



# Simulazione d'esame

---

*Roberto De Virgilio*

*Sistemi operativi - 10 Gennaio 2018*

# Simulazione: domanda di teoria

*Definire il significato di risorsa **prerilasciabile** ("**preemptable**") e di risorsa **non prerilasciabile** ("**not preemptable**") indicando anche le **condizioni** affinchè tali risorse si possano definire tali.*

# Simulazione: domanda di teoria

## RISPOSTA:

- una risorsa si dice **prerilasciabile** se la funzione di gestione può sottrarla ad un processo prima che questo l'abbia effettivamente rilasciata.
- una risorsa è **prerilasciabile** se il suo stato non si modifica durante l'utilizzo oppure il suo stato può essere facilmente salvato e ripristinato.

# Simulazione: domanda di teoria

## **RISPOSTA:**

- una risorsa si dice **non prerilasciabile** se la funzione di gestione non può sottrarla al processo al quale è stata assegnata.
- una risorsa è **non prerilasciabile** se il suo stato non può essere salvato e ripristinato.

# Simulazione: gestione espressioni regolari in Linux

Si consideri il file di testo **5Maggio.txt** contenente le strofe della famosa poesia del Manzoni, ad esempio:

Muta pensando all'ultima  
Ora dell'uom fatale;  
Nè sa quando una simile  
Orma di più mortale  
La sua cruenta polvere  
A calpestar, verrà.

Definire un comando linux per stampare solo le righe tale per cui:

- il primo carattere è una vocale maiuscola
- l'ultimo carattere è un punto ('.')
- contengono il carattere virgola (',')

Rispetto all'esempio precedente l'esecuzione del comando stamperà su output le seguenti righe:

A calpestar, verrà.

## Simulazione: gestione espressioni regolari in Linux

- ◆ `grep '^[AEIOU].*, v.*[. ]$' 5Maggio.txt`

A calpestar, verrà.

# Simulazione: manipolazione file tramite script AWK

Si consideri il file `lotto1.txt` in cui ogni riga rappresenta una estrazione del lotto che segue il seguente schema:

`CodiceEstrazione numero1 numero2 numero3 numero4 numero5 numero6`

si scriva uno script in linux tale per cui si producano in output righe con il seguente schema

`numero_estratto #estrazioni`

in cui si stampino per ogni numero quante volte questo è stato estratto (quindi in quante estrazioni è comparso). Bisogna però escludere le estrazioni in cui ricorrono “**numeri non validi**” (numeri non compresi nell’intervallo 1-90). Ordinare il risultato in ordine crescente rispetto al numero estratto.

Ad esempio, rispetto al file fornito, sarà stampato:

```
7 1  
11 1  
12 1  
16 1  
17 1  
18 4  
19 2  
...  
...
```

## Simulazione: manipolazione file tramite script AWK

```
#!/usr/bin/awk -f

BEGIN {
    printf("-----\n")
    printf("Numero #Estrazioni\n")
    printf("-----\n")
}

}
```

## Simulazione: manipolazione file tramite script AWK

```
{  
    test = 0  
    for (i=2; i<8; i++)  
        if (($i < 1) || ($i > 90)) {  
            test = 1  
            break  
        }  
  
    if (test == 0)  
        for (i=2; i<8; i++)  
            estrazioni[$i]++  
}
```

## Simulazione: manipolazione file tramite script AWK

```
END {  
    ordina = "sort -k 1n"  
    print ""  
    print "-----"  
    for (i in estrazioni)  
        printf("%d %d\n", i, estrazioni[i]) | ordina  
    print ""  
}
```

# Simulazione: gestione file in linguaggio C

Si consideri il file di testo `persone_frutta.txt` contenente le seguenti righe:

```
Paul mela pera pesca  
Mark pesca mela banana  
John mela pesca kiwi  
Bob pera pesca albicocca
```

Ogni riga contiene quattro stringhe separate da spazio con il seguente schema:

`NomePersona Frutto1Preferito Frutto2Preferito Frutto3Preferito`

Scrivere in linguaggio C la seguente funzione

```
int creaBinario (Stringa nome_file, Stringa frutto)
```

Tale funzione deve creare un file binario di nome "`frutto_preferito.dat`" contenente, per ogni riga del file di testo `nome_file`, un record a due campi di cui:

- il primo campo è una stringa (il nome della persona sulla riga corrente)
- il secondo campo è un numero intero/booleano (0 o 1), corrispondente all fatto che la persona ha il frutto indicato tra i tre preferiti

e alla fine tale funzione deve restituire quante persone hanno il frutto indicato tra i preferiti.

Si supponga la dichiarazione

```
typedef char Stringa[30];
```

per definire il tipo `Stringa` da utilizzare all'interno della funzione.

Ad esempio `creaBinario ("persone_frutta.txt", "pera")` ritornerà il valore `2` e creerà il file `frutto_preferito.dat` come segue:

```
[Paul] [1]  
[Mark] [0]  
[John] [0]  
[Bob] [1]
```

# Simulazione: gestione file in linguaggio C

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

typedef char Stringa[30];

typedef struct {
    Stringa nome;
    int flag;
} record_bin;
```

# Simulazione: gestione file in linguaggio C

```
int creaBinario (Stringa nome_file, Stringa frutto){
    int fg, count;
    Stringa nome, f1, f2, f3;
    FILE* fp = fopen(nome_file,"r");
    FILE* fp2 = fopen("frutto_preferito.dat","ab");

    while (!feof(fp)) {
        fscanf (fp, "%s %s %s %s\n", nome, f1, f2, f3);
        record_bin rb;
        fg = (strcmp(frutto,f1)==0) ||
             (strcmp(frutto,f2)==0) ||
             (strcmp(frutto,f3)==0);

        if (fg) count++;
        strcpy(rb.nome, nome);
        rb.flag = fg;
        fwrite(&rb, 1, sizeof(record_bin), fp2);

    }
    fclose(fp);
    fclose(fp2);
    return count;
}
```

# Simulazione: gestione file in linguaggio C

```
void leggibinario(){
    FILE* fp = fopen("frutto_preferito.dat","rb");

    while (!feof(fp)) {
        record_bin rb;
        fread(&rb, 1, sizeof(record_bin), fp);
        printf("%s %d\n", rb.nome, rb.flag);
    }

    fclose(fp);
}

int main(void) {
    creaBinario("persone_frutta.txt","pera");
    leggibinario();
    exit(0);
}
```

# Simulazione: gestione processi in linguaggio C

Si scriva in linguaggio C il programma `creazione.c` per la gestione dei processi che utilizzi la seguente sintassi:

`./creazione N`

dove `N` è un numero intero positivo (il suo valore deve essere compreso tra `1` e `10`).

Il processo iniziale (padre) `P0` deve creare `N` processi figli: per ogni processo creato il processo `P0` deve stampare su output il messaggio

`"Creato processo figlio: <pid del processo creato>"`

Il processo figlio `Pi`, una volta creato, deve porsi in attesa (mettersi in pausa) di un'eventuale attivazione da parte del padre. In caso di attivazione, `Pi` stamperà su output il messaggio

`"Processo attivato: <pid del processo attivato>"`

altrimenti stamperà su output il messaggio

`"Processo non attivato: <pid del processo non attivato>"`

in entrambi i casi, subito dopo la stampa `Pi` terminerà.

Il processo padre `P0`, una volta creato un processo figlio `Pi`, chiederà da input se attivare o meno il processo `Pi`: il processo `P0` stamperà il messaggio

`"Attivare il processo creato? [y/n]"`

e subito dopo chiederà da input un carattere ('`y`' o '`n`'); nel caso in cui il carattere sia '`y`' il processo `Pi` sarà attivato, nel caso del carattere '`n`', `Pi` non sarà attivato.

Ad esempio si consideri la seguente esecuzione:

`./creazione 3`

`Creato processo figlio: 2816`

`Attivare il processo 2816? [y/n]: n`

`Processo non attivato: 2816`

`Creato processo figlio: 2817`

`Attivare il processo 2817? [y/n]: y`

`Processo attivato: 2817`

`Creato processo figlio: 2818`

`Attivare il processo 2818? [y/n]: n`

`Processo non attivato: 2818`

# Simulazione: gestione processi in linguaggio C

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

#define max 10

void gestore_att(int sig){
    if (sig == SIGUSR1)
        printf("Processo attivato: %d\n", getpid());
    else
        printf("Processo non attivato: %d\n", getpid());
exit(0);
}
```

# Simulazione: gestione processi in linguaggio C

```
int main(int argc , char *argv[]){  
    int pid[max], pf;  
    int i, status;  
    char request;  
  
    int N = atoi(argv[1]);  
    if (argc==1){  
        printf("sintassi sbagliata!\n");  
        exit(1);  
    }
```

# Simulazione: gestione processi in linguaggio C

```
for(i=0; i< N; i++){
    pid[i]=fork();
    if (pid[i]==0){
        signal(SIGUSR1, gestore_att);
        signal(SIGUSR2, gestore_att);
        pause();
    }
    else {
```

# Simulazione: gestione processi in linguaggio C

```
printf("Creato processo figlio: %d\n", pid[i]);

printf("Attivare il processo %d? [y/n]: ", pid[i]);

request=getchar();

if ((request=='Y') || (request=='y'))
    kill(pid[i], SIGUSR1);
else
    kill(pid[i], SIGUSR2);

request=getchar(); //serve a catturare l'andare a capo
                  //nel momento in cui si preme INVIO
                  //subito dopo aver digitato il carattere 'y' o 'n'

pf= wait(&status);

}

exit(0);
} /* fine padre */
```

