Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

22 Febbraio 2018

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Quer ◯ Sterpone

Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Risolvere gli esercizi negli spazi riservati. Fogli aggiuntivi sono permessi solo quando strettamente necessari. Riportare i passaggi principali.
Durata della prova: 100 **minuti.**

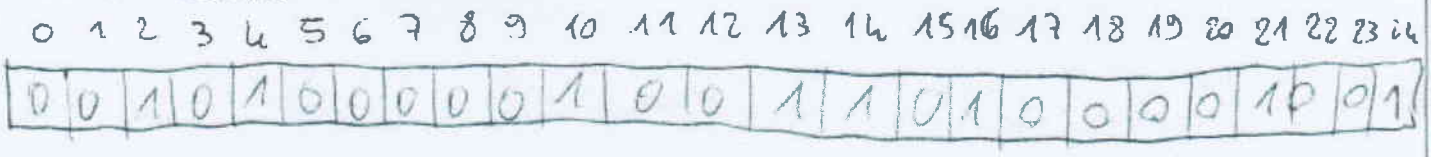
1. Si supponga che il disco rigido di un piccolo sistema embedded sia costituito da 25 blocchi di 1 MByte, che tali blocchi siano numerati da 0 a 24, che il sistema operativo mantenga traccia dei blocchi liberi (occupati) indicandoli in un vettore con il valore 0 (1), e che la situazione attuale del disco sia rappresentata dal seguente vettore:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	0	1	0	1	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	1

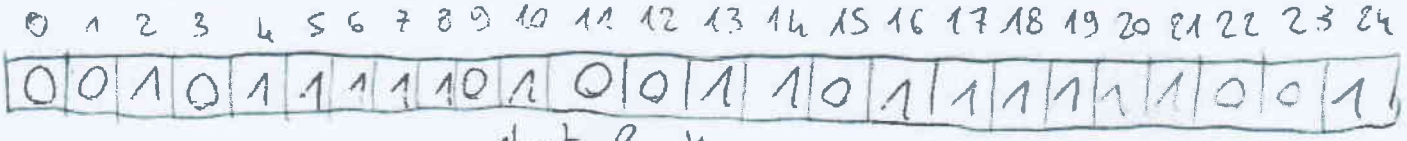
   Con riferimento alle metodologie di allocazione di file contigua, concatenata, FAT e indicizzata, indicare come possono essere allocati in sequenza i file `File1`, `File2` e `File3` ciascuno di dimensione uguale a 3.4 Mbyte.

   Riportare schematicamente per ogni strategia le informazioni memorizzate nella directory entry ed i relativi puntatori.

Stato disco iniziale



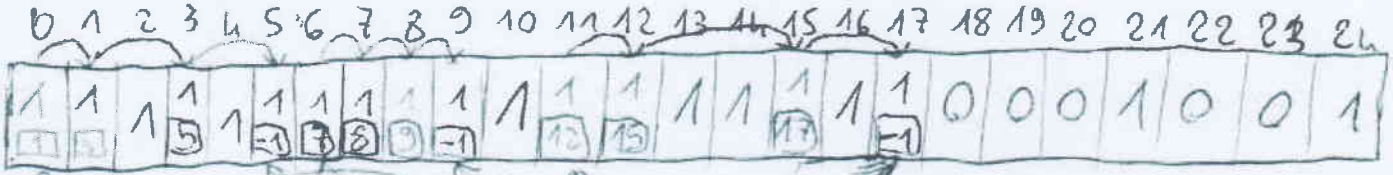
Contigua (First fit)



Directory entry:	start	length
File 1	5	4
File 2	17	4

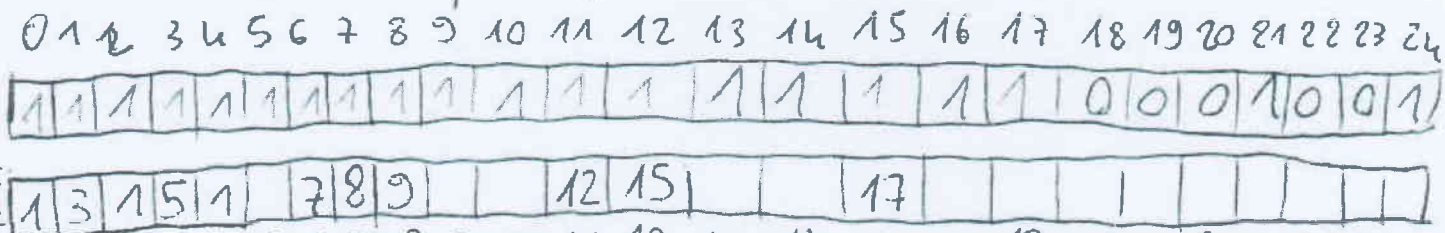
File 3 non può essere allocato

Allocazione concatenata



Directory entry
File 1 0 5
File 2 6 9
File 3 11 17

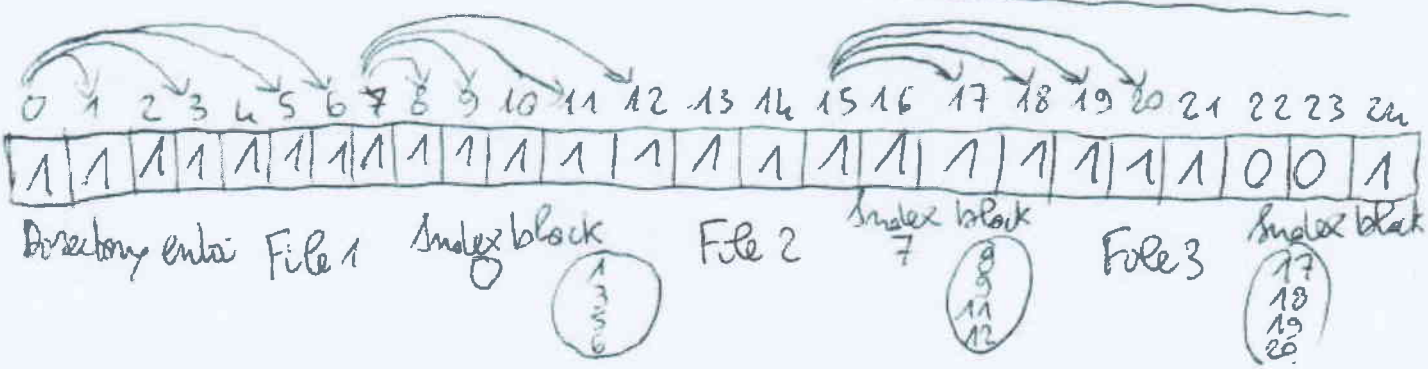
FAT (Uguale a concatenata, ma con sequenza blocchi memorizzata in struttura separata)



Directory entry

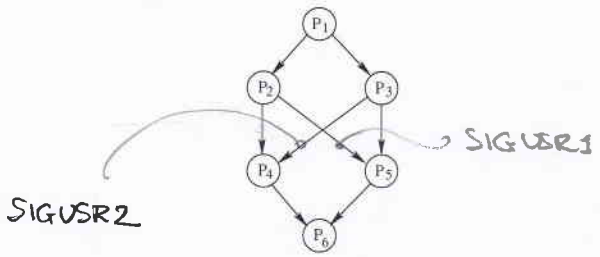
	start block
File 1	0
File 2	6
File 3	11

Allocazione indicizzata



Directory entry
File 1 — Index block: 1, 3, 5, 6
File 2 — Index block: 7 → 8, 9, 11, 12
File 3 — Index block: 17 → 18, 19, 20

2. Ignorando la possibilità di corse critiche si realizzi il seguente grafo di precedenza in cui i vari vertici rappresentano **blocchi di istruzioni** all'interno dello stesso programma. Si utilizzino **esclusivamente** system call `fork`, `wait` e `waitpid` e la gestione dei **segnali** (in altre parole **non** è permesso l'utilizzo di semafori). Si noti che uno stesso processo può anche eseguire più blocchi di istruzioni. Si scriva il **codice C** corrispondente.



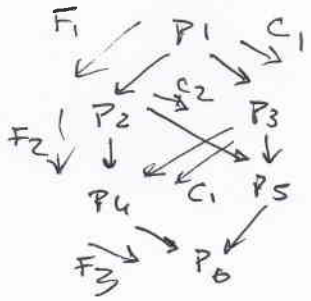
Si chiarisca inoltre quali tecniche può utilizzare un processo per attendere la terminazione di un figlio, indicando il significato dell'istruzione `signal (SIGCHLD, SIG_IGN)` e il suo effetto su eventuali istruzioni `wait` successive.

SOLUZIONE 1

```
static void sigUsr(int signo) {
    if (signo == SIGUSR1 || signo == SIGUSR2) {
        return;
    }else {
        fprintf(stderr, "Error\n");
        exit(1);
    }
}
int main(void) {
    pid_t pid;
    if (signal(SIGUSR1, sigUsr) == SIG_ERROR || signal(SIGUSR2, sigUsr) == SIG_ERROR)
        return(1);
    printf("P1\n");
    pid = fork();
    if(pid > 0) { // Father
        printf("P2\n");
        kill (pid, SIGUSR2);
        pause();
        printf("P4");
        wait ((int *) 0);
    } else { // Child
        printf("P3\n");
        kill (getppid(), SIGUSR2);
        pause();
        printf("P5");
        exit(0);
    }
}
```

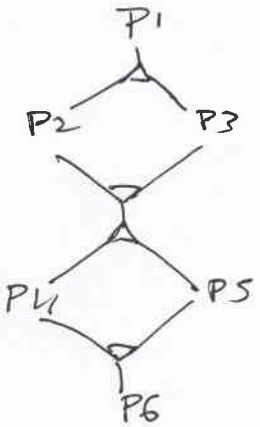
```
printf("P6\n");
return(0);
}
```

SOLUZIONE 2:




```
...
printf("P1\n");
pid1 = fork();
if (pid1 > 0) {
    // F1
    printf("P2\n");
    waitpid(pid1, &stat, 0);
    pid2 = fork();
    if (pid2 > 0) {
        // F2
        printf("P4\n");
        waitpid(pid2, &stat, 0);    // exit da F2
    } else {
        // C2
        printf("P5\n");
        exit(0);
    }
} else {
    // C1
    printf("P3\n");
    exit();
}
printf("P6\n");
return(1);
}
```

SOLUZIONE 3:




```
...
if (fork()) {
    printf("P2\n");
    wait((int *)0);
} else {
    printf("P3\n");
    exit(0);
}
if (fork()) {
    printf("P4\n");
    wait((int *)0);
} else {
    printf("P5\n");
    exit(0);
}
printf("P6\n");
```

Gestione sincrona: `pid = wait(&code);`

OPPURE

`pid = waitpid(?, &stat, options);`

↑ 0, WNOHANG, ...

Gestione asincrona: definizione gestore di segnali

...

`signal(SIGCHLD, gestore);`

Gestione mascherata: `signal(SIGCHLD, SIG_IGN);`

la wait/waitpid ritorna -1 senza attendere.

Il segnale SIGCHLD viene trascurato

3. Con riferimento alle soluzioni hardware al problema della sincronizzazione, si riportino le soluzioni mediante le procedure di `testAndSet` e di `swap`. Se ne illustrino le principali differenze (vantaggi e svantaggi) rispetto alle tecniche software, nonchè la loro relazione con il problema della "starvation".

   Si indichi inoltre che cosa si intende per "spin-lock" e per "priority inversion".

```
char TestAndSet (char * lock) {
    char val;
    val = *lock;
    *lock = TRUE;
    return val;
}

char lock = FALSE;

...
while (TRUE) {
  while (TestAndSet(&lock));
   SC
   lock = FALSE;
   S non C
}
...
```

```
void swap(char *v1, char *v2) {
    char tmp;
    tmp = *v1;
    *v1 = *v2;
    *v2 = tmp;
    return;
}

char lock = FALSE;

...
while (TRUE) {
    key = TRUE;
    while (key == TRUE)
        swap(&lock, &key);
    SC
    lock = FALSE;
    S non C
}
...
```

- Assicurano ME
- Assicurano progresso (no deadlock)
- NON assicurano attesa definita ⇝ ∃ possibilità di starvation
- Sono simmetriche — Possibilità di estensione (Burns, 1978)
- Utilizzabili in ambienti multi-proc
- Facilmente estendibili a N processi
- Complicazioni h/w
- Busy waiting su spin-lock (ciclo while con spreco risorse)
- Priority inversion (P con alta priorità in spin-lock non fa procedere P con bassa priorità in SC)

4. Un file contiene un elenco di stringhe, in ragione di una stringa su ciascuna riga, che individuano path completi a file ASCII.

   Si scriva uno script BASH che riceva quali parametri il nome di un tale file (`nome`) e tre valori interi (`n1`, `n2` e `n3`) e sia in grado di effettuare le seguenti operazioni:

   - Verificare che i 4 parametri siano specificati correttamente sulla riga di comando, che i valori interi siano positivi e che il primo valore numerico `n1` sia minore del secondo `n2`. In caso contrario lo script visualizzi un messaggio di errore e termini.
   - Leggere il file `nome` e per ciascuna stringa in esso contenuto:
     - Verificare tale stringa individui un file regolare. In caso contrario visualizzare un messaggio di errore.
     - Se la dimensione del file è inferiore a `n1` byte, cancellarlo.
     - Se la dimensione del file è compresa tra `n1` e `n2` byte, ignorarlo.
     - Se la dimensione del file è maggiore di `n2` byte, comprimerlo. Comprimere un file significa cancellarlo dopo averne effettuato una copia in un file con stesso lo path ma con aggiunta una ulteriore estensione `.compresso`. Inoltre il contenuto va modificato in modo da copiare solo una stringa ogni `n3` stringhe (ovvero occorre copiare solo le stringhe di posizione 1, 1+`n3`, 1 + 2·`n3`, etc.). Si considerino le stringhe separate da spazi o da caratteri di "a capo".

```
#!/bin/bash

# Command to test the script with the provided input files:
#      ./ex_bash_22022018.sh ex_bash_22022018_list.txt 1000 2000 3

# Check number of parameters
if [ $# -ne 4 ]; then
  echo "Usage $0 <list> <n1> <n2> <n3>"
  exit 1
fi

# Check validity of input parameters
if [ ! -f $1 ]; then
  echo "List is not a valid file."
  exit 1
fi
if [ $2 -lt 0 ] || [ $3 -lt 0 ] || [ $4 -lt 0 ]; then
  echo "Values n1, n2 and n3 should be non-negative integers."
  exit 1
fi
if [ $2 -gt $3 ]; then
  echo "Values n1 should be non-greater than n2."
  exit 1
fi

# Read paths from list file
while read file; do

  # Skip invalid paths to regular files
  if [ ! -f "$file" ]; then
    echo "Invalid file: $file"
    continue
  fi

  # Compute file size
  size=$(cat $file | wc -c)

  # Remove files with size lower than the first threshold
  if [ $size -lt $2 ]; then
    rm -f $file

  # Compress files with size greater than the second threshold
  elif [ $size -gt $3 ]; then
    i=1
    for word in $(cat $file); do
      let "i--"
      if [ $i -eq 0 ]; then
        echo $word >> $file".compresso"
        i=$4
      fi
    done
  fi
done < $1
```

5. Un primo file contiene un elenco di azioni con il loro valore borsistico. Ogni riga è composta dal nome dell'azione seguito dal suo valore.

   Un secondo file ha righe composte dal nome di un utente (che inizia con una lettera), seguito dal nome di un'azione, dalla parola "ACQUISTO" o "VENDITA" e dal numero di azioni acquistate o vendute. Si noti che le righe file che iniziano con la sequenza di caratteri ``//'' o con un carattere numerico (da 0 a 9) sono da considerarsi alla stregua di commenti e vanno ignorate.

   Si analizzi l'esempio successivo per maggiori dettagli.

Input file #1	**Input file #2**	**Stampa finale**
eni 15.00	12345 file di esempio	giulia 25.00
snam 5.00	stefano eni ACQUISTO 10	stefano 135.00
ilsole24ore 0.50	// Capitale stefano 150.00 euro	TOTALE: 160.00
	giulia snam ACQUISTO 5	
	// capitale giulia 25.00 euro	
	stefano eni VENDITA 1	
	// capitale stefano 135.00 euro	

   Si scriva uno script AWK in grado di valutare il capitale azionario di ogni individuo specificato nel secondo file:

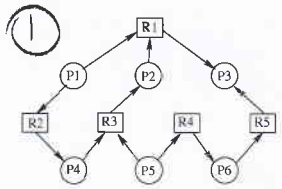
   - Il capitale azionario iniziale è nullo per tutti gli utenti.
   - Ogni acquisto (vendita) di un'azione aumenta (diminuisce) il capitale di chi lo ha effettuato di un valore pari al numero di azioni moltiplicato per il valore dell'azione.

   Lo script deve visualizzare il capitale di ogni individuo nonché il capitale totale di tutti gli individui al termine delle operazioni effettuate. L'ordine di stampa dei nomi utenti e il numero di cifre decimali è a scelta del candidato. Si supponga che i file siano corretti.

```
#!/usr/bin/awk

BEGIN {

  while(getline < ARGV[1]) {
    valori[$1]=$2
  }

  ARGV[1]=ARGV[2]
  ARGC--

}

$1 !~ /^[0-9]/ && $1 !~ /^\/\// {

  if($3 == "ACQUISTO") {
    capitale[$1] += valori[$2]*$4
  } else if ($3 == "VENDITA") {
    capitale[$1] -= valori[$2]*$4
  }

}

END {

  for(azionista in capitale) {
    printf("%s %.2f\n", azionista, capitale[azionista])
    tot+=capitale[azionista]
  }
  printf("TOTALE: %.2f\n", tot)

}
```

6. Si illustri che cosa si intende per "stato sicuro", "sequenza sicura" e (con un esempio) per "grafico delle traiettorie delle risorse".

Per gestire le condizioni di stallo, due sistemi operativi utilizzano rispettivamente il grafo riportato a sinistra e la tabella riportata a destra della seguente figura.



Processo	Fine	Assegnate			Massimo			Necessità			Disponibilità		
		R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	F	1	0	1	4	3	4	3	3	3	3	2	2
P_2	F	0	0	1	4	4	2	4	4	1	3	3	3
P_3	F	0	1	1	2	3	3	2	2	2	4	3	4
P_4	F	0	0	0	3	4	5	3	4	5	4	4	4
P_5	F	0	1	0	1	4	4	1	3	4	4	4	5
											4	4	5

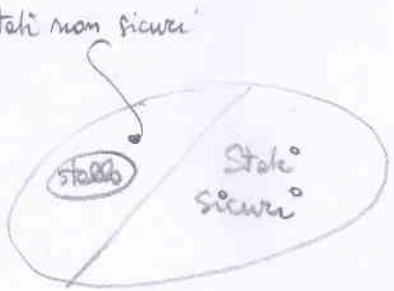
(1) (2) P3 P1 P5 P2 P4

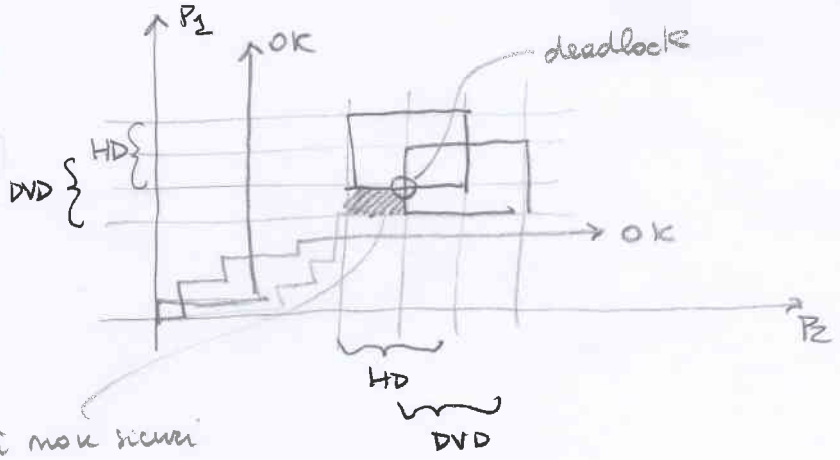
Si indichi se gli stati rappresentati sono sicuri e si indichi una eventuale sequenza sicura nei due casi.

Stato sicuro : il sistema si trova in uno stato dal quale può determinare una sequenza sicura

Sequenza sicura : sequenza schedulazione processi in modo da tutti possano essere terminati senza entrare in stallo

Progresso congiunto di 2 processi :





(1) Esiste una Sequenza sicura :

P3 P2 P4 P6 P5 P1

P_3 termina
R_5 è data a P6 ; P6 termina
R_1 è data a P2 ; P2 termina
R_3 e R_4 sono date a P5 ; P5 termina
R_3 è data a P4 ; P4 termina
R_2 è data a P1 ; P1 termina

P3 P6 OR P2 P2 OR P6 P5 OR P4 P4 OR P3 P1 P1 P1 P5

(2) idem ; 1 sola