread_mutex_unlock(&log_mutex);`

(b) In assenza delle linee menzionate, il thread principale non attenderà la terminazione dei figli, provocando la fine prematura del processo (e di conseguenza di tutti i MAX thread creati).

# Laboratorio di Sistemi Operativi

## 13 Luglio 2023

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scrivano dei comandi/delle pipeline per risolvere i compiti seguenti:
  - (a) fornire il numero totale di processi in esecuzione sulla macchina;
  - (b) trovare e stampare a video tutti i percorsi dei sorgenti C a partire dalla directory corrente;
  - (c) cancellare tutti i file con estensione `bak` a partire dalla directory corrente;

Esempio di soluzione:

- (a) `ps ax --no-headers | wc -l`
- (b) `find . -name "*.c" -print`
- (c) `find . -name "*.bak" -exec rm {} \;`

2. (8 punti) Si scriva uno script `min_max.sh` della shell che prenda come argomento sulla linea di comando il percorso di un file e, se quest'ultimo esiste, è un file regolare, è leggibile dall'utente, stampi a video il minimo ed il massimo dei numeri che contiene, assumendo che sia costituito soltanto da interi separati da spazi bianchi (i.e., spazi, tabulazioni, newline).

Esempio, si supponga che il file file.txt contenga quanto segue:

```
32 45 54
65
65 778
54
999
-1
-888
```

Allora, l'esecuzione di `min_max.sh file.txt` deve stampare a video:

```
Min: -888, Max: 999
```

Si gestiscano gli eventuali errori.

```
1 #!/bin/bash
2
3 if test $# -ne 1
4 then
5   echo "Utilizzo: $0 <file>"
6   exit 1
7 fi
8
9 if test -f $1 -a -r $1
10 then
11   linee='wc -l < $1'
12   i=1
13   read_first=0
14   while test $i -le $linee
15   do
16     linea='tail -n +$i $1 | head -1'
17     i=$[ $i + 1 ]
18
19   for n in $(echo $linea)
20   do
21     if test $read_first -eq 0
```

**Laboratorio di Sistemi Operativi**  
**13 Luglio 2023**  
**Compito**

```
22     then
23         read_first=1
24         min=$n
25         max=$n
26     else
27         if test $n -lt $min
28             then
29                 min=$n
30             fi
31             if test $n -gt $max
32                 then
33                     max=$n
34                 fi
35             fi
36         done
37     done
38     echo "Min: $min, Max: $max"
39 else
40     echo "Il file passato da linea di comando deve esistere, essere
41         leggibile ed essere regolare."
42     exit 2
43 fi
44 exit 0
```

3. (8 punti) Si scriva un programma C `min_max.c` che si comporti come lo script dell'esercizio precedente. Si gestiscano gli eventuali errori.

Esempio di soluzione:

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     if(argc!=2) {
5         fprintf(stderr,"Utilizzo: %s <file>\n",argv[0]);
6         return 1;
7     }
8
9     FILE *f=fopen(argv[1],"r");
10    if(!f) {
11        fprintf(stderr,"Impossibile aprire il file!\n");
12        return 2;
13    }
14    int min, max, i;
15    if(fscanf(f,"%d",&i)==1) {
16        min=max=i;
17
18        while(fscanf(f,"%d",&i)==1) {
19            min=i<min ? i : min;
20            max=i>max ? i : max;
21        }
22
23        printf("Min: %d, Max: %d\n", min, max);
24    }
```

# Laboratorio di Sistemi Operativi

## 13 Luglio 2023

### Compito

```
25     else {
26         fprintf(stderr,"Il file non è nel formato corretto!\n");
27         return 4;
28     }
29     fclose(f);
30     return 0;
31 }
```

4. (10 punti) Si scriva un programma C che legga da linea di comando un numero intero **n** maggiore di zero e generi altrettanti thread. Ogni thread deve leggere, ad intervalli di 5 secondi, un numero intero dal file speciale `/dev/urandom`. Quando l'utente preme la combinazione di tasti Ctrl-C (tasti Ctrl e C premuti contemporaneamente) il processo deve terminare stampando tutti i numeri letti da `/dev/urandom` fino a quel momento.

**Nota:** `/dev/urandom` è assimilabile ad un file binario.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <signal.h>
6 #include <stdbool.h>
7
8 #define MAX_THREADS 100
9
10 int n, dim, n_read;
11 int *numbers = NULL;
12 pthread_t threads[MAX_THREADS];
13 pthread_mutex_t numbers_mutex = PTHREAD_MUTEX_INITIALIZER;
14 bool stop = false;
15
16 void* read_random_number(void* arg) {
17     while (!stop) {
18         FILE* urandom = fopen("/dev/urandom", "rb");
19         if (urandom == NULL) {
20             perror("Errore nell'apertura di /dev/urandom");
21             exit(1);
22         }
23
24         int random_number;
25         fread(&random_number, sizeof(int), 1, urandom);
26         fclose(urandom);
27
28         pthread_mutex_lock(&numbers_mutex);
29         if(n_read==dim)
30             numbers=realloc(numbers , (++dim)*sizeof(int));
31         numbers[n_read++]=random_number;
32         pthread_mutex_unlock(&numbers_mutex);
33
34         sleep(5);
35     }
36
37     return NULL;
38 }
39
40 void sigint_handler(int signum) {
41     stop = true;
```

# Laboratorio di Sistemi Operativi

## 13 Luglio 2023

### Compito

```
42     for (int i = 0; i < n; i++) {
43         pthread_join(threads[i], NULL);
44     }
45
46     for(int i = 0; i<n_read; i++)
47         printf("Number %d: %d\n", i, numbers[i]);
48     free(numbers);
49
50     exit(0);
51 }
52
53 int main(int argc, char *argv[]) {
54     if (argc != 2) {
55         printf("Utilizzo: %s <numero_thread>\n", argv[0]);
56         return 1;
57     }
58
59     n = atoi(argv[1]);
60     if (n <= 0 || n > MAX_THREADS) {
61         printf("Il numero di thread deve essere compreso tra 1 e %d\n",
62                MAX_THREADS);
63         return 1;
64     }
65
66     numbers=malloc(sizeof(int));
67     dim=1;
68     n_read=0;
69
70     signal(SIGINT, sigint_handler);
71
72     for (int i = 0; i < n; i++) {
73         if (pthread_create(&threads[i], NULL, read_random_number, NULL)
74             != 0) {
75             perror("Errore nella creazione del thread");
76             return 1;
77         }
78     }
79
80     for (int i = 0; i < n; i++) {
81         pthread_join(threads[i], NULL);
82     }
83
84     free(numbers);
85
86     return 0;
87 }
```

# Laboratorio di Sistemi Operativi

## 13 Settembre 2019

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che stampi a video il numero di utenti che abbiano la home directory all'interno del percorso `/home`.

**Suggerimento:** si ricorra al file `/etc/passwd` dove ogni linea è una successione di campi separati dai due punti (`:`) e la home directory è rappresentata dal sesto campo. Ad esempio:

```
pippo:x:1017:1019:Pippo,,,:/home/fernandez:/bin/bash
```

```
cat /etc/passwd | cut -d':' -f6 | grep '^/home' | wc -l
```

2. (6 punti) Si scriva uno script `home.sh` della shell che stampi a video il numero di sottodirectory visibili di primo livello (quindi non si deve scendere ricorsivamente nell'albero del filesystem) della directory `/home`.

```
1 count=0;
2
3 for i in /home/*
4 do
5     if test -d $i
6     then
7         count=$((count+1))
8     fi
9 done
10
11 echo $count
12
13 exit 0
```

3. (10 punti) 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 dei numeri interi dallo standard input fino a quando non incontri il segnale di fine file (EOF);
2. per ogni intero letto dal file allochi un nodo della lista e vi memorizzi il numero intero;
3. alla fine stampi la lista di interi letti e memorizzati e la rimuova dalla memoria.

Suggerimento: usare `scanf()` 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;
};
```

**Laboratorio di Sistemi Operativi**  
**13 Settembre 2019**  
Compito

```
typedef struct node_t Node;

Node *insert(Node *p, int d);
void list(Node *p);
void delete(Node *p);

int main() {
    Node *head=NULL;
    int n;

    while(scanf("%d",&n)==1) {
        head=insert(head,n);
    }

    list(head);
    delete(head);
    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;
}

void list(Node *p) {
    if(p==NULL) {
        printf("_\n");
    } else {
        printf("%d -> ",p->data);
        list(p->next);
    }
}

void delete(Node *p) {
    Node *q;

    if(p!=NULL) {
        q=p->next;
        free(p);
        delete(q);
    }
}
```

4. (12 punti) Si scriva un programma C (**stampa\_file.c**) che accetti su linea di comando il percorso di un file, generi un processo figlio e si metta in attesa della sua terminazione prima di terminare a

# Laboratorio di Sistemi Operativi

## 13 Settembre 2019

### Compito

sua volta. Il processo figlio deve stampare a video il contenuto del file passato su linea di comando. La gestione degli errori (e.g., mancanza del parametro su linea di comando, numero di parametri errati ecc.) deve essere eseguita dal padre prima della generazione del processo figlio (che andrà in porto solo in caso di assenza di errori).

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5
6 int main(int argc, char **argv) {
7     pid_t pid;
8     FILE *f;
9     char c;
10
11    if(argc!=2) {
12        fprintf(stderr,"wrong number of arguments!\n");
13        return 1;
14    }
15
16    f=fopen(argv[1],"r");
17
18    if(f==NULL) {
19        fprintf(stderr,"fopen failed!\n");
20        return 1;
21    }
22
23    pid=fork();
24
25    switch(pid) {
26        case -1:
27            fprintf(stderr,"fork failed!\n");
28            return 1;
29        case 0:
30            while((c=fgetc(f))!=EOF) {
31                printf("%c",c);
32            }
33            fclose(f);
34
35            return 0;
36        default:
37            waitpid(pid,NULL,0);
38    }
39
40    return 0;
41 }
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2018

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) sapendo che il comando `date` produce un output come il seguente:

```
gio 1 feb 2018 15:45:26
```

si completi la pipeline seguente, aggiungendo quanto necessario (al posto dei ...) per estrarre le ore (ovvero, nell'esempio riportato il numero 15):

```
date | tr -s ' ' | ...
```

A cosa serve il comando `tr -s ' '`?

Per completare la pipeline ed estrarre l'anno va aggiunta la pipeline `cut -d ' ' -f5 | cut -d ':' -f1`. Il comando `tr -s ' '` serve a comprimere eventuali spazi multipli come per esempio quelli tra `gio` e `1` in modo da non sbagliare il riferimento numerico al campo da estrarre con il parametro `-f` del comando `cut`.

2. (5 punti) Si completi il ciclo `while` dello script `explore.sh` seguente:

```
1 # imposta i separatori interni alla stringa vuota in modo da
   preservare i newline nelle command substitution
2 IFS=
3 lista_file='find $1' # lista_file conterrà tante righe quante sono
4                   # quelle prodotte dal find
5 l='echo $lista_file | wc -l' # calcola il numero di elementi (righe)
6                   # di lista_file
7 i=1
8 while test $i -le $l;
9 do
10   f='echo $lista_file | ...'
11   echo -n $f '--> '
12   ...                                # qui possono esserci più linee di codice
13 done
```

in modo che lo script stampi per ogni elemento in `lista_file`, l'elemento stesso (il pathname), una freccia composta dai caratteri `-->` e la scritta `directory` se si tratta di una directory oppure `file` in caso contrario. Ad esempio:

```
$ ./explore.sh ./concurrent_programming/
./concurrent_programming/prenotazioni.txt --> file
./concurrent_programming/acme_thread.exe --> file
./concurrent_programming/acme_thread.c --> file
./concurrent_programming/ --> directory
```

```
1 IFS=
2 lista_file='find $1'
3 l='echo $lista_file | wc -l'
4 i=1
5
6 while test $i -le $l;
7 do
8   f='echo $lista_file | tail -n $i | head -1'
9   echo -n $f '--> '
10  if test -d $f;
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2018

### Compito

```
11     then
12         echo 'directory'
13     fi
14     if test -f $f;
15     then
16         echo 'file'
17     fi
18     i=$[$i + 1]
19 done
```

3. (8 punti) Scrivere il codice di un programma C che riceva in input il percorso di un file e generi un secondo file che contenga lo stesso contenuto del primo file a cui viene applicato il seguente filtro: devono essere rimossi dal testo del primo file tutti i caratteri eccetto le lettere dell'alfabeto (maiuscole e minuscole). Si gestiscano inoltre gli eventuali errori (numero di argomenti errato, file non leggibile, ecc.).

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

#define LEN 1024

int main(int argc, char **argv) {
    char line[150];
    int i, j;
    FILE *f,*f2;

    if(argc<2) {
        fprintf(stderr,"Use: filter filename\n");
        return 1;
    }

    f=fopen(argv[1],"r");

    if(f==NULL) {
        fprintf(stderr,"Error opening %s\n",argv[1]);
        return 2;
    }

    f2=fopen("secondo_file","w");

    if(f2==NULL) {
        fprintf(stderr,"Error opening second file\n");
        return 2;
    }

    while(fgets(line,LEN-1,f)!=NULL) {
        for(i = 0; line[i] != '\0'; ++i) {
            while (!( (line[i] >= 'a' && line[i] <= 'z') ||
                      (line[i] >= 'A' && line[i] <= 'Z') || (line[i] == '\0'))) {
                for(j = i; line[j] != '\0'; ++j) {
                    line[j] = line[j+1];
                }
                line[j] = '\0';
            }
        }
    }
}
```

**Laboratorio di Sistemi Operativi**  
**14 Giugno 2018**  
**Compito**

```
        }
    }
    fputs(line,f2);
}

fclose(f2);
fclose(f);
return 0;
}
```

4. (5 punti) Scrivere l'output del seguente programma C.

```
1 #include <stdio.h>
2 int main()
3 {
4     int levels=5;
5
6     for (int i = levels-1; i >= 0; i--) {
7         for (int j = 0; j < levels - i; j++){
8             printf(" ");
9         }
10        for (int k = 0; k < (2 * i + 1); k++){
11            printf("*");
12        }
13        printf("\n");
14    }
15    return 0;
16 }
```

```
*****
*****
****
 ***
 *
```

5. (8 punti) Completare il seguente codice che fa uso di allocazione dinamica della memoria.

```
1 #include <stdio.h>
2 #include<stdlib.h>
3
4 struct course
5 {
6     int marks;
7     char subject[30];
8 };
9
10 int main()
11 {
12     struct course *ptr;
13     int i, noOfRecords;
14     printf("Enter number of records: ");
15     scanf(...);
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2018

### Compito

```
16
17 // Allocates the memory for noOfRecords structures with pointer
18 // ptr pointing to the base address .
19 ...
20
21 for(i = 0; i < noOfRecords; ++i){
22     printf("Enter name of the subject and marks respectively:\n");
23     scanf("%s %d", ... , ... );
24 }
25
26 printf("Displaying Information:\n");
27
28 for(i = 0; i < noOfRecords ; ++i)
29     printf("%s\t%d\n", ... , ... );
30 return 0;
31 }
```

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

struct course
{
    int marks;
    char subject[30];
};

int main()
{
    struct course *ptr;
    int i, noOfRecords;
    printf("Enter number of records: ");
    scanf("%d", &noOfRecords);

    // Allocates the memory for noOfRecords structures with pointer
    // ptr pointing to the base address .
    ptr = (struct course*) malloc (noOfRecords * sizeof(struct course));

    for(i = 0; i < noOfRecords; ++i){
        printf("Enter name of the subject and marks respectively:\n");
        scanf("%s %d", &(ptr+i)->subject , &(ptr+i)->marks);
    }

    printf("Displaying Information:\n");

    for(i = 0; i < noOfRecords ; ++i)
        printf("%s\t%d\n", (ptr+i)->subject , (ptr+i)->marks);
    return 0;
}
```

6. (4 punti) Completare il seguente codice che fa uso di puntatori.

```
1 #include <stdio.h>
2 int main(){
3     int* pc;
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2018

### Compito

```
4     int c;
5     c=22;
6     printf("Indirizzo di c:%u\n", ...);
7     printf("Valore di c:%d\n\n", ...);
8     pc=&c;
9     printf("Indirizzo memorizzato nel puntatore pc:%u\n", ...);
10    printf("Valore puntato da pc:%d\n\n", ...);
11    c=11;
12    printf("Indirizzo memorizzato nel puntatore pc:%u\n", ...);
13    printf("Valore puntato da pc:%d\n\n", ...);
14    *pc=2;
15    printf("Indirizzo di c:%u\n", ...);
16    printf("Valore di c:%d\n\n", ...);
17    return 0;
18 }
```

```
#include <stdio.h>
int main(){
    int* pc;
    int c;
    c=22;
    printf("Address of c:%u\n",&c);
    printf("Value of c:%d\n\n",c);
    pc=&c;
    printf("Address of pointer pc:%u\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    c=11;
    printf("Address of pointer pc:%u\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    *pc=2;
    printf("Address of c:%u\n",&c);
    printf("Value of c:%d\n\n",c);
    return 0;
}
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2023

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `psn.sh` che ogni 5 secondi stampi a video il numero totale di processi in esecuzione sulla macchina. Lo script termina la propria esecuzione quando nella cartella di lancio è presente un file con nome `stop`.

Esempio di soluzione:

```
#!/bin/bash

while true; do
    # Conta il numero totale di processi
    num_processi=$(ps -e --no-headers | wc -l)

    # Stampa il numero totale di processi
    echo "Numero totale di processi in esecuzione: $num_processi"

    # Verifica se il file "stop" è presente nella cartella di
    # lancio
    if [ -f "stop" ]; then
        echo "Terminazione dello script."
        break
    fi

    # Attendi 5 secondi
    sleep 5
done
```

2. (8 punti) Si scriva uno script `dim.sh` della shell che prenda come argomento sulla linea di comando il percorso di un file e, se quest'ultimo esiste, è un file regolare ed è leggibile dall'utente, ne stampi a video la dimensione in byte e stampi una serie di asterischi di lunghezza pari alla dimensione.

Esempio (si supponga che il file `indice.txt` sia lungo 7 byte):

```
> ./dim.sh indice.txt
Dimensione di indice.txt: 7 byte
*****
```

```
1 #!/bin/bash
2
3 if test $# -ne 1
4 then
5     echo "Utilizzo: $0 <file>"
6     exit 1
7 fi
8
9 if test -f $1 -a -r $1
10 then
11     dim=`wc -c < $1`
12     echo "Dimensione di $1: $dim byte"
13
14     i=0;
15
16     while test $i -lt $dim
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2023

### Compito

```
17 do
18     echo -n '*'
19     i=$[$i+1]
20 done
21
22 if test $dim -gt 0
23 then
24     echo
25 fi
26
27 else
28     echo "Il file passato da linea di comando deve esistere, essere
29         leggibile ed essere regolare."
30     exit 2
31 fi
32 exit 0
```

3. (8 punti) Sia MAX\_SIZE un intero che rappresenti la dimensione massima che possono assumere le righe e le colonne di una matrice. Scrivere il codice di un programma C che esegua i seguenti compiti:

1. dichiari una struttura Matrix con i seguenti membri:
  - int rows: numero delle righe della matrice;
  - int cols: numero delle colonne della matrice;
  - int \*\*data: puntatore per la memorizzazione degli elementi della matrice;
2. legga dalla linea di comando le dimensioni di righe e colonne;
3. se le dimensioni sono minori od uguali a MAX\_SIZE, si dichiari ed inizializzi una variabile di tipo struct Matrix, allocando la memoria necessaria e chiedendo all'utente di inserire gli elementi della matrice;
4. prima di uscire dal processo, si liberi la memoria allocata per la matrice.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX_SIZE 100
5
6 typedef struct Matrix {
7     int rows;
8     int cols;
9     int **data;
10 } Matrix;
11
12 // Funzione per l'allocazione di memoria per una matrice
13 int** allocateMatrix(int rows, int cols) {
14     int **matrix = (int **)malloc(rows * sizeof(int *));
15     for (int i = 0; i < rows; i++) {
16         matrix[i] = (int *)malloc(cols * sizeof(int));
17     }
18     return matrix;
19 }
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2023

### Compito

```
21 // Funzione per la liberazione della memoria di una matrice
22 void freeMatrix(int **matrix, int rows) {
23     for (int i = 0; i < rows; i++) {
24         free(matrix[i]);
25     }
26     free(matrix);
27 }
28
29 // Funzione per la lettura della matrice dall'utente
30 void readMatrix(Matrix *matrix) {
31     printf("Inserisci gli elementi della matrice:\n");
32     for (int i = 0; i < matrix->rows; i++) {
33         for (int j = 0; j < matrix->cols; j++) {
34             scanf("%d", &(matrix->data[i][j]));
35         }
36     }
37 }
38
39 int main() {
40     Matrix matrix;
41     int rows, cols;
42
43     printf("Inserisci il numero di righe della matrice: ");
44     scanf("%d", &rows);
45
46     printf("Inserisci il numero di colonne della matrice: ");
47     scanf("%d", &cols);
48
49     if (rows <= MAX_SIZE && cols <= MAX_SIZE) {
50         matrix.rows = rows;
51         matrix.cols = cols;
52         matrix.data = allocateMatrix(rows, cols);
53
54         readMatrix(&matrix);
55
56         printf("Matrice inserita:\n");
57         for (int i = 0; i < matrix.rows; i++) {
58             for (int j = 0; j < matrix.cols; j++) {
59                 printf("%d ", matrix.data[i][j]);
60             }
61             printf("\n");
62         }
63
64         freeMatrix(matrix.data, matrix.rows);
65     } else {
66         printf("Dimensioni non valide. Le dimensioni devono essere
67             minori o uguali a %d\n", MAX_SIZE);
68     }
69
70     return 0;
71 }
```

4. (10 punti) Si scriva un programma C che legga da linea di comando un numero intero **n** maggiore di zero e generi altrettanti thread. Ogni thread deve leggere, ad intervalli di 5 secondi, un numero intero dal file speciale **/dev/urandom**. Quando l'utente preme la combinazione di tasti Ctrl-C (tasti

# Laboratorio di Sistemi Operativi

## 14 Giugno 2023

### Compito

Ctrl e C premuti contemporaneamente) il processo deve terminare stampando il massimo numero letto fino a quel momento.

**Nota:** /dev/urandom è assimilabile ad un file binario.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <signal.h>
6 #include <stdbool.h>
7
8 #define MAX_THREADS 100
9
10 int n;
11 int max_number = 0;
12 pthread_t threads[MAX_THREADS];
13 pthread_mutex_t max_number_mutex = PTHREAD_MUTEX_INITIALIZER;
14 volatile bool stop = false;
15
16 void* read_random_number(void* arg) {
17     while (!stop) {
18         FILE* urandom = fopen("/dev/urandom", "rb");
19         if (urandom == NULL) {
20             perror("Errore nell'apertura di /dev/urandom");
21             exit(1);
22         }
23
24         int random_number;
25         fread(&random_number, sizeof(int), 1, urandom);
26         fclose(urandom);
27
28         pthread_mutex_lock(&max_number_mutex);
29         if (random_number > max_number) {
30             max_number = random_number;
31         }
32         pthread_mutex_unlock(&max_number_mutex);
33
34         sleep(5);
35     }
36
37     return NULL;
38 }
39
40 void sigint_handler(int signum) {
41     stop = true;
42     printf("Massimo numero letto: %d\n", max_number);
43
44     for (int i = 0; i < n; i++) {
45         pthread_join(threads[i], NULL);
46     }
47
48     exit(0);
49 }
50
51 int main(int argc, char *argv[]) {
52     if (argc != 2) {
53         printf("Utilizzo: %s <numero_thread>\n", argv[0]);
54         return 1;
55     }
56 }
```

# Laboratorio di Sistemi Operativi

## 14 Giugno 2023

### Compito

```
56
57     n = atoi(argv[1]);
58     if (n <= 0 || n > MAX_THREADS) {
59         printf("Il numero di thread deve essere compreso tra 1 e %d\n",
60                MAX_THREADS);
61         return 1;
62     }
63     signal(SIGINT, sigint_handler);
64
65     for (int i = 0; i < n; i++) {
66         if (pthread_create(&threads[i], NULL, read_random_number, NULL)
67             != 0) {
68             perror("Errore nella creazione del thread");
69             return 1;
70         }
71     }
72     for (int i = 0; i < n; i++) {
73         pthread_join(threads[i], NULL);
74     }
75
76     return 0;
77 }
```

# Laboratorio di Sistemi Operativi

## 14 luglio 2015

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Qual è il significato e l'importanza dell'*exit status* dei comandi e degli script della shell?
- (b) Si indichi un comando per visualizzare su standard output l'esito (exit status) dell'ultimo comando eseguito.

#### Soluzione:

- (a) L'*exit status* di un comando o script della shell è un intero che rappresenta l'esito della sua esecuzione. In particolare, per convenzione, si stabilisce che un exit status pari a 0 indica una terminazione normale, mentre un exit status diverso da zero indica una terminazione con errore. Ciò è importante in quanto l'analisi di tale valore consente di eseguire delle scelte nel flusso di esecuzione dei comandi successivi (attraverso l'utilizzo dei costrutti di scelta condizionale e di iterazione negli script, ad esempio).
- (b) Un esempio di comando per visualizzare su standard output l'esito (exit status) dell'ultimo comando eseguito è il seguente:  
`echo $?`

2. Qual è l'effetto dei seguenti comandi?

1. `lista='ls ~'`
2. `echo '$lista'`
3. `echo "$lista"`

**Attenzione:** nel punto 1 gli apici sono dei *backquote* (apici rovesciati).

#### Soluzione:

1. `lista='ls ~'` esegue il comando `ls ~` e ne salva l'output (ovvero l'elenco dei file contenuti nella home directory dell'utente) come valore della variabile `lista`.
2. `echo '$lista'` produce la visualizzazione su standard output della stringa `$lista`, dato che gli apici singoli inibiscono il metacarattere `$`.
3. `echo "$lista"` produce la visualizzazione su standard output del valore della variabile `lista`, dato che le virgolette non inibiscono il metacarattere `$`.

3. Si predisponga uno script della shell `ord_ext.sh` che prenda come argomento sulla linea di comando il percorso di una directory e compia le seguenti azioni:
  - (a) controlli il numero degli argomenti forniti, terminando la propria esecuzione nel caso il numero sia diverso da uno;
  - (b) controlli che il percorso fornito corrisponda ad una directory e sia leggibile dall'utente (altrimenti l'esecuzione deve terminare);
  - (c) esegua il comando `ls` senza opzioni sul percorso passato, visualizzandone l'output ordinato in base all'**estensione** dei vari file (per semplicità, si supponga che in ogni nome di file vi sia al più un'occorrenza del carattere punto `'.'`).

Ad esempio se nella directory `/home/user/test` sono contenuti i seguenti file:

```
compito.doc fact.sh host manuale.pdf mthread.c prova.sh relazione.tex
```

il comando `./ord_ext.sh /home/user/test` deve restituire quanto segue:

# Laboratorio di Sistemi Operativi

## 14 luglio 2015

### Compito

```
host
mthread.c
compito.doc
manuale.pdf
fact.sh
prova.sh
relazione.tex
```

#### Soluzione:

Esempio di soluzione:

```
1 if test $# -ne 1
2 then
3   echo "Utilizzo dello script:$0<path>"
4   exit 1
5 fi
6
7 if ! test -d $1 -a -r $1
8 then
9   echo "Il percorso $1 deve essere leggibile e deve essere una
       directory"
10  exit 2
11 fi
12
13 ls -r $1 | sort -t '.' -k2,3
14
15 exit 0
```

4. Sia data la seguente struttura ricorsiva in C per la rappresentazione di alberi binari:

```
struct bintree {
    int val;
    struct bintree *left;
    struct bintree *right;
};
```

dove `val` rappresenta il valore del nodo, mentre `left` e `right` puntano, rispettivamente, al figlio sinistro ed al figlio destro (tali puntatori assumono il valore `NULL` quando non esistono i rispettivi figli).

Si scriva il codice di una funzione avente il seguente prototipo:

```
int conta(struct bintree *root);
```

che restituisca come valore di ritorno il numero di nodi (elementi) dell'albero binario puntato da `root`.

#### Soluzione:

Esempio di soluzione:

```
1 int conta(struct bintree *root) {
2     if (root==NULL) return 0;
3     else
4         return 1+conta(root->left)+conta(root->right);
5 }
```

# Laboratorio di Sistemi Operativi

## 14 luglio 2015

### Compito

5. Il programma seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) e di sincronizzazione (condition variable) introdotti a lezione, per implementare una soluzione al problema classico dei produttori e consumatori con memoria limitata. Ci sono `NUM_P` thread produttori e `NUM_C` thread consumatori, che accedono in modo concorrente al vettore condiviso `buffer` con `LENGTH` posizioni per altrettanti interi positivi (che assumono valori da 1 a `MAX`). Per convenzione il valore -1 indica che la posizione del vettore è libera (vuota). Un thread produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera. Un thread consumatore (se il buffer non è vuoto) preleva un elemento dalla prima posizione non vuota.

Supponendo di avere a disposizione

- la funzione `full()` che restituisce 1 se il buffer è pieno e 0 altrimenti,
- la funzione `empty()` che restituisce 1 se il buffer è vuoto e 0 altrimenti,

si completa il sorgente, specificando i comandi mancanti da inserire al posto dei ... negli 8 punti indicati, affinché il codice risultante rappresenti una soluzione corretta del problema dei produttori e consumatori con memoria limitata.

```
int buffer[LENGTH];           // buffer condiviso di lunghezza LENGTH
pthread_t threadP[NUM_P];    // vettore che contiene gli ID dei thread produttori
pthread_t threadC[NUM_C];    // vettore che contiene gli ID dei thread consumatori

// mutex per l'accesso esclusivo
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
// condition variable: buffer non vuoto
pthread_cond_t not_empty_buffer=PTHREAD_COND_INITIALIZER;
// condition variable: buffer non pieno
pthread_cond_t not_full_buffer=PTHREAD_COND_INITIALIZER;

void *producer(void *n) {
    int elemento, i;
    while(1) {
        printf("Thread produttore (ID %lu)\n",threadP[((int)n)-1]);
        ... // <-- inizio sezione critica: completare (1)
        ... // <-- controllo se posso inserire un nuovo elemento: completare (2)
        for(i=0; i<LENGTH; i++)
            if(buffer[i]==-1) {
                elemento=random()%MAX+1; // genero l'elemento
                buffer[i]=elemento;      // inserisco l'elemento
                break;
            }
        ... // <-- quale evento devo segnalare qui? completare (3)
        ... // <-- fine sezione critica: completare (4)
    }
};

void *consumer(void *n) {
    int elemento, i;
    while(1) {
        printf("Thread consumatore (ID %lu)\n",threadC[((int)n)-1]);
        ... // <-- inizio sezione critica: completare (5)
        ... // <-- controllo se posso prelevare un elemento (6)
        for(i=0; i<LENGTH; i++)
            if(buffer[i]!=-1) {
                elemento=buffer[i]; // prelevo l'elemento
                buffer[i]=-1;       // segnalo che la posizione è libera
                break;
            }
        ... // <-- quale evento devo segnalare qui? completare (7)
        ... // <-- fine sezione critica: completare (8)
    }
};
```

**Laboratorio di Sistemi Operativi**  
**14 luglio 2015**  
**Compito**

**Soluzione:**

```
1. pthread_mutex_lock(&mutex);  
2. if(full()) pthread_cond_wait(&not_full_buffer,&mutex);  
3. pthread_cond_signal(&not_empty_buffer);  
4. pthread_mutex_unlock(&mutex);  
5. pthread_mutex_lock(&mutex);  
6. if(empty()) pthread_cond_wait(&not_empty_buffer,&mutex);  
7. pthread_cond_signal(&not_full_buffer);  
8. pthread_mutex_unlock(&mutex);
```

# Laboratorio di Sistemi Operativi

## 15 Giugno 2022

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si consideri il file `/etc/passwd` ed una variabile di ambiente `user`: si scriva una pipeline che stampi a video la riga (se esiste) del file `/etc/passwd` corrispondente al nome utente memorizzato nella variabile `user`. Ad esempio, se `user` ha come valore la stringa `root`, la pipeline deve stampare a video la linea di `/etc/passwd` relativa all'utente `root`:

```
root:x:0:0:root:/root:/bin/bash
```

Esempio di soluzione:

```
cat /etc/passwd | grep "^\$user"
```

2. (6 punti) Si realizzi uno script della shell Bash che prenda come parametro sulla linea di comando un percorso di un file di testo. Quest'ultimo deve essere composto da numeri interi separati da spazi bianchi (i.e., spazi, tabulazioni, newline), ad esempio:

```
2 4 10      -80  
7 5  
8  
  
1
```

Lo script deve leggere i numeri contenuti nel file e stampare a video il risultato della loro somma, ad esempio per il caso precedente:

```
-43
```

Si gestiscono eventuali errori dovuti al parametro mancante o alla non esistenza/accessibilità in lettura del file.

Esempio di soluzione:

```
1 if ! test $# -eq 1  
2 then  
3     echo "Utilizzo: $0 <file>"  
4     exit 1  
5 fi  
6  
7 if ! test -f $1 -a -r $1  
8 then  
9     echo "Errore nell'accesso al file $1"  
10    exit 2  
11 fi  
12  
13 s=0  
14  
15 for n in `cat $1`  
16 do  
17     s=$[ $s + $n ]  
18 done  
19  
20 echo $s  
21  
22 exit 0
```

# Laboratorio di Sistemi Operativi

## 15 Giugno 2022

### Compito

3. (8 punti) Scrivere il codice di un programma C che si comporti come lo script della shell dell'esercizio 2. Si gestiscano gli eventuali errori (parametro non presente, file non esistente o non accessibile in lettura, ecc.).

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     if(argc!=2) {
5         fprintf(stderr,"Uso: %s pathname\n",argv[0]);
6         return 1;
7     }
8
9     FILE *f=fopen(argv[1],"r");
10    if(f==NULL) {
11        fprintf(stderr,"Il file %s non è accessibile.\n",argv[1]);
12        return 2;
13    }
14
15    int s=0,x;
16    while(fscanf(f,"%d",&x)==1)
17        s+=x;
18
19    fclose(f);
20    printf("%d\n",s);
21    return 0;
22 }
```

4. (5 punti) Si consideri la seguente dichiarazione in C per rappresentare un elemento di una struttura a pila (stack):

```
1 struct stack {
2     int n;
3     struct stack *next;
4 };
```

Si scriva il codice necessario per implementare le operazioni push (per inserire in testa alla pila un nuovo elemento) e pop (per eliminare l'elemento di testa della pila) del tipo di dati astratto stack.

Risposte:

```
1 struct stack *push(struct stack *p, int n) {
2     struct stack *q=malloc(sizeof (struct stack));
3     if(q) {
4         q->n=n;
5         q->next=p;
6     }
7     return q; // q punta alla nuova cima dello stack
8 }
9
10 int pop(struct stack **p) {
11     int r=-1;
12     struct stack *q=(*p);
13     if(*p) {
14         (*p)=(*p)->next;
```

# Laboratorio di Sistemi Operativi

## 15 Giugno 2022

### Compito

```
15     r=q->n;
16     free(q);
17 }
18 return r;
19 }
```

5. (10 punti) Si scriva un programma C che chieda all'utente quanti numeri interi pseudo-casuali vuole generare. Dopodiché il programma alloca memoria sufficiente per un array che possa contenere tali numeri ed inizia a generare un numero pseudo-casuale ogni 3 secondi. Alla fine stampa tutti i numeri generati, il minimo ed il massimo fra questi e termina. Tuttavia, prima di giungere a terminazione, se l'utente preme Ctrl-C (ovvero, preme contemporaneamente i tasti Ctrl e C), il programma stampa i numeri generati fino a quel momento ed il minimo ed il massimo fra questi.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
                     attuale, assicurandosi di generare sequenze diverse ad ogni
                     esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
              tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <signal.h>
5 #include <unistd.h>
6
7 int compute_min_max=0;
8
9 void catchint(int signo) {
10     compute_min_max=1;
11 }
12
13 void min_max(long int *min, long int *max, int index, long int *
               buffer) {
14     for(int i=0;i<index;i++) {
15         printf("%ld\n",buffer[i]);
16         if(*max== -1 || *max < buffer[i])
17             *max=buffer[i];
18         if(*min== -1 || *min > buffer[i])
19             *min=buffer[i];
20     }
21 }
22
23 int main() {
24     int current_index, n;
```

**Laboratorio di Sistemi Operativi**  
**15 Giugno 2022**  
**Compito**

```
25     long int *buf, max, min;
26     current_index=n=0;
27     buf=NULL;
28     max=min=-1;
29     srand(time(NULL));
30     signal(SIGINT, catchint);
31     printf("Inserisci il numero di interi da generare: ");
32     scanf("%d",&n);
33     buf=malloc(n*sizeof(long int));
34     while(current_index<n) {
35         if(compute_min_max) {
36             printf("Numeri casuali generati finora:\n");
37             min_max(&min, &max, current_index, buf);
38             printf("Minimo numero generato finora: %ld\n",min);
39             printf("Massimo numero generato finora: %ld\n",max);
40             compute_min_max=0;
41         }
42     else {
43         buf[current_index]=random();
44         current_index++;
45     }
46     sleep(3);
47 }
48
49 printf("Numeri casuali generati:\n");
50 min_max(&min, &max, n, buf);
51 printf("Minimo numero generato: %ld\n",min);
52 printf("Massimo numero generato: %ld\n",max);
53
54 free(buf);
55 return 0;
56 }
```

# Laboratorio di Sistemi Operativi

## 18 Gennaio 2022

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Dato un file di testo di nome `test.txt` ed una variabile di ambiente `i`, si scriva una pipeline che stampi a video l'`i`-esima riga del file. Ad esempio, se il contenuto di `test.txt` è il seguente:

```
abc  
def  
ghi  
jkl
```

l'effetto della pipeline, dato l'assegnamento `i=3`, deve essere la stampa a video di:

```
ghi
```

ovvero, la terza riga del file. Si trascuri la gestione di eventuali errori (e.g., l'assegnamento alla variabile `i` di valori non congrui).

Esempio di soluzione:

```
cat test.txt | head -n +$i | tail -1
```

2. (6 punti) Si realizzi uno script della shell Bash che prenda come parametro sulla linea di comando un percorso di un file di testo. Quest'ultimo deve essere composto da linee contenenti due interi separati da una virgola, ad esempio:

```
56,89  
120,-1  
12,12  
30,5
```

Lo script deve leggere linea per linea il file e stampare a video il risultato della somma dei due numeri, ad esempio per il caso precedente:

```
145  
119  
24  
35
```

Si gestiscono eventuali errori dovuti al parametro mancante o alla non esistenza/accessibilità in lettura del file.

Esempio di soluzione:

```
1 if ! test $# -eq 1  
2 then  
3     echo "Utilizzo: $0 pathname"  
4     exit 1  
5 fi  
6  
7 if ! test -f $1 -a -r $1  
8 then  
9     echo "File $1 non accessibile"  
10    exit 2  
11 fi  
12
```

**Laboratorio di Sistemi Operativi**  
**18 Gennaio 2022**  
**Compito**

```
13 num_linee='cat $1 | wc -l'
14 i=1
15 while test $i -le $num_linee
16 do
17     linea='cat $1 | head -n +$i | tail -1'
18     num1='echo $linea | cut -d ',' -f1'
19     num2='echo $linea | cut -d ',' -f2'
20     echo ${num1}+${num2}
21     i=$((i+1))
22 done
23
24 exit 0
```

3. (8 punti) Scrivere il codice di un programma C che si comporti come lo script della shell dell'esercizio  
2. Si gestiscano gli eventuali errori (parametro non presente, file non esistente o non accessibile in lettura, ecc.).

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     if(argc!=2) {
5         fprintf(stderr,"Uso: %s pathname\n",argv[0]);
6         return 1;
7     }
8
9     FILE *f=fopen(argv[1],"r");
10    if(f==NULL) {
11        fprintf(stderr,"Il file %s non è accessibile.\n",argv[1]);
12        return 2;
13    }
14
15    int x,y;
16    while(fscanf(f,"%d,%d",&x,&y)==2)
17        printf("%d\n",x+y);
18
19    fclose(f);
20    return 0;
21 }
```

4. (5 punti) Si considerino le seguenti dichiarazioni in C:

```
1 struct list {
2     int n;
3     struct list *next;
4 };
5 struct binary_tree {
6     int n;
7     struct binary_tree *left;
8     struct binary_tree *right;
9 };
```

Si scriva il codice necessario per svolgere i seguenti compiti:

# Laboratorio di Sistemi Operativi

## 18 Gennaio 2022

### Compito

1. dichiarare due variabili del tipo appropriato ed inizializzarle per rappresentare, rispettivamente, una lista vuota ed un albero vuoto (i.e., senza nodi);
2. dichiarare una variabile del tipo appropriato ed inizializzarla in modo che rappresenti la seguente lista:

1, 2, 3

3. dichiarare una variabile del tipo appropriato ed inizializzarla in modo che rappresenti il seguente albero binario:



Risposte:

1. esempio:

```
1 struct list *l=NULL;
2 struct binary_tree *t=NULL;
```

2. esempio:

```
1 struct list *l=malloc(sizeof (struct list));
2 l->n=1;
3 l->next=malloc(sizeof (struct list));
4 l->next->n=2;
5 l->next->next=malloc(sizeof (struct list))
6 l->next->next->n=3;
7 l->next->next->next=NULL;
```

3. esempio:

```
1 struct binary_tree *t=malloc(sizeof (struct binary_tree));
2 t->n=1;
3 t->left=malloc(sizeof (struct binary_tree));
4 t->left->n=2;
5 t->left->left=t->left->right=NULL;
6 t->right=malloc(sizeof (struct binary_tree));
7 t->right->n=3;
8 t->right->left=t->right->right=NULL;
```

5. (10 punti) Si scriva un programma C `dice.c` che simuli il lancio di un dado. Il programma prende da linea di comando il numero `n` (maggiore o uguale a 1) di thread che deve lanciare in esecuzione ed un valore soglia intero `s`. Ogni thread inizia a generare un numero pseudo-casuale compreso tra 1 e 6 ogni 5 secondi. I punteggi vengono sommati ad una variabile intera `accumulatore` inizializzata a zero ed il valore di tale variabile viene stampato a video (seguito dall'ID del thread che ha eseguito l'operazione) ad ogni incremento. Quando il valore di `accumulatore` supera il valore soglia `s`, l'intero processo termina. Prima di terminare, il processo salva nel file `log.txt` il messaggio `Soglia superata dal thread con ID: x.`, dove `x` è l'ID del thread che ha provocato il superamento della soglia `s`.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
```

# Laboratorio di Sistemi Operativi

## 18 Gennaio 2022

### Compito

```
4 srand(time(NULL)); // per inizializzare il seme con la data/ora  
attuale , assicurandosi di generare sequenze diverse ad ogni  
esecuzione  
5 ...  
6 long int r;  
7 r=random(); // genera un numero pseudo-casuale con valore compreso  
tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di esecuzione:

```
1 $ ./dice 8 20  
2 2 (139687228475136)  
3 13 (139687203297024)  
4 19 (139687194904320)  
5 21 (139687186511616)  
6 7 (139687220082432)  
7 11 (139687211689728)  
8 $ cat log.txt  
9 Soglia superata dal thread con ID: 139687186511616.
```

La soluzione proposta utilizza due mutex: uno per incrementare il valore della variabile condivisa `accumulatore` e l'altro per stampare a video il punteggio raggiunto. Questa è la ragione per cui i valori a video possono apparire “non ordinati”; infatti un thread potrebbe essere sospeso in favore di un altro thread, prima di riuscire a stampare il valore (facendolo così in un secondo momento). Soluzioni che utilizzino un unico mutex e stampino il valore nella stessa regione critica in cui avviene l'aggiornamento della variabile `accumulatore` vedranno invece le stampe “ordinate”.

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <string.h>  
4 #include <time.h>  
5 #include <unistd.h>  
6 #include <pthread.h>  
7  
8 int stop=0;  
9 int accumulatore=0;  
10 int s=0;  
11 pthread_t ultimo;  
12 pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;  
13 pthread_mutex_t stampa=PTHREAD_MUTEX_INITIALIZER;  
14  
15 void *dice(void *ptr) {  
16     int local_acc=0;  
17     while(!stop) {  
18         long int r=random();  
19         int score=r%6+1;  
20         pthread_mutex_lock(&mutex);  
21         if(!stop) {  
22             accumulatore+=score;  
23             local_acc=accumulatore;  
24             if(accumulatore>s) {  
25                 stop=1;  
26                 ultimo=*((pthread_t *)ptr);  
27             }  
28         }  
    }
```

**Laboratorio di Sistemi Operativi**  
**18 Gennaio 2022**  
Compito

```
29     pthread_mutex_unlock(&mutex);
30     pthread_mutex_lock(&stampa);
31     if(local_acc!=0) {
32         printf("%d (%lu)\n",local_acc,*((pthread_t *)ptr));
33         local_acc=0;
34     }
35     pthread_mutex_unlock(&stampa);
36     sleep(5);
37 }
38 return NULL;
39 }

40 int main(int argc, char** argv) {
41     if(argc!=3) {
42         fprintf(stderr,"Uso: %s n s\n",argv[0]);
43         return 1;
44     }
45

46     int num_thread=atoi(argv[1]);
47     s=atoi(argv[2]);
48     pthread_t *thread_id=NULL;
49

50     if(num_thread>=1) {
51         thread_id=(pthread_t*)malloc(sizeof(pthread_t)*num_thread);
52
53         if(thread_id!=NULL) {
54             for(int i=0; i<num_thread; i++) {
55                 if(pthread_create(&thread_id[i],NULL,dice,&thread_id[i])
56                     !=0) {
57                     fprintf(stderr,"Errore nella creazione del thread n. %d
58                         .\n",i+1);
59                     exit(1);
60                 }
61             }
62         } else {
63             perror("Memoria insufficiente.\n");
64             exit(1);
65         }
66         for(int i=0; i<num_thread; i++)
67             pthread_join(thread_id[i],NULL);
68
69         FILE* log_file=fopen("log.txt","w");
70         char buffer[80];
71         sprintf(buffer,"Soglia superata dal thread con ID: %lu.\n",
72             ultimo);
73         fwrite(buffer,sizeof(char),strlen(buffer),log_file);
74         fclose(log_file);
75         free(thread_id);
76     }
77
78     return 0;
79 }
```

# 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
> _
```

```
1 ids='cat /etc/passwd | cut -d ":" -f3'
2 count=0
3
4 for i in $ids
5 do
6   count=$((count+$i))
7 done
8
9 echo $count
10
11 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:

# Laboratorio di Sistemi Operativi

## 18 Luglio 2019

### Compito

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):

# Laboratorio di Sistemi Operativi

## 18 Luglio 2019

### Compito

```
> ./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.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     int n1, n2;
6
7     while(scanf("%d - %d", &n1, &n2)==2) {
8         printf("%d\n", n1-n2);
9     }
10
11     return 0;
12 }
```

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
```

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5
6 int main(int argc, char **argv) {
7     pid_t pid;
8
9     pid=fork();
10
11     switch(pid) {
12         case -1:
13             fprintf(stderr,"fork failed!\n");
14             return 1;
15         case 0:
16             if(argc==2)
17                 execvp("ps","ps",argv[1],NULL);
```

**Laboratorio di Sistemi Operativi**  
**18 Luglio 2019**  
**Compito**

```
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 }
```

# Laboratorio di Sistemi Operativi

## 19 giugno 2014

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (3 punti) Si consideri il seguente output prodotto dal comando `ls -l` eseguito nella home directory dell'utente `pippo`:

```
...
-rwxr--r--  1 pippo    users   1045 Jan 25 18:05 f1
...
```

Si supponga che l'utente `pippo` esegua il comando `chmod go+x ~/f1`. Quali saranno i nuovi permessi del file `f1`? Se invece del comando precedente l'utente `pippo` eseguisse il comando `chmod go=x ~/f1`, quali sarebbero i nuovi permessi del file `f1`?

#### Soluzione:

Nel caso in cui l'utente `pippo` esegua il comando `chmod go+x ~/f1`, ai permessi correnti del gruppo e del resto degli utenti verrà aggiunto il permesso di esecuzione. Quindi i permessi finali del file `f1` saranno `rwxr-xr-x`.

Se invece l'utente `pippo` eseguisse il comando `chmod go=x ~/f1`, l'operatore `=` avrebbe un effetto diverso da `+` in quanto assegnerebbe *esattamente* e *soltamente* i permessi specificati (in questo caso al gruppo ed al resto degli utenti). Quindi i nuovi permessi del file `f1` sarebbero `rwx--x--x`.

2. (3 punti) Scrivere un unico comando (facendo uso degli opportuni operatori di combinazione di comandi) che compili il programma C contenuto nel file `prog.c` nella directory corrente e ne lanci in esecuzione l'eseguibile prodotto soltanto in caso di successo della fase di compilazione.

#### Soluzione:

Un modo per ottenere l'effetto richiesto è quello di utilizzare l'operatore condizionale `&&` di combinazione dei comandi:

```
gcc prog.c && ./a.out
```

in tal modo il prodotto della compilazione `a.out` verrà lanciato soltanto se la compilazione di `prog.c` sarà andata a buon fine.

3. (3 punti) Qual è l'effetto dei seguenti comandi (nella sequenza fornita)?

```
cd
ps -ef > log.txt
cat log.txt | grep '^pippo' | wc -l
```

#### Soluzione:

Il primo comando (`cd`) porta l'utente nella sua home directory. Il secondo comando della sequenza (`ps -ef > log.txt`) redirige l'output di `ps -ef` (ovvero il full listing di tutti i processi in esecuzione collegati o meno ad un terminale) nel file `log.txt` nella directory corrente (sostituendone il contenuto precedente nel caso in cui il file esistesse già). Infine il terzo comando (`cat log.txt | grep '^pippo' | wc -l`) legge il contenuto del file `log.txt` per darlo in pasto (tramite la pipeline) al filtro `grep` che passa in input (sempre tramite la pipeline) al comando `wc` soltanto le linee che iniziano con la stringa `pippo`; quest'ultimo (`wc`) conta il numero di tali linee e lo emette in output.

L'effetto finale è che viene riportato sullo schermo del terminale il numero dei processi dell'utente `pippo` correntemente in esecuzione (infatti il full listing del comando `ps` produce come prima informazione su ogni linea il nome dell'utente che ha lanciato il processo).

# Laboratorio di Sistemi Operativi

## 19 giugno 2014

### Compito

4. (3 punti) Si predisponga uno script della shell che prenda in input sulla linea di comando un percorso di una directory e cancelli ricorsivamente a partire da quest'ultima tutti i file aventi un nome terminante in .bak. Si ignori la gestione degli eventuali errori.

**Soluzione:**

```
find $1 -name "*.bak" -type f -exec rm {} \;
```

5. (3 punti) Spiegare qual è l'effetto delle seguenti dichiarazioni in C:

1. struct nodo\_lista {  
 int val;  
 struct nodo\_lista \*prossimo;  
};
2. struct nodo\_lista l1;
3. struct nodo\_lista \*l2;

**Soluzione:**

1. Viene dichiarata la struttura ricorsiva **struct nodo\_lista** con un membro **val** di tipo **int** ed un membro **prossimo** di tipo puntatore alla struttura stessa.
2. Viene dichiarata la variabile **l1** di tipo **struct nodo\_lista**.
3. Viene dichiarato il puntatore **l2** ad una struttura di tipo **struct nodo\_lista**.

6. In base alle dichiarazioni dell'esercizio 5, dire se i seguenti frammenti di codice C sono corretti o errati (spiegando il motivo):
- (a) (1 punto) **l1.val=50;**
  - (b) (1 punto) **l1->val=50;**
  - (c) (1 punto) **l2=(struct nodo\_lista \*)malloc(sizeof(struct nodo\_lista));**  
**l2.val=20;**
  - (d) (1 punto) **l2=(struct nodo\_lista \*)malloc(sizeof(struct nodo\_lista));**  
**l2->val=20;**
  - (e) (1 punto) **l2=(struct nodo\_lista \*)malloc(sizeof(struct nodo\_lista));**  
**(\*l2).val=20;**

**Soluzione:**

- (a) Corretto: il punto è l'operatore di selezione dei campi di una struttura.
- (b) Errato: l'operatore **->** si può utilizzare soltanto con i puntatori ad una struttura.
- (c) Errato: nonostante venga correttamente allocata la memoria per un elemento di tipo **struct nodo\_lista** ed il relativo indirizzo venga assegnato a **l2**, essendo quest'ultimo un puntatore, non si può utilizzare direttamente il punto per selezionare il membro **val** della struttura puntata.
- (d) Corretto: viene allocata la memoria per un elemento di tipo **struct nodo\_lista** ed il relativo indirizzo viene assegnato a **l2**; quindi l'operatore **->** viene utilizzato correttamente per selezionare il membro **val** della struttura puntata ai fini dell'assegnamento.

# Laboratorio di Sistemi Operativi

## 19 giugno 2014

### Compito

(e) Corretto: viene allocata la memoria per un elemento di tipo `struct nodo_lista` ed il relativo indirizzo viene assegnato a 12; quindi l'operatore di deriferimento `*` viene utilizzato per accedere alla struttura puntata ed il punto permette di selezionarne correttamente il membro `val` per l'assegnamento.

7. (4 punti) Utilizzando la struttura `nodo_lista` dichiarata nell'esercizio 5, scrivere il codice C per allocare in memoria una lista formata da 3 nodi, il primo con valore (campo `val`) 100, il secondo con valore 200 ed il terzo con valore 300. Si memorizzi il puntatore al primo nodo della lista nella variabile `testa`.

**Soluzione:**

```
struct nodo_lista *testa;
testa=(struct nodo_lista *)malloc(sizeof(struct nodo_lista));
testa->val=100;
testa->prossimo=(struct nodo_lista *)malloc(sizeof(struct nodo_lista));
testa->prossimo->val=200;
testa->prossimo->prossimo=(struct nodo_lista *)malloc(sizeof(struct nodo_lista));
testa->prossimo->prossimo->val=300;
testa->prossimo->prossimo->prossimo=NULL;
```

8. Si consideri il seguente programma `triangle.c`:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i,j,n=atoi(argv[1]);
    for(i=0; i<n; i++) {
        for(j=0; j<i+1; j++)
            printf("*");
        printf("\n");
    }
}
```

Una volta compilato, l'eseguibile accetta un intero sulla linea di comando e stampa un triangolo di asterischi nel modo seguente:

```
> ./triangle 5
*
**
***
****
*****
```

Si vuole ottenere lo stesso effetto con un altro programma `triangle_fork.c` che genera `n` processi figli (dove `n` è il valore dell'intero passato sulla linea di comando) facendo stampare una linea ad ogni figlio:

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

void print_row(int l) { // funzione che stampa una linea di l asterischi
    int i;
```

# Laboratorio di Sistemi Operativi

## 19 giugno 2014

### Compito

```
for(i=0;i<l;i++)
    printf("*");
    printf("\n");
}

int main(int argc, char *argv[]) {
    int i,j,n=atoi(argv[1]), *pid;
    pid=(int *)calloc(n,sizeof(int)); // alloco memoria per un array di n interi

    for(i=0; i<n; i++) {
        pid[i]=fork();           // creazione del figlio i-esimo
        switch(pid[i]) {
            case -1:           // fallimento della fork
                perror("fork failed!\n");
                exit(1);
            case 0:
                print_row(i+1); // il figlio i-esimo stampa la sua riga...
                exit(0);         // ... e termina la sua esecuzione
        }
    }

    for(i=0; i<n; i++) {
        waitpid(pid[i],NULL,0); // attendo la terminazione del figlio i-esimo
    }

    free(pid);      // libero la memoria allocata per il vettore di interi
    return 0;
}
```

- (a) (3 punti) il programma `triangle_fork.c` funzionerà correttamente, ovvero, produrrà in output il triangolo di asterischi? Perché?
- (b) (5 punti) Se si è risposto negativamente alla precedente domanda, modificare il programma in modo che rispetti la consegna.

#### Soluzione:

- (a) Il programma `triangle_fork.c` non è corretto, in quanto non è affatto detto che la sequenza di esecuzione dei processi figli sia la stessa della loro creazione. Quindi è possibile che le righe di asterischi vengano stampate fuori ordine.
- (b) Ci sono vari modi per correggere il programma: il più semplice è quello di aggiungere il caso di default allo `switch` in questo modo:

```
...
default:
    waitpid(pid[i],NULL,0);
```

eliminando poi il ciclo `for` finale con i `waitpid`. In questo modo il padre (che eseguirà per l'appunto il codice relativo al caso di default) attenderà la terminazione del processo figlio  $i$ -esimo (che stamperà la sua linea di asterischi) prima di lanciare il processo figlio  $(i+1)$ -esimo.

# Laboratorio di Sistemi Operativi

## 20 Gennaio 2021

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `check_file_tree.sh` che prenda come argomento sulla linea di comando un percorso, controlli che corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, stampando alla fine il numero di file e directory incontrati.

Esempio:

```
$ ./check_file_tree.sh /home/ivan
Numero di file: 65856
Numero di directory: 5318
```

Esempio di soluzione:

```
1 if ! test $# -eq 1
2 then
3     echo "Utilizzo $0 pathname"
4     exit 1
5 fi
6
7 num_file=0
8 num_dir=0
9
10 if test -e $1 -a -d $1
11 then
12     lista='find $1 -print 2>/dev/null'
13     for i in $lista
14     do
15         if test -f $i
16         then
17             num_file=$[num_file+1]
18         fi
19         if test -d $i
20         then
21             num_dir=$[num_dir+1]
22         fi
23     done
24 fi
25
26 echo "Numero di file: $num_file"
27 echo "Numero di directory: $num_dir"
28
29 exit 0
```

2. (6 punti) Si scriva uno script della shell `check_login.sh` che prenda sulla linea di comando una lista arbitraria di stringhe e controlli se ognuna di esse sia un nome di login valido. In caso positivo deve stampare lo user ID corrispondente, mentre in caso negativo deve stampare un messaggio che indichi che non è un nome di login valido.

Esempio:

```
$ ./check_login.sh ivan gdm system
ivan ha id 1000
gdm ha id 121
system non e' un nome di login valido
```

# Laboratorio di Sistemi Operativi

## 20 Gennaio 2021

### Compito

**Suggerimento:** si ricorra al file `/etc/passwd` dove ogni linea corrisponde ad un account del sistema ed è una successione di campi separati dai due punti (`:`). Il nome di login è rappresentato dal primo campo, mentre il terzo campo è lo user ID. Ad esempio:

```
pippo:x:1017:1019:Pippo,,,:/home/pippo:/bin/bash
```

Esempio di soluzione:

```
1 n=$#
2 for ((i=1; i<=$n; i++))
3 do
4     if grep "^\$1:" /etc/passwd 2>/dev/null >/dev/null
5     then
6         id='cat /etc/passwd | grep "^\$1:" 2>/dev/null | cut -d ":" -f3'
7         echo $1 ha id $id
8     else
9         echo "\$1 non e' un nome di login valido"
10    fi
11    shift
12 done
```

3. (6 punti) Si considerino le seguenti dichiarazioni in C:

```
1 int *a=(int*) malloc(10*sizeof(int));
2 int b[10];
```

Si dica, per ognuno dei seguenti comandi, se è lecito oppure no (motivando la risposta):

1. `a[5]=67;`
2. `*(a+5)=67;`
3. `b[5]=67;`
4. `*(b+5)=67;`
5. `a=b;`
6. `b=a;`

Risposte:

1. `a[5]=67;` è corretto: in generale, `a[i]` è sinonimo di `*(a+i)` e `a` è un puntatore ad interi che punta ad un'area di memoria allocata dinamicamente per contenere 10 interi; pertanto `a[5]` punterà all'indirizzo di memoria del sesto intero;
2. `*(a+5)=67;` è corretto: `a` è un puntatore ad interi che punta ad un'area di memoria allocata dinamicamente per contenere 10 interi; pertanto `a+5` punterà all'indirizzo di memoria del sesto intero;
3. `b[5]=67;` è corretto in quanto `b` è il nome di un vettore di 10 interi;
4. `*(b+5)=67;` è corretto: in generale, `*(b+i)` è sinonimo di `b[i]`;
5. `a=b;` è corretto: `a` è un puntatore a interi a cui viene assegnato l'indirizzo del primo elemento (di tipo intero) del vettore `b`;
6. `b=a;` non è corretto: il nome di un array non può apparire a sinistra di un comando di assegnamento in questo modo (è possibile assegnare soltanto i singoli elementi del vettore).

# Laboratorio di Sistemi Operativi

## 20 Gennaio 2021

### Compito

4. (4 punti) Si dica qual è l'output generato dal seguente programma C:

```
1 #include <stdio.h>
2
3 int main() {
4     int i, j;
5     for(i=0; i<3; i++) {
6         for(j=0; j<i; j++)
7             printf(" ");
8         for(j=0; j<(3-i)*2-1; j++)
9             printf("+");
10        printf("\n");
11    }
12
13    return 0;
14 }
```

```
+++++
+++
+
```

5. (10 punti) Si scriva un programma C che chieda all'utente quanti numeri interi pseudo-casuali vuole generare. Dopodiché il programma alloca memoria sufficiente per un array che possa contenere tali numeri ed inizia a generare un numero pseudo-casuale ogni 5 secondi. Alla fine stampa tutti i numeri generati ed il massimo fra questi e termina. Tuttavia, prima di giungere a terminazione, se l'utente preme Ctrl-C (ovvero, preme contemporaneamente i tasti Ctrl e C), il programma stampa il numero di interi generati fino a quel momento ed il massimo fra questi.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
                     attuale, assicurandosi di generare sequenze diverse ad ogni
                     esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
              tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <signal.h>
5 #include <unistd.h>
6
7 int compute_max=0;
8
9 void catchint(int signo) {
10     compute_max=1;
```

**Laboratorio di Sistemi Operativi**  
**20 Gennaio 2021**  
**Compito**

```
11 }
12
13 int main() {
14     int last_index, current_index, n;
15     long int *buf, max;
16     current_index=last_index=0;
17     buf=NULL;
18     max=-1;
19     srand(time(NULL));
20     signal(SIGINT, catchint);
21     printf("Inserisci il numero di interi da generare: ");
22     scanf("%d",&n);
23     buf=malloc(n*sizeof(long int));
24     while(current_index<n) {
25         if(compute_max) {
26             for(int i=last_index;i<current_index;i++)
27                 if(max<buf[i])
28                     max=buf[i];
29             printf("Numeri casuali generati finora: %d\n - massimo
30                   numero generato finora : %ld\n",current_index,max);
31             last_index=current_index;
32             compute_max=0;
33         }
34         else {
35             buf[current_index]=random();
36             current_index++;
37         }
38         sleep(5);
39     }
40     for(int i=0; i<n; i++) {
41         if(i>=last_index) {
42             if(max<buf[i])
43                 max=buf[i];
44         }
45         printf("buf[%d]=%ld\n",i,buf[i]);
46     }
47     printf("Il massimo è: %ld\n",max);
48     free(buf);
49     return 0;
50 }
```

# Laboratorio di Sistemi Operativi

## 20 Gennaio 2023

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `check_path` che prenda come argomento sulla linea di comando una stringa e stampi a video `OK` se quella stringa coincide con uno dei percorsi elencati nella variabile di ambiente `PATH`. Se non c'è nessuna occorrenza lo script non stampa nulla. Ad esempio (supponendo che `PATH=/bin:/usr/local/bin`):

```
> ./check_path /usr/local/bin
OK
> ./check_path bin
>
```

Esempio di soluzione:

```
if test -n "$1"
then
    if echo :$PATH: | grep ":$1:" >/dev/null 2>&1
    then
        echo OK
    fi
fi
```

2. (8 punti) Scrivere il codice di uno script della shell `bash` che prenda come parametro sulla linea di comando un percorso di un file di testo. Quest'ultimo deve essere composto da linee contenenti un carattere (fra `,`, `-`, `*`, `/`) e due interi, separati da virgole, ad esempio:

```
*,56,89
-,120,-1
+,12,12
+,30,5
/,5,2
```

Lo script deve leggere linea per linea il file e stampare a video il risultato dell'operazione, indicata dal carattere ad inizio linea, sui due numeri interi. Ad esempio per il caso precedente:

```
4984
121
24
35
2
```

Si gestiscono eventuali errori dovuti al parametro mancante o alla non esistenza/accessibilità in lettura del file.

Esempio di soluzione:

```
1 if ! test $# -eq 1
2 then
3     echo "Utilizzo: $0 pathname"
4     exit 1
5 fi
6
7 if ! test -f $1 -a -r $1
8 then
```

**Laboratorio di Sistemi Operativi**  
**20 Gennaio 2023**  
**Compito**

```
9 echo "File $1 non accessibile"
10 exit 2
11 fi
12
13 num_linee='cat $1 | wc -l'
14 i=1
15 while test $i -le $num_linee
16 do
17 linea='cat $1 | head -n +$i | tail -1'
18 op='echo $linea | cut -d , -f1'
19 num1='echo $linea | cut -d , -f2'
20 num2='echo $linea | cut -d , -f3'
21 echo ${num1} ${op} ${num2}
22 i=$((i+1))
23 done
24
25 exit 0
```

3. (8 punti) Scrivere il codice di un programma C che si comporti come lo script della shell dell'esercizio precedente.

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     if(argc!=2) {
5         fprintf(stderr,"Uso: %s pathname\n",argv[0]);
6         return 1;
7     }
8
9     FILE *f=fopen(argv[1],"r");
10    if(f==NULL) {
11        fprintf(stderr,"Il file %s non è accessibile.\n",argv[1]);
12        return 2;
13    }
14
15    char c;
16    int x,y;
17    while(fscanf(f,"%c,%d,%d",&c,&x,&y)==3) {
18        switch(c) {
19            case '+':
20                printf("%d\n",x+y);
21                break;
22            case '-':
23                printf("%d\n",x-y);
24                break;
25            case '*':
26                printf("%d\n",x*y);
27                break;
28            case '/':
29                printf("%d\n",x/y);
30                break;
31        }
32    }
}
```

# Laboratorio di Sistemi Operativi

## 20 Gennaio 2023

### Compito

```
33     fclose(f);
34     return 0;
35 }
```

4. (10 punti) Si consideri il problema classico dei produttori e consumatori con memoria limitata. Ci sono **NUM\_P** thread produttori e **NUM\_C** thread consumatori, che accedono in modo concorrente al vettore condiviso **buffer** con **LENGTH** posizioni per altrettanti interi positivi (con valori da 1 a **MAX**). Il valore -1 indica che la posizione del vettore è libera (vuota). Un produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera, generandolo casualmente. Un consumatore (se il buffer non è vuoto) preleva un elemento dalla prima posizione non vuota.

Supponendo di avere a disposizione:

1. la funzione **full()** che restituisce 1 se il buffer è pieno e 0 altrimenti,
2. la funzione **empty()** che restituisce 1 se il buffer è vuoto e 0 altrimenti,
3. **pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;** (mutex per l'accesso esclusivo),
4. **pthread\_cond\_t not\_empty\_buffer=PTHREAD\_COND\_INITIALIZER;** (condition variable: buffer non vuoto),
5. **pthread\_cond\_t not\_full\_buffer=PTHREAD\_COND\_INITIALIZER;** (condition variable: buffer non pieno).

si fornisca il codice sorgente delle funzioni **void \*producer(void \*)** (eseguita dai thread produttori) e **void \*consumer(void \*)** (eseguita dai thread consumatori).

```
1 int buffer[LENGTH];           // buffer condiviso di lunghezza LENGTH
2
3 // mutex per l'accesso esclusivo
4 pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
5 // condition variable: buffer non vuoto
6 pthread_cond_t not_empty_buffer=PTHREAD_COND_INITIALIZER;
7 // condition variable: buffer non pieno
8 pthread_cond_t not_full_buffer=PTHREAD_COND_INITIALIZER;
9
10 void *producer(void *n) {
11     int elemento, i;
12     while(1) {
13         pthread_mutex_lock(&mutex); // <— inizio sezione critica
14         while(full()) pthread_cond_wait(&not_full_buffer,&mutex); // <—
15             controllo se posso inserire un nuovo elemento
16         for(i=0; i<LENGTH; i++)
17             if(buffer[i]==-1) {
18                 elemento=random()%MAX+1; // genero l'elemento
19                 buffer[i]=elemento;    // inserisco l'elemento
20                 break;
21             }
22         pthread_cond_signal(&not_empty_buffer); // <— segnalo che il buffer
23             non è vuoto
24         pthread_mutex_unlock(&mutex); // <— fine sezione critica
25     }
26 };
27
28 void *consumer(void *n) {
29     int elemento, i;
30     while(1) {
31         pthread_mutex_lock(&mutex); // <— inizio sezione critica
```

**Laboratorio di Sistemi Operativi**  
**20 Gennaio 2023**  
**Compito**

```
30     while(empty()) pthread_cond_wait(&not_empty_buffer,&mutex); // <—
           controllo se posso prelevare un elemento
31     for(i=0; i<LENGTH; i++)
32       if(buffer[i]!=-1) {
33         elemento=buffer[i]; // prelevo l'elemento
34         buffer[i]=-1;      // indico che la posizione è libera
35         break;
36       }
37     pthread_cond_signal(&not_full_buffer); // <— segnalo che il buffer
           non è pieno
38     pthread_mutex_unlock(&mutex); // <— fine sezione critica
39   }
40 };
```

# Laboratorio di Sistemi Operativi

## 21 Giugno 2021

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (6 punti) Si scriva uno script della shell `check_file_tree_size.sh` che prenda come argomento sulla linea di comando un percorso, controlli che corrisponda ad una directory e attraversi ricorsivamente il file system a partire da essa, stampando alla fine il numero dei soli file incontrati e del totale di byte occupati da essi.

Esempio:

```
./check_file_tree_size.sh .
Numero di file: 7
Numero di byte: 52493
```

Esempio di soluzione:

```
1 f ! test $# -eq 1
2 then
3     echo "Utilizzo $0 pathname"
4     exit 1
5 fi
6
7 num_file=0
8 num_bytes=0
9
10 if test -e $1 -a -d $1
11 then
12     lista='find $1 -print 2>/dev/null'
13     for i in $lista
14     do
15         if test -f $i
16         then
17             num_file=$((num_file+1))
18             dim=$(cat $i | wc -c)
19             num_bytes=$((num_bytes+$dim))
20         fi
21     done
22 fi
23
24 echo "Numero di file: $num_file"
25 echo "Numero di byte: $num_bytes"
26
27 exit 0
```

2. (6 punti) Si scriva una pipeline che stampi a video l'elenco (senza ripetizioni e ordinato lessicograficamente) delle shell di login assegnate agli utenti di sistema.

**Suggerimento:** si ricorra al file `/etc/passwd` dove ogni linea corrisponde ad un account del sistema ed è una successione di campi separati dai due punti (:). Il campo relativo alla shell di login è il settimo.

Esempio di soluzione:

```
1 cat /etc/passwd | cut -d ':' -f7 | sort | uniq
```

3. (6 punti) Si considerino le seguenti dichiarazioni in C:

# Laboratorio di Sistemi Operativi

## 21 Giugno 2021

### Compito

```
1 int *a=(int*)malloc(10*sizeof(int));  
2 int b[10];
```

Si dica, per ognuno dei seguenti comandi, se è lecito oppure no (motivando la risposta):

1. a[4+5]=67;
2. \*(a+4+5)=67;
3. b[10]=68;
4. \*(b+10)=68;
5. a=b+1;
6. b=a+1;

Risposte:

1. a[4+5]=67; è corretto: in generale, a[i] è sinonimo di \*(a+i) e a è un puntatore ad interi che punta ad un'area di memoria allocata dinamicamente per contenere 10 interi; pertanto a[4+5] punterà all'indirizzo di memoria dell'ultimo intero (a[9]);
2. \*(a+4+5)=67; è corretto: a è un puntatore ad interi che punta ad un'area di memoria allocata dinamicamente per contenere 10 interi; pertanto a+4+5 punterà all'indirizzo di memoria del decimo intero (ovvero, l'ultimo);
3. b[10]=68; è scorretto in quanto b è il nome di un vettore di 10 interi e si sta tentando di accedere all'undicesimo;
4. \*(b+10)=67; è scorretto: in generale, \*(b+i) è sinonimo di b[i] (vedi punto precedente);
5. a=b+1; è corretto: a è un puntatore a interi a cui viene assegnato l'indirizzo del secondo elemento (di tipo intero) del vettore b;
6. b=a+1; non è corretto: il nome di un array non può apparire a sinistra di un comando di assegnamento in questo modo (è possibile assegnare soltanto i singoli elementi del vettore).

4. (4 punti) Si dica qual è l'output generato dal seguente programma C:

```
1 #include <stdio.h>  
2  
3 int main() {  
4     int i,j;  
5     for(i=2; i>=0; i--) {  
6         for(j=0; j<i; j++)  
7             printf(" ");  
8         for(j=0; j<(3-i)*2-1; j++)  
9             printf("*");  
10        printf("\n");  
11    }  
12  
13    return 0;  
14 }
```

```
*  
***  
*****
```

# Laboratorio di Sistemi Operativi

## 21 Giugno 2021

### Compito

5. (10 punti) Si scriva un programma C `dice.c` che simuli il lancio di un dado. Il programma prende da linea di comando il numero di volte `n` che il punteggio massimo possibile (ovvero 6) deve uscire come esito del lancio. Dopodiché inizia a generare un numero pseudo-casuale compreso tra 1 e 6 ogni 5 secondi.

Raggiunto il numero `n` di lanci con punteggio massimo, termina la propria esecuzione stampando il messaggio `Si sono verificati n lanci con punteggio massimo.`

Tuttavia, prima di giungere a terminazione, se l'utente preme Ctrl-C (ovvero, preme contemporaneamente i tasti Ctrl e C), il programma termina immediatamente, stampando il numero di lanci con punteggio massimo eventualmente generati fino a quel momento.

**Esempio:** se l'utente digita

```
./dice 7
```

e non preme Ctrl-C, il programma si fermerà dopo che il numero 6 sarà uscito per la settima volta, stampando

`Si sono verificati 7 lanci con punteggio massimo.`

Altrimenti uscirà al momento della pressione di Ctrl-C stampando il numero di volte che il numero 6 sarà uscito fino a quel momento.

**Suggerimento:** si ricorda che, per generare dei numeri pseudo-casuali, è sufficiente utilizzare la funzione di libreria `random()` come segue:

```
1 #include <stdlib.h>
2 #include <time.h>
3 ...
4 srand(time(NULL)); // per inizializzare il seme con la data/ora
    attuale , assicurandosi di generare sequenze diverse ad ogni
    esecuzione
5 ...
6 long int r;
7 r=random(); // genera un numero pseudo-casuale con valore compreso
    tra 0 e RAND_MAX e lo assegna alla variabile r
```

Esempio di soluzione:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <signal.h>
5 #include <unistd.h>
6
7 int stop=0;
8
9 void catchint(int signo) {
10     stop=1;
11 }
12
13 int main(int argc, char** argv) {
14     long int max_count=atoi(argv[1]);
15     long int max_current_count=0;
16     srand(time(NULL));
17     signal(SIGINT, catchint);
18     while(max_current_count<max_count && !stop) {
19         long int r=random();
20         int score=r%6+1;
21         if(score==6)
```

**Laboratorio di Sistemi Operativi**  
**21 Giugno 2021**  
**Compito**

```
22     max_current_count++;
23     printf("Punteggio dell'ultimo lancio: %d.\n", score);
24     sleep(5);
25 }
26
27 printf("Si sono verificati %ld lanci con punteggio massimo.\n",
28        max_current_count);
29
30 return 0;
31 }
```

# Laboratorio di Sistemi Operativi

## 28 Giugno 2019

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che, partendo dalla directory corrente, stampi ricorsivamente i nomi delle directory leggibili ed attraversabili da tutte le categorie di utenti (ovvero, proprietario, gruppo e resto degli utenti).

```
find . -type d -ls | grep 'r.xr.xr.x'
```

2. (6 punti) si scriva uno script `explore.sh` della shell che accetti come argomento sulla linea di comando il percorso di una directory e la esplori ricorsivamente, stampando alla fine i valori di due contatori: uno relativo al numero di directory incontrate ed uno relativo al numero di file regolari incontrati.

Esempio:

```
> ./explore.sh /home/pippo
number of dirs: 10
number of files: 23
> _
```

```
1 if test $# -ne 1
2 then
3     echo "Wrong number of arguments!"
4     exit 1
5 fi
6
7 if ! test -d $1
8 then
9     echo "$1 must be a directory!"
10    exit 2
11 fi
12
13 find $1 -type d > /tmp/dirs.txt
14 find $1 -type f > /tmp/files.txt
15
16 dirs='cat /tmp/dirs.txt | wc -l'
17 files='cat /tmp/files.txt | wc -l'
18
19 echo "number of directories: $dirs"
20 echo "number of files: $files"
21
22 rm /tmp/dirs.txt
23 rm /tmp/files.txt
24
25 exit 0
```

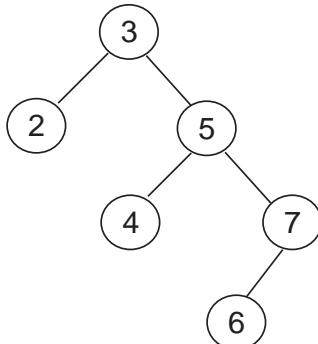
3. (8 punti) Data la seguente struttura dati ricorsiva `Node`:

```
struct node_t {
    int data;
    struct node_t *left;
    struct node_t *right;
};
```

**Laboratorio di Sistemi Operativi**  
**28 Giugno 2019**  
**Compito**

```
typedef struct node_t Node;
```

si scriva un programma C che allochi spazio in memoria ed inizializzi il seguente albero binario (con radice nel nodo 3):



Si scriva infine una procedura ricorsiva che riceva come parametro il puntatore alla radice di un albero binario (come, ad esempio, quello precedente) e ne liberi la memoria occupata dai vari nodi.

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

struct node_t {
    int data;
    struct node_t *left;
    struct node_t *right;
};

typedef struct node_t Node;

void deltree(Node *r) {
    if(r!=NULL) {
        if(r->left!=NULL) deltree(r->left);
        if(r->right!=NULL) deltree(r->right);
        free(r);
    }
}

int main() {
    Node *root=NULL;
    root=(Node*)malloc(sizeof(Node));
    root->data=3;
    root->left=(Node*)malloc(sizeof(Node));
    root->right=(Node*)malloc(sizeof(Node));
    root->left->data=2;
    root->left->left=NULL;
    root->left->right=NULL;
    root->right=(Node*)malloc(sizeof(Node));
    root->right->data=5;
    root->right->left=(Node*)malloc(sizeof(Node));
    root->right->right=(Node*)malloc(sizeof(Node));
    root->right->left->data=4;
```

# Laboratorio di Sistemi Operativi

## 28 Giugno 2019

### Compito

```
root->right->left->left=NULL;
root->right->left->right=NULL;
root->right->right->data=7;
root->right->right->left=(Node*)malloc(sizeof(Node));
root->right->right->right=NULL;
root->right->right->left->data=6;
root->right->right->left->left=NULL;
root->right->right->left->right=NULL;
deltree(root);
return 0;
}
```

4. (6 punti) Si dica qual è l'output generato dal seguente programma C:

```
1 #include <stdio.h>
2
3 int main() {
4     int i,j;
5     for(i=0; i<3; i++) {
6         for(j=0; j<i; j++)
7             printf(" ");
8         for(j=0; j<(3-i)*2-1; j++)
9             printf("|");
10        printf("\n");
11    }
12
13    return 0;
14 }
```

```
|||||
|||
|
```

5. (8 punti) Il programma seguente utilizza i thread, i mutex e le condition variable, per implementare una soluzione al problema classico dei produttori e consumatori con memoria limitata. Ci sono **NUM\_P** thread produttori e **NUM\_C** thread consumatori, che accedono in modo concorrente al vettore condiviso **buffer** con **LENGTH** posizioni per altrettanti interi positivi (con valori da 1 a **MAX**). Il valore -1 indica che la posizione del vettore è libera (vuota). Un produttore (se il buffer non è pieno) inserisce un nuovo elemento nella prima posizione libera. Un consumatore (se il buffer non è vuoto) preleva un elemento dalla prima posizione non vuota.

Supponendo di avere a disposizione

- la funzione **full()** che restituisce 1 se il buffer è pieno e 0 altrimenti,
  - la funzione **empty()** che restituisce 1 se il buffer è vuoto e 0 altrimenti,
- si completi il sorgente negli 8 punti indicati, affinché il codice risultante rappresenti una soluzione corretta del problema.

```
int buffer[LENGTH];           // buffer condiviso di lunghezza LENGTH
pthread_t threadP[NUM_P];   // vettore che contiene gli ID dei thread produttori
pthread_t threadC[NUM_C];   // vettore che contiene gli ID dei thread consumatori

// mutex per l'accesso esclusivo
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
// condition variable: buffer non vuoto
```

# Laboratorio di Sistemi Operativi

## 28 Giugno 2019

### Compito

```
pthread_cond_t not_empty_buffer=PTHREAD_COND_INITIALIZER;
// condition variable: buffer non pieno
pthread_cond_t not_full_buffer=PTHREAD_COND_INITIALIZER;

void *producer(void *n) {
    int elemento, i;
    while(1) {
        printf("Thread produttore (ID %lu)\n",threadP[((int)n)-1]);
        ... // <-- inizio sezione critica: completare (1)
        ... // <-- controllo se posso inserire un nuovo elemento: completare (2)
        for(i=0; i<LENGTH; i++)
            if(buffer[i]==-1) {
                elemento=random()%MAX+1; // genero l'elemento
                buffer[i]=elemento;      // inserisco l'elemento
                break;
            }
        ... // <-- quale evento devo segnalare qui? completare (3)
        ... // <-- fine sezione critica: completare (4)
    }
};

void *consumer(void *n) {
    int elemento, i;
    while(1) {
        printf("Thread consumatore (ID %lu)\n",threadC[((int)n)-1]);
        ... // <-- inizio sezione critica: completare (5)
        ... // <-- controllo se posso prelevare un elemento (6)
        for(i=0; i<LENGTH; i++)
            if(buffer[i]!=-1) {
                elemento=buffer[i]; // prelevo l'elemento
                buffer[i]=-1;       // segnalo che la posizione è libera
                break;
            }
        ... // <-- quale evento devo segnalare qui? completare (7)
        ... // <-- fine sezione critica: completare (8)
    }
};
```

1. `pthread_mutex_lock(&mutex);`
2. `if(full()) pthread_cond_wait(&not_full_buffer,&mutex);`
3. `pthread_cond_signal(&not_empty_buffer);`
4. `pthread_mutex_unlock(&mutex);`
5. `pthread_mutex_lock(&mutex);`
6. `if(empty()) pthread_cond_wait(&not_empty_buffer,&mutex);`
7. `pthread_cond_signal(&not_full_buffer);`
8. `pthread_mutex_unlock(&mutex);`

# Laboratorio di Sistemi Operativi

## 30 Gennaio 2019

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (4 punti) Si scriva una pipeline che, dato un file di testo con nome **test.txt**, stampi le sue linee, evitando le ripetizioni di linee che occorrono più volte, e (prima di ogni linea) il conteggio delle sue occorrenze nel file. Ad esempio, supponendo che il file **test.txt** sia formato dalle seguenti linee:

```
abc
abc
def
abc
ghi
ghi
ghi
abc
```

la pipeline dovrà produrre quanto segue:

```
4 abc
1 def
3 ghi
```

```
sort test.txt | uniq -c
```

2. (6 punti) si scriva uno script **dim.sh** della shell che legga delle linee di testo dallo standard input fintanto che l'utente non digiti una linea di terminazione composta dalla sola parola **fine**. Per ogni linea diversa da quella di terminazione, lo script deve stampare a video la dimensione in byte della linea ed una serie di asterischi di lunghezza pari alla dimensione.

Esempio:

```
> ./dim.sh
abc
3 byte
***
ciao, mondo!
12 byte
*****
fine
> _
```

```
1 while true
2 do
3     read linea
4
5     if test $linea = 'fine'
6     then
7         exit 0
8     else
9         dim='echo $linea | wc -c'
10        dim=${dim%?} # escludo dal conteggio il newline
11        i=0;
12
13        while test $i -lt $dim
14        do
```

**Laboratorio di Sistemi Operativi**  
**30 Gennaio 2019**  
**Compito**

```
15      echo -n '*'
16      i=$[$i+1]
17      done
18
19      if test $dim -gt 0
20      then
21          echo
22      fi
23
24  fi
25
26 done
```

3. (8 punti) Data la seguente struttura dati ricorsiva **Node**:

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

typedef struct node_t Node;
```

si scriva un programma C che allochi spazio in memoria ed inizializzi una lista finita di naturali con i seguenti valori:

11, 6, 30

Si scriva infine una procedura ricorsiva per stampare a video i valori dei nodi di una lista di naturali, partendo dal puntatore alla testa della lista.

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

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

typedef struct node_t Node;

void print_list(Node *h);

int main() {
    Node *head=NULL;
    head=(Node*)malloc(sizeof(Node));
    head->data=11;
    head->next=(Node*)malloc(sizeof(Node));
    head->next->data=6;
    head->next->next=(Node*)malloc(sizeof(Node));
    head->next->next->data=30;
    head->next->next->next=NULL;
    print_list(head);
    return 0;
}
```

**Laboratorio di Sistemi Operativi**  
**30 Gennaio 2019**  
**Compito**

```
}

void print_list(Node *h) {
    if(h!=NULL) {
        printf(" %d ",h->data);
        print_list(h->next);
    }
}
```

4. (6 punti) Scrivere l'output del seguente programma C.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int levels=5;
6
7     for (int i = 0; i < levels; i++) {
8         for (int j = 0; j < levels - i; j++){
9             printf("-");
10            }
11            for (int k = 0; k < (2 * i + 1); k++){
12                if(i>0 && i<levels-1) {
13                    if(k>0 && k < (2 * i + 1) - 1)
14                        printf(" ");
15                    else
16                        printf("*");
17                }
18                else
19                    printf("*");
20            }
21            printf("\n");
22        }
23        return 0;
24 }
```

```
-----*
----* *
---*   *
--*   *
-*****
```

5. (8 punti) La procedura `logger` seguente è la start routine di una famiglia di POSIX thread che accedono concorrentemente in scrittura al file `log.txt`.

```
1 struct channel {
2     int fd; /* file descriptor del canale di comunicazione */
3     int num; /* numero intero identificante il thread */
4 };
5
6 void *logger(void *c) {
7     char inputline[LINESIZE];
8     char buffer[2*LINESIZE];
```

# Laboratorio di Sistemi Operativi

## 30 Gennaio 2019

### Compito

```
9  int len;
10 int fd=((struct channel *)c)->fd;
11 int num=((struct channel *)c)->num;
12 FILE *logfd;
13 /* finché ricevo linee da fd... */
14 while ((len = recv(fd, inputline, LINESIZE-1, 0)) > 0) {
15     inputline[len]='\0';           /* imposto il terminatore */
16     sprintf(buffer,"Thread n. %d: ",num); /* scrivo il n. del thread in buffer */
17     strcat(buffer,inputline);      /* concateno in buffer quanto ricevuto da fd */
18     logfd=fopen("log.txt","a");    /* apro il file log.txt in append */
19     fputs(buffer,logfd);          /* scrivo nel file */
20     fclose(logfd);              /* chiudo il file log.txt */
21 }
22
23 close(fd);
24 }
```

- (a) C'è il pericolo che si verifichino delle race condition, con un possibile accavallamento delle linee di un thread con quelle di altri thread nel file `log.txt`?
- (b) In caso di risposta affermativa alla domanda precedente, specificare come modificare la routine in modo da garantire l'integrità del file condiviso, aggiungendo le dichiarazioni ed i comandi necessari.

(a) Sì, c'è il pericolo che si verifichino delle race condition, con la conseguente "corruzione" del contenuto di `log.txt` in quanto tra le scritture nel file da parte di un thread potrebbero intervenire altre operazioni di scrittura da parte di altri thread della stessa famiglia, accavallando le linee.

(b) Bisogna introdurre un mutex globale:

```
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

Dopodiché bisogna proteggere il ciclo while nel modo seguente:

void *logger(void *c) {
    char inputline[LINESIZE];
    char buffer[2*LINESIZE];
    int len;
    int fd=((struct channel *)c)->fd;
    int num=((struct channel *)c)->num;
    FILE *logfd;

    pthread_mutex_lock(&mutex);          /* inizio sezione critica */

    while ((len = recv(fd, inputline, LINESIZE-1, 0)) > 0) {
        inputline[len]='\0';
        sprintf(buffer,"Thread n. %d: ",num);
        strcat(buffer,inputline);
        logfd=fopen("log.txt","a");
        fputs(buffer,logfd);
        fclose(logfd);
    }

    pthread_mutex_unlock(&mutex);        /* fine sezione critica */

    close(fd);
}
```

# 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,$y/dc/zy/'`
- (b) (2 punti) `echo ababab | sed '1,$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".

# Laboratorio di Sistemi Operativi

## 22 settembre 2014

### Compito

- (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 whitespace 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 richiamata 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)  
4          oppure no (0) */  
5     int word_begin=0;  
6     /* last è l'indice del prossimo carattere da processare */  
7     static int last=0;  
8     /* Controllo se è il caso di resettare la scansione della stringa  
9      */
```

# Laboratorio di Sistemi Operativi

## 22 settembre 2014

### Compito

```
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] == '\u0020' ||
17             inputline[i] == '\t' ||
18             inputline[i] == '\n') {
19             /* se word_begin vale 1, allora sono arrivato alla fine della
20                parola, imposto il terminatore ed esco dal ciclo */
21             if (word_begin) {
22                 word[j]='\0';
23                 /* la prossima volta inizio la scansione dal carattere
24                  successivo */
25                 last=i+1;
26                 break;
27             }
28         else {
29             /* non \u00e8 un carattere whitespace quindi inizia una parola e
30                copio il carattere in word */
31             word_begin=1;
32             word[j++]=inputline[i];
33         }
34     }
35
36     return j;
37 }
```

5. (10 punti) Si completi il seguente frammento di programma C in cui \u00e8 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)) <
5          0) {
6          perror("accepting connection");
7          exit(1);
8      }
9
10     /* ogni volta che il server accetta una nuova connessione,
11        quest'ultima viene gestita da un nuovo processo figlio
12        */
13     switch(fork()) {
14         case -1:
15             perror("Errore nella chiamata alla fork");
16             exit(2);
17         case 0:
18             fprintf(stderr, "Aperta connessione (PID %d).\n", getpid());
19             ...
20             fprintf(stderr, "Chiusa connessione (PID %d).\n", getpid());
21             exit(0);
22     }
23 }
```

# Laboratorio di Sistemi Operativi

## 22 settembre 2014

### Compito

```
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
11           split dell'esercizio precedente e le invio al client, ognuna
12           seguita da un newline */
13        while((i=split(inputline, outputline, len))!=0) {
14            send(out, outputline, i, 0);
15            send(out, "\n", 1, 0);
16        }
17    }
18 }
```

# Laboratorio di Sistemi Operativi

## 22 settembre 2015

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. (a) Cosa si intende con ridirezione da linea di comando (altrimenti nota come “here document”)?  
(b) Si faccia un esempio di utilizzo della ridirezione da linea di comando.

#### Soluzione:

(a) Con l'espressione "ridirezione da linea di comando" (i.e., *here document*) si intende la possibilità di fornire in input ad un comando od uno script del testo "catturato" dallo standard input (i.e., dalla tastiera) come se provenisse da un file.

(b) Ecco un esempio di *here document*:

```
> wc <<fine_esercizio
? se non so rispondere
? a questa domanda
? forse mi conviene ritirarmi...
? fine_esercizio
3 11 69
```

Praticamente quanto segue l'operatore << assume il ruolo di marcatore di fine input: la shell tiene traccia di tutto quanto viene digitato fintanto che non incontra il marcatore su una linea isolata. A questo punto fornisce il testo raccolto (escluso il marcatore) come input al comando (in questo caso `wc`) come se provenisse da un normale file di testo, per farlo processare.

2. Qual è l'effetto dei seguenti comandi?

1. `ls -l | grep '^-'`
2. `n='wc -l registro.txt | cut -d' ' -f1'` (si supponga che l'output di `wc -l registro.txt` sia la stringa 10 `registro.txt`);
3. `echo $n` (dove `n` è la variabile inizializzata nel punto precedente).

**Attenzione:** nel punto 2 gli apici esterni sono dei *backquote* (apici rovesciati).

#### Soluzione:

1. Il comando `ls -l | grep '^-'` seleziona dall'output di `ls -l` soltanto le linee che iniziano con il carattere -. Quindi vengono stampate a video soltanto le linee che corrispondono a file ordinari (i.e., non directory, link ecc.) in quanto il trattino è il carattere che li contraddistingue nell'output di `ls -l`.
2. Il comando `n='wc -l registro.txt | cut -d' ' -f1'` opera una *command substitution* assegnando alla variabile `n` il risultato di `wc -l registro.txt | cut -d' ' -f1`, ovvero, 10 in questo caso (infatti `cut`, utilizzando come separatore di campo lo spazio, seleziona il primo campo della stringa prodotta da `wc -l registro.txt`, ovvero, `texttt10`).
3. Il comando `echo $n` stampa a video il valore della variabile `n` assegnato al punto precedente, ovvero, 10.

3. Si predisponga uno script della shell `line_ith.sh` che prenda come argomento sulla linea di comando il percorso di un file di testo ed un intero e compia le seguenti azioni:
  - (a) controlli il numero degli argomenti forniti, terminando la propria esecuzione nel caso il numero sia diverso da due;
  - (b) controlli che il percorso fornito come primo argomento corrisponda ad un file e sia leggibile dall'utente (altrimenti l'esecuzione deve terminare);

# Laboratorio di Sistemi Operativi

## 22 settembre 2015

### Compito

- (c) controlli che il numero intero fornito come secondo argomento sia un numero compreso fra 1 ed il numero di linee del file corrispondente al primo argomento;
- (d) tra le righe del file passato come primo argomento stampi su standard output quella corrispondente al numero passato come secondo argomento.

Ad esempio il comando

```
> line_ith testo.txt 5
```

deve emettere su standard output la quinta riga del file `testo.txt`.

**Suggerimento:** si ricordi che `tail -n +k f` stampa le linee di `f`, a partire dalla `k`-esima, mentre `head -n +k f` stampa le prime `k` linee di `f`.

#### Soluzione:

Esempio di soluzione:

```
1 #!/bin/bash
2
3 if test $# -ne 2
4 then
5   echo "Utilizzo dello script: $0 <file> <n. di linea>"
6   exit 1
7 fi
8
9 if ! test -f $1 -a -r $1
10 then
11   echo "$1 non è un file oppure non è leggibile."
12   exit 2
13 fi
14
15 num_linee='wc -l $1 | cut -d " " -f1'
16
17 if ! test $2 -ge 1 -a $2 -le $num_linee
18 then
19   echo "$2 non è un numero compreso fra 1 e $num_linee."
20   exit 3
21 fi
22
23 tail -n +$2 $1 | head -n +1
24
25 exit 0
```

4. Sia data la seguente struttura ricorsiva in C per la rappresentazione di alberi binari:

```
struct bintree {
    int val;
    struct bintree *left;
    struct bintree *right;
};
```

dove `val` rappresenta il valore del nodo, mentre `left` e `right` puntano, rispettivamente, al figlio sinistro ed al figlio destro (tali puntatori assumono il valore `NULL` quando non esistono i rispettivi figli).

Si scriva il codice di una funzione avente il seguente prototipo:

# Laboratorio di Sistemi Operativi

## 22 settembre 2015

### Compito

```
void raddoppia(struct bintree *root);
```

che raddoppi il valore del membro `val` di ogni nodo dell'albero binario puntato da `root`.

#### Soluzione:

Esempio di soluzione:

```
1 void raddoppia (struct bintree *root) {
2     if (root!=NULL) {
3         root->val *= 2;
4         raddoppia (root->left);
5         raddoppia (root->right);
6     }
7 }
```

5. Si consideri il seguente programma `triangle.c`:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i,j,n=atoi(argv[1]);
    for(i=0; i<n; i++) {
        for(j=0; j<i+1; j++)
            printf("*");
        printf("\n");
    }
}
```

Una volta compilato, l'eseguibile accetta un intero sulla linea di comando e stampa un triangolo di asterischi nel modo seguente:

```
> ./triangle 5
*
**
***
****
*****
```

Si vuole ottenere lo stesso effetto con un altro programma `triangle_thread.c` che genera `n` thread figli (dove `n` è il valore dell'intero passato sulla linea di comando) facendo stampare una linea ad ogni figlio:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/types.h>

void *print_row(void *n) { // funzione che stampa una linea di n asterischi
    int i;
    sleep(random()%5+1); // attende da 1 a 5 secondi
    for(i=0;i<((int)n);i++)
        printf("*");
    printf("\n");
```

# Laboratorio di Sistemi Operativi

## 22 settembre 2015

### Compito

```
}
```

```
int main(int argc, char *argv[]) {
    int i,j,n=atoi(argv[1]);
    pthread_t *thr;
    thr=(pthread_t *)calloc(n,sizeof(pthread_t)); // alloco memoria per un array di n thread id

    for(i=1; i<=n; i++) {
        if(pthread_create(&thr[i-1],NULL,print_row,(void *)i)!=0) { // creo il thread i-esimo
            perror("Errore nella creazione del thread.\n");
            exit(1);
        }
    }

    for(i=1;i<=n;i++)
        pthread_join(thr[i-1],NULL); // attendo la terminazione dell'i-esimo thread figlio

    free(thr); // libero la memoria allocata per il vettore di thread id

    return 0;
}
```

- (a) Il programma `triangle_thread.c` funzionerà correttamente, ovvero, produrrà in output il triangolo di asterischi? Perché?
- (b) Se si è risposto negativamente alla precedente domanda, modificare (si può cancellare/modificare/aggiungere codice a piacimento) il programma in modo che rispetti la consegna.

#### Soluzione:

- (a) Il programma `triangle_thread.c` non funzionerà correttamente in quanto la funzione `print_row()` introduce dei ritardi casuali prima di stampare gli asterischi di una linea. Inoltre, anche eliminando tali ritardi, non è affatto detto che il sistema esegua i singoli thread nell'ordine in cui sono stati creati.
- (b) È possibile apportare diverse correzioni (eliminare i ritardi e rendere i thread FIFO oppure usare una variabile che rappresenti il turno del thread corretto ed utilizzare mutex e condition variable per sincronizzare le stampe delle singole linee). Tuttavia la soluzione più semplice consiste nell'attendere la terminazione del thread appena creato prima di procedere alla creazione del successivo. Per far ciò, è sufficiente eliminare il ciclo `for` finale con `i join ai vari thread ed introdurre una chiamata a pthread_join all'interno del primo ciclo for come segue:`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <pthread.h>
5 #include <fcntl.h>
6 #include <sys/types.h>
7
8 void *print_row(void *n) { // funzione che stampa una linea di n asterischi
9     int i;
10    sleep(random()%5+1); // attende da 1 a 5 secondi
11    for(i=0;i<((int)n);i++)
12        printf("*");
13    printf("\n");
14 }
```

# Laboratorio di Sistemi Operativi

## 22 settembre 2015

### Compito

```
15
16 int main(int argc, char *argv[]) {
17     int i,j,n=atoi(argv[1]);
18     pthread_t *thr;
19     thr=(pthread_t *)calloc(n,sizeof(pthread_t)); // alloco memoria
19     per un array di n thread id
20
21     for(i=1; i<=n; i++) {
22         if(pthread_create(&thr[i-1],NULL,print_row,(void *)i)!=0) {
22             // creo il thread i-esimo
23             perror("Errore nella creazione del thread.\n");
24             exit(1);
25         }
26         pthread_join(thr[i-1],NULL); // attendo la terminazione dell'
26         i-esimo thread figlio
27     }
28
29     free(thr); // libero la memoria allocata
29     per il vettore di thread id
30
31     return 0;
32 }
```

# Laboratorio di Sistemi Operativi

## 23 giugno 2015

### Compito

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Si illustri la differenza fra link simbolico e link hard.

#### Soluzione:

In UNIX ogni entry di una directory è un link e rappresenta l'associazione fra un nome di file ed il corrispondente indice nell'array degli inode del dispositivo. Un link hard (creabile con il comando `ln`) ad un dato file è semplicemente un alias per lo stesso numero di inode del file. Al contrario, un link simbolico (creabile con il comando `ln -s`) ad un dato file è un file di testo (con un proprio inode diverso da quello del file a cui punta) trattato in modo speciale dal sistema operativo. Il file di testo contiene il path assoluto del file "puntato". Come conseguenza l'accesso tramite link hard ai file è molto più veloce rispetto all'accesso tramite link simbolici, ma questi ultimi consentono di far riferimento anche a file che risiedano su dispositivi e partizioni diversi.

2. Qual è l'effetto dei seguenti comandi?

1. `f=~/bash_profile`
2. `echo 'basename $f'`
3. `echo "basename $f"`
4. `echo 'basename $f'`

**Attenzione:** nel punto 4 gli apici sono dei *backquote* (apici rovesciati).

#### Soluzione:

L'effetto dei comandi è il seguente:

1. `f=~/bash_profile` assegna alla variabile `f` la stringa `/home/username/.bash_profile` dove `username` è il nome dell'account dell'utente che ha lanciato il comando.
2. `echo 'basename $f'` visualizza su standard output la stringa `basename $f` in quanto gli apici inibiscono l'interpretazione dei metacaratteri (compreso il `$`).
3. `echo "basename $f"` visualizza su standard output la stringa `basename /home/username/.bash_profile` dove `username` è il nome dell'account dell'utente che ha lanciato il comando (i doppi apici permettono l'interpretazione del metacarattere `$`).
4. `echo 'basename $f'` visualizza su standard output la stringa `.bash_profile` ovvero l'output del comando racchiuso tra gli apici rovesciati.

3. L'output seguente mostra un frammento del contenuto del file `/etc/passwd`:

```
root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/adm:/bin/false
lp:x:4:7:lp:/var/spool/lpd:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
```

Ogni linea contiene informazioni su un account del sistema in uso e tali informazioni sono formattate in campi separati dai due punti (`:`). Si scriva una successione di comandi che fornisca in output la lista dei soli nomi degli account (primo campo) e della relativa home directory (sesto campo) ordinati lessicograficamente in modo crescente in base al primo campo. Ad esempio, per il frammento precedente l'output deve essere il seguente:

# Laboratorio di Sistemi Operativi

## 23 giugno 2015

### Compito

```
adm      /var/adm
bin      /bin
daemon   /sbin
lp       /var/spool/lpd
root     /root
sync     /sbin
```

#### Soluzione:

Una possibile soluzione è la seguente:

```
sort -t: -k1,2 /etc/passwd | cut -d: -f1 > login.txt
sort -t: -k1,2 /etc/passwd | cut -d: -f6 > home.txt
paste login.txt home.txt
```

4. Si predisponga uno script della shell che prenda come argomento sulla linea di comando un intero positivo o nullo e ne calcoli il fattoriale, stampandolo su standard output. Si gestiscano gli eventuali errori relativamente a:

1. passaggio di un numero di argomenti errato (ovvero, diverso da uno);
2. passaggio di un intero negativo.

Esempio:

```
> ./fact.sh 4
24
```

#### Soluzione:

Esempio di soluzione:

```
if test $# -ne 1
then
    echo "utilizzo: $0 n"
    exit 1
fi

if test $1 -lt 0
then
    echo "l'argomento deve essere un intero positivo o nullo"
    exit 2
fi

fact=1
n=$1

while test $n -gt 1
do
    fact=$[fact * $n]
    n=$[n - 1]
done

echo $fact

exit 0
```

# Laboratorio di Sistemi Operativi

## 23 giugno 2015

### Compito

5. Sia data la seguente struttura ricorsiva in C:

```
struct elemento {
    int val;
    struct elemento *prossimo;
};
struct elemento *lista=NULL;
```

Si scriva il codice di una funzione avente il seguente prototipo:

```
void raddoppia(struct elemento *head);
```

che, scorrendo gli elementi della lista puntata da `head`, raddoppi il valore memorizzato nel membro `val` di ogni elemento.

#### Soluzione:

Esempio di soluzione:

```
void raddoppia(struct elemento *head) {
    while(head!=NULL) {
        head->val*=2;
        head=head->prossimo;
    }
}
```

6. Il programma seguente utilizza i thread ed i relativi meccanismi di accesso esclusivo (mutex) introdotti a lezione per creare `NUM_THR` thread figli e consentire ad ognuno di essi di scrivere il proprio THREAD ID nel file `registro.txt` nella linea corrispondente (il primo thread figlio nella prima linea, il secondo nella seconda ecc.). Ogni linea è lunga `LENGTH` caratteri: l'ultimo carattere è il newline.

Si completi il sorgente specificando i comandi mancanti da inserire al posto dei ... nei 5 punti indicati, affinché un solo thread per volta possa accedere in modo esclusivo al file `registro.txt`.

```
void *scrivi(void *n) { // il parametro n è l'indice corrispondente al thread
    // ovvero, la sua posizione (linea) in registro.txt
    int i;
    char buffer[LENGTH];

    for(i=0;i<LENGTH-1;i++) buffer[i]=' '; // pulisce il buffer
    buffer[LENGTH-1]='\n'; // imposta il newline
    sprintf(buffer,"%lu",thread[((int)n)-1]); // scrive il THREAD ID nel buffer
    ... // <-- inizio sezione critica: completare (1)
    ... // sposta il puntatore di lettura/scrittura sulla linea giusta: completare (2)
    ... // scrive nel file: completare (3)
    ... // <-- fine sezione critica: completare (4)
};

pthread_mutex_t file_mutex=PTHREAD_MUTEX_INITIALIZER;

int main() {
    int n;
    fd=open("registro.txt",O_WRONLY | O_CREAT,0644); // apre il file registro.txt in scrittura
    for(n=1;n<=NUM_THR;n++) {
        if(pthread_create(&thread[n-1],NULL,scrivi,(void *)n)!=0) {
            perror("Errore nella creazione del thread.\n");
        }
    }
}
```

**Laboratorio di Sistemi Operativi**  
**23 giugno 2015**  
**Compito**

```
    exit(1);
}
printf("Sono il padre %d: ho creato il thread %lu.\n",getpid(),thread[n-1]);
}
for(n=1;n<=NUM_THR;n++)
    ... // <-- il padre attende la terminazione del figlio n-esimo: completare (5)
close(fd);
return 0;
}
```

**Soluzione:**

1. pthread\_mutex\_lock(&file\_mutex);
2. lseek(fd,(((int)n)-1)\*LENGTH,SEEK\_SET);
3. write(fd,buffer,LENGTH);
4. pthread\_mutex\_unlock(&file\_mutex);
5. pthread\_join(thread[n-1],NULL);