Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri **prof.ssa Donatella Sciuto**
prof. Gerardo Pelosi **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – lunedì 14 febbraio 2022

Cognome_____ **Nome** _____

Matricola _____**Firma** _____

Istruzioni

- Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.
- Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta se staccati vanno consegnati intestandoli con nome e cognome.
- È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- Tempo a disposizione 1 h : 30 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (4 punti) _____

esercizio 2 (5 punti) _____

esercizio 3 (5 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "`#include`" e le inizializzazioni dei *mutex* sono omessi, come anche il prefisso `pthread` delle funzioni di libreria NPTL):

```
pthread_mutex_t sun, moon
sem_t star
int global = 0
```

```
void * day (void * arg) {
   mutex_lock (&sun)
   sem_post (&star)
   global = 1                                          /* statement A */
   mutex_unlock (&sun)
   global = 2
   mutex_lock (&moon)
   sem_wait (&star)
   mutex_unlock (&moon)
   return (void *) 3
} /* end day */
```

```
void * night (void * arg) {
   mutex_lock (&sun)
   sem_wait (&star)
   mutex_lock (&moon)
   global = 4                                          /* statement B */
   mutex_unlock (&moon)
   sem_post (&star)
   global = 5                                          /* statement C */
   mutex_unlock (&sun)
   return NULL
} /* end night */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&star, 0, 0)
   create (&th_2, NULL, night, NULL)
   create (&th_1, NULL, day, NULL)
   join (th_1, &global)                                /* statement D */
   join (th_2, NULL)
   return
} /* end main */
```

Si completi la tabella qui sotto **indicando lo stato di esistenza del *thread*** nell'istante di tempo specificato da ciascuna condizione, così: se il *thread* **esiste**, si scriva ESISTE; se **non esiste**, si scriva NON ESISTE; e se può essere **esistente** o **inesistente**, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede lo stato che il *thread* assume tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	*thread*	
	th_1 – *day*	**th_2 – *night***
subito dopo stat. **A**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D**		

Si completi la tabella qui sotto, **indicando i valori delle variabili globali** (sempre esistenti) nell'istante di tempo specificato da ciascuna condizione. Il **valore** della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)
- si supponga che il mutex valga 1 se occupato, e valga 0 se libero

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	**variabili globali**			
	sun	*moon*	*star*	*global*
subito dopo stat. **A**				
subito dopo stat. **B**				
subito dopo stat. **C**				
subito dopo stat. **D**				

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **tre casi diversi**. Si chiede di precisare il comportamento dei thread in **due casi**, indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	**th_1 – *day***	**th_2 – *night***	*global*
1			
2			
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma foo.c
int main ( ) {
      pid1 = fork ( )                                  // creazione del processo Q
      if (pid1 == 0) {                                 // codice eseguito da Q
         write (stdout, "proc Q", 6)
         execl ("/acso/progXY", "progXY", NULL)
         exit (-1)
      } else {                                         // codice eseguito da P
         write (stdout, "Avvio Prog", 10)
         pid2 = fork ( )                               // creazione del processo R
      } /* if */
      if (pid2 == 0) {                                 // codice eseguito da R
         write (stdout, "proc R", 6)
         execl ("/acso/progXY", "progXY", NULL)
         exit (-2)
      } /* if */
      pid2 = waitpid (pid2, NULL, 0)                   // codice eseguito da P
      pid1 = waitpid (pid2, NULL, 0)
} /* main */
```

```
// programma progXY.c
pthread_mutex_t fence = PTHREAD_MUTEX_INITIALIZER
sem_t flag
```

`void * routine1 (void * arg) {`	`void * routine2 (void * arg) {`
`  sem_wait(&flag)`	`  sem_post (&flag)`
`  sem_post (&flag)`	`  pthread_mutex_lock (&fence)`
`  return NULL`	`  sem_wait (&flag)`
`} /* routine1 */`	`  pthread_mutex_unlock (&fence)`
	`  sem_post (&flag)`
	`  return NULL`
	`} /* routine2 */`

```
// codice eseguito da R
int main ( ) {
   pthread_t TH_1, TH_2
   sem_init (&flag, 0, 0)
   pthread_create (&TH_1, NULL, routine1, NULL)
   pthread_create (&TH_2, NULL, routine2, NULL)
   write (stdout, "Fine!", 5)
   pthread_join (TH_2, NULL)
   pthread_join (TH_1, NULL)
   exit (1)
} /* main */
```

Un processo **P** esegue il programma **foo.c** e crea i processi **Q** e **R.** Il processo **Q** esegue una mutazione di codice che **non** va a buon fine. Il processo **R** effettua con successo una mutazione di codice ed esegue il programma **progXY.c**, creando i thread **TH_1** e **TH_2**.

Si simuli l'esecuzione dei processi così come risulta dal codice dato, dagli eventi indicati.

Si completi la tabella riportando quanto segue:

- PID e TGID di ogni processo che viene creato
- identificativo del processo-chiamata di sistema / libreria nella prima colonna, dove necessario
- in funzione del codice proposto in ciascuna riga, lo stato dei processi **al termine del tempo indicato**

TABELLA DA COMPILARE (numero di colonne non significativo)

identificativo simbolico del processo		**IDLE**	**P**	**Q**	**R**	**TH_1**	**TH_2**
evento oppure processo-chiamata	PID	**1**					
	TGID	**1**					
P – write	**1**	**esec**	**attesa (write)**	**attesa (write)**	**NE**	**NE**	**NE**
10 interrupt da stdout	**2**			**esec**			
	3						
	4						
6 interrupt da stdout (tutti i caratteri trasferiti)	**5**						
	6						
P – waitpid pid2	**7**			**esec**			
	8						
	9						
6 interrupt da stdout	**10**						
	11						
	12						
	13					**pronto**	
R – write	**14**						
	15						
	16						
	17						
	18						
	19						

seconda parte – moduli, pila e strutture dati HW

Si consideri il seguente stato di esecuzione:

CURR = P Q = WAIT (write)

La runqueue contiene un solo task dato dal processo **P,** mentre il processo **Q** è in stato di **attesa da stdout** per la scrittura di 10 caratteri a seguito dell'esecuzione di una **write (…)** della libreria *glibc*. Il sistema non contiene altri task.

Si consideri questo evento: il **processo P** è in esecuzione in **modo U**, e si attivano 10 interruzioni da stdout risvegliando il processo **Q**; di esse si considera solo l'ultima. **La condizione di preemption è verificata.**

Domanda:

- Mostrare le **invocazioni di tutti i moduli** (ed eventuali relativi **ritorni**) eseguiti nel contesto del processo **P** e del processo **Q**, per gestire l'evento indicato fino al ritorno di **Q** nella **write (…)** della *glibc*.
- Mostrare (in modo simbolico) l'evoluzione dello **stack di sistema** del processo **P** al termine della gestione dell'evento considerato (parte dello stack finale di **P** è già mostrata). È mostrato per intero lo **stack di sistema iniziale** del processo **Q**.

invocazione moduli – numero di righe non significativo

processo	modo	modulo
P	U – S	>R_Int (stdout)

sStack_P finale	sStack_Q iniziale
	USP
	~~a schedule da pick_next_task~~
	a wait_event da schedule
	a sys_write da wait_event
	a System_Call da sys_write
PSR (u)	PSR (u)
a codice utente da R_Int	a syscall da System_Call

PAGINA BIANCA DI ALLINEAMENTO – usabile per continuazione o brutta copia

esercizio n. 3 – memoria e file system

prima parte – gestione dello spazio di memoria

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = **3** **MINFREE** = **2**

situazione iniziale (esistono un processo P, un processo Q e un processo R)

```
PROCESSO: P  ********************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X,0>
          D  000000600,  4 , W  ,  P  ,  A  , <-1,0>
          P  7FFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>

    PT: <c0 :1  R>  <c1 :- ->   <d0 :s1 R>  <d1 :7  W>  <d2 :2  W>  <d3 :- ->
        <p0 :6  W>  <p1 :5  R>  <p2 :- ->

    process P - NPV of PC and SP:  c0, p1

PROCESSO: Q  *****SOLO LE INFORMAZIONI RILEVANTI **********************
    process Q - NPV of PC and SP:  c0, p0

PROCESSO: R  *****SOLO LE INFORMAZIONI RILEVANTI **********************
    process R - NPV of PC and SP:  c0, p0

____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>               ||  01 : Pc0 / Qc0 / Rc0 / <X,0> ||
      02 : Pd2                ||  03 : Rp0   D                 ||
      04 : ----               ||  05 : Pp1 / Rp1               ||
      06 : Pp0                ||  07 : Pd1                     ||
      08 : ----               ||  09 : ----                    ||

____STATO del TLB_____________________________________________________
      Pc0 : 01 -  0: 1:       ||   Pp0 : 06 -  1: 1:           ||
      Pd2 : 02 -  1: 0:       ||   Pp1 : 05 -  1: 1:           ||
      Pd1 : 07 -  1: 0:       ||         -----                 ||
            -----             ||         -----                 ||

SWAP FILE:     Qp0  ,Pd0/Rd0, ----, ----, ----, ----,
LRU ACTIVE:    PP0,  PC0,  PP1,
LRU INACTIVE: pd2,  pd1,  rp0,  rc0,  rp1,  qc0,
```

evento 1: *read* (Pp2) *write* (Pp3)

VMA del processo **P** (è da compilare solo la riga relativa alla VMA **P**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
P							

PT del processo: **P**				
p0:	p1:	p2:	p3:	p4:

process P	NPV of **PC**:	NPV of **SP**:

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0:	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: _____

LRU INACTIVE: _____

evento 2: *read* (Pd0) *write* (Pp1)

PT del processo: **P**				
d0:	d1:			

process P	NPV of **PC**:	NPV of **SP**:

MEMORIA FISICA	
00: <ZP>	01:
02:	03:
04:	05:
06:	07:
08:	09:

SWAP FILE	
s0:	s1:
s2:	s3:
s4:	s5:

LRU ACTIVE: _____

LRU INACTIVE: _____

seconda parte – file system

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = **2** **MINFREE** = **1**

Si consideri la seguente **situazione iniziale.**

process P – NPV of PC and SP: c2, p0

```
____MEMORIA FISICA____(pagine libere: 1)_____________________________
      00 : <ZP>                ||  01 : Pc2 / Qc2 / <X, 2>  ||
      02 : Qp0  D              ||  03 : <F, 0>  D           ||
      04 : <F, 1>  D           ||  05 : <F, 2>  D           ||
      06 : Pp0                 ||  07 : ----                ||

____STATO del TLB_____________________________________________________
      Pc2 : 01 -  0: 1:        ||   Pp0 : 06 -  1: 1:       ||
            -----              ||         -----             ||
            -----              ||         -----             ||
```

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P, Q	**F**	**9000**	**2**	**3**	**0**

ATTENZIONE: **è presente la colonna "processo" in cui va specificato il nome/i del/i processo/i a cui si riferiscono le informazioni "f_pos" e "f_count" (campi di struct file) relative al file indicato**

Il processo **P** è in esecuzione. Il file **F** è stato aperto da **P** tramite chiamata **fd =** ***open*** **(F).**

ATTENZIONE: il numero di pagine lette o scritte di un file è cumulativo, ossia è la somma delle pagine lette o scritte su quel file da tutti gli eventi precedenti oltre a quello considerato. Si ricorda inoltre che la primitiva *close* scrive le pagine dirty di un file solo se *f_count* diventa = 0.

Per ciascuno degli eventi seguenti, **compilare** le tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative ai file aperti e al numero di accessi a disco effettuati in lettura e in scrittura.

eventi 1 e 2: *context switch* (Q) *close* (fd)

MEMORIA FISICA	
00: <ZP>	01: Pc2 / Qc2 / <X, 2>
02:	03:
04:	05:
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte

eventi 3 e 4: *context switch* **(P)** *write* **(fd, 5000)**

MEMORIA FISICA	
00: <ZP>	01: `Pc2 / Qc2 / <X, 2>`
02:	03:
04:	05:
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte

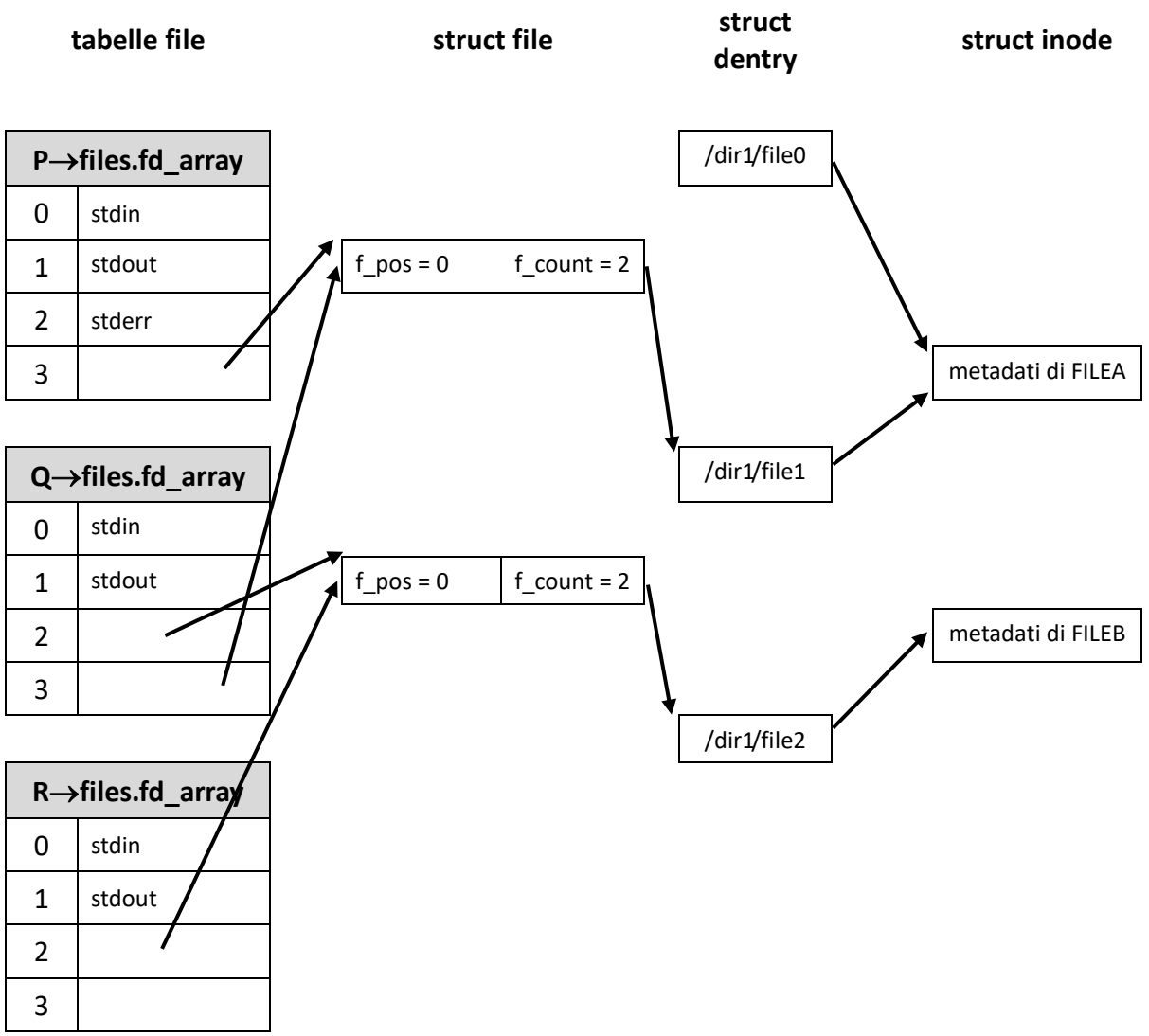
evento 5: *close* **(fd)**

MEMORIA FISICA	
00: <ZP>	01: `Pc2 / Qc2 / <X, 2>`
02:	03:
04:	05:
06:	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte

esercizio n. 4 – strutture dati del file system

La figura sottostante è una rappresentazione dello stato del VFS raggiunto dopo l'esecuzione in sequenza di un certo numero di chiamate di sistema (non riportate):

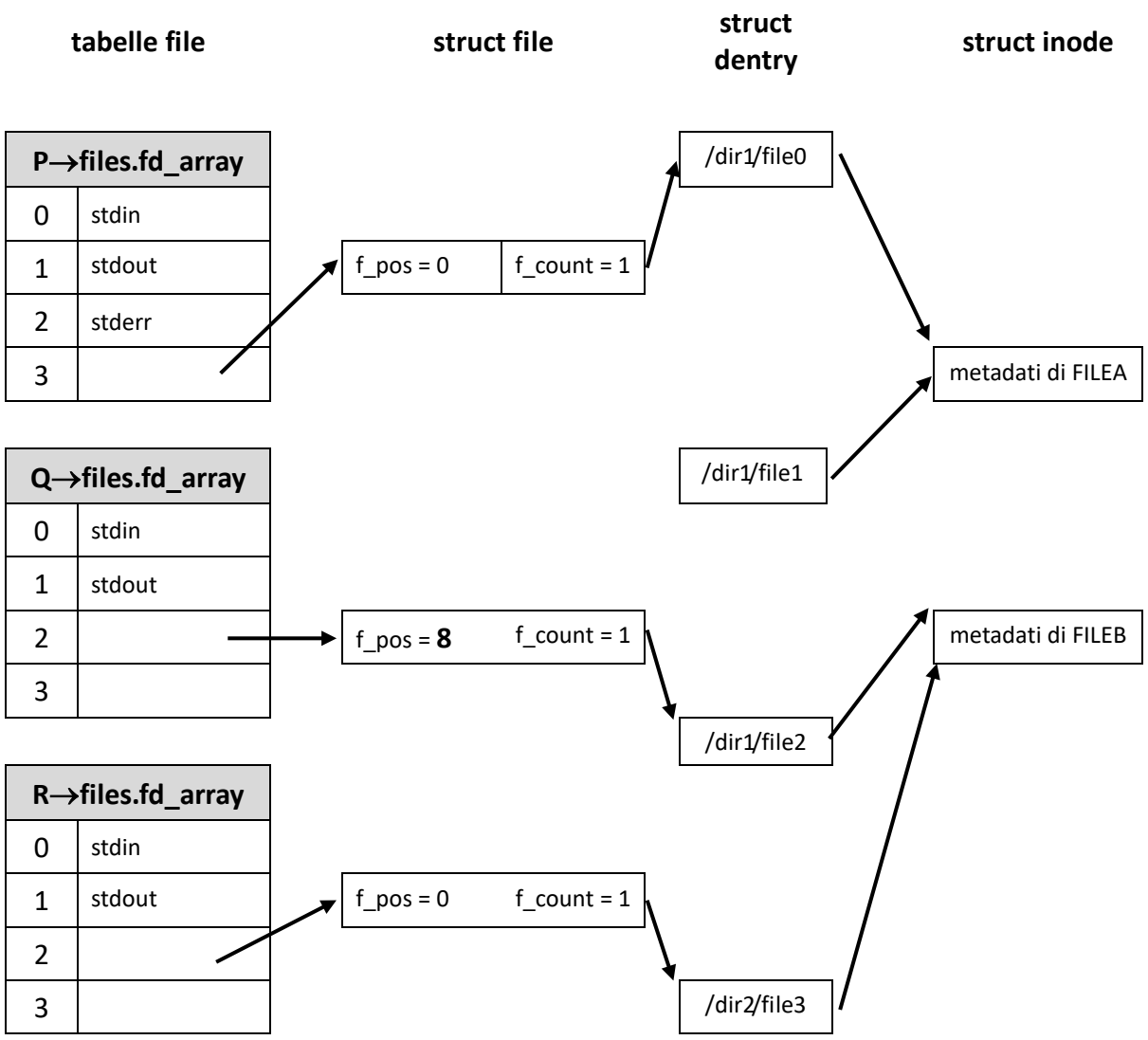


Il processo **P** ha creato il processo figlio **Q**, che a sua volta ha creato il processo figlio **R**.

Ora si supponga di partire dallo stato del VFS mostrato nella figura iniziale e si risponda alla **domanda** alla pagina seguente, riportando **una possibile sequenza di chiamate di sistema** che può avere generato la nuova situazione di VFS mostrata nella figura successiva. Il numero di eventi è esattamente 7 e questi sono eseguiti, nell'ordine, dai processi indicati.

Le sole chiamate di sistema usabili sono: **open** (nomefile, …), **close** (numfd), **link** (oldfilename, newfilename), **read** (numfd, numchar).

domanda



sequenza di chiamate di sistema

#	processo	chiamata di sistema
1	P	
2	P	
3	Q	
4	Q	
5	R	
6	R	
7	R	

spazio libero per brutta copia o continuazione