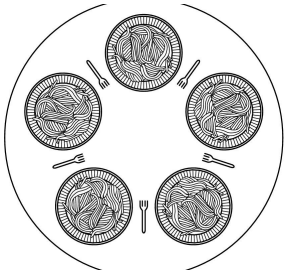
Quesito 1:

DOMANDA	Vero/Falso
In un sistema di memoria a paginazione, il *Translation Lookaside Buffer* (TLB) velocizza la traduzione di indirizzi virtuali in indirizzi fisici	
La segmentazione consente a due processi di condividere un segmento	
Se non vi sono percorsi chiusi in un grafo di allocazione allora non vi è situazione di stallo	
FAT è un file system ad allocazione concatenata	
La politica di scheduling round robin minimizza il tempo medio di attesa dei processi	
Se un processo è in blocco da 10 ms significa che 10 ms fa ha eseguito una *system call*	
Ogni *interrupt* può essere associato ad un processo che ha richiesto una operazione di I/O	
Con NTFS è possibile che il file system scriva il contenuto di file di piccola dimensione (es. <1KB) direttamente nell'inode	
pwd è un comando GNU/Linux per modificare la password	

Quesito 2:



I "filosofi a cena" è un classico problema di sincronizzazione tra più processi (i filosofi) che accedono concorrentemente a risorse condivise (le forchette).

Come visto in aula, lo studente utilizzi i ***semafori*** per scrivere una procedura `Filosofo` che cerchi a fasi alterne di pensare e mangiare. Tali procedure dovranno poter essere eseguite concorrentemente (come fossero un gruppo di filosofi a tavola) evitando *deadlock* del sistema o *starvation* di filosofi.

Si consideri un tavolo con `N` filosofi ed `N` forchette.

Nota: lo studente si ricordi di inizializzare i valori delle variabili semaforo usate nella sua soluzione.

Quesito 3:

[3.A] La dimensione massima di un file ottenibile con file system ext2fs dipende dalla contiguità con cui sono scritti i blocchi del file su disco? *Sì / No* ? Perché? (Spiegato in due righe)

[3.B]La dimensione massima di un file ottenibile con file system FAT dipende dalla contiguità con cui sono scritti i blocchi del file su disco? *Sì / No* ? Perché? (Spiegato in due righe)

[3.C] La dimensione massima di un file ottenibile con file system NTFS dipende dalla contiguità con cui sono scritti i blocchi del file su disco? *Sì / No* ? Perché? (Spiegato in due righe)

[3.D] Sia data una partizione di disco ampia 64 GB organizzata in blocchi dati di ampiezza 1 kB. In caso serva, si consideri l'ipotesi di contiguità nulla di un file (ciascun blocco si trova su disco in posizione non adiacente al blocco precedente e a quello successivo nella composizione del file).
Si determini l'ampiezza massima di file ottenibile per l'architettura di file system ext2fs assumendo i-node ampi 128 B, i-node principale contenente 12 indici di blocco e 1 indice di I, II e III indirezione ciascuno. Si determini poi il rapporto inflattivo che ne risulta, ossia l'onere proporzionale dovuto alla memorizzazione della struttura di rappresentazione rispetto a quella dei dati veri e propri.

Quesito 4: Cinque processi *batch*, identificati dalle lettere A, B, C, D, E, arrivano all'elaboratore agli istanti 0, 1, 2, 6, 7 rispettivamente. Essi hanno un tempo di esecuzione stimato di 3, 7, 2, 3, 1 unità di tempo rispettivamente e priorità 3, 5, 2, 4, 1 rispettivamente (dove 5 è la massima priorità e 0 è la minima). Per ognuna delle seguenti politiche di ordinamento:

A) *Round Robin* (divisione di tempo, con priorità, senza prerilascio per priorità, e con quanto di tempo di ampiezza 2)
B) *Fair Priority Scheduling*
   Per evitare attesa infinita la politica di *Fair Priority Scheduling* prevede che, a seguito di due unità di tempo **consecutive** di esecuzione, la priorità del processo in esecuzione scenda di un punto. (*Esempio 1.* Se il processo è in esecuzione per 4 unità di tempo consecutive, la priorità di tale processo scende di 1 punto dopo le prime due unità temporali e di 1 altro punto dopo le ultime due unità temporali.) (*Esempio 2.* Se un processo è in esecuzione per 3 unità di tempo consecutivamente, la priorità di tale processo scende di 1 punto dopo le prime due unità temporali e basta; l'altra unità temporale di esecuzione non concorre in alcun modo, nemmeno successivamente, a far decrementare la priorità del processo in considerazione.) Infine, la priorità di un processo non risale mai e non può scendere sotto lo zero.

Determinare, trascurando i ritardi dovuti allo scambio di contesto: (i) il tempo medio di risposta; (ii) il tempo medio di attesa; (iii) il tempo medio di *turn around*.
Nel caso di arrivi simultanei di processi allo stato di pronto, fatta salva l'eventuale considerazione del rispettivo valore di priorità, si dia la precedenza ai processi usciti dallo stato di esecuzione rispetto a quelli appena arrivati.
Nel caso di due processi aventi la stessa priorità, di cui uno in esecuzione, si dia la precedenza a quello in esecuzione.

[4.A]: *RR* (divisione di tempo, con priorità, senza prerilascio per priorità, e con quanto di tempo di ampiezza 2)

Proc. A																								
Proc. B																								
Proc. C																								
Proc. D																								
Proc. E																								

CPU																								
Coda																								

processo	t. risposta	t. attesa	*turn-around*
A			
B			
C			
D			
E			
medie			

[4.B]: *Fair Priority Scheduling* (con priorità 3, 5, 2, 4, 1 per i processi A, B, C, D, E, dove 5 è la massima priorità)

Proc. A																								
Proc. B																								
Proc. C																								
Proc. D																								
Proc. E																								

CPU																								
Coda																								

processo	t. risposta	t. attesa	*turn-around*
A			
B			
C			
D			
E			
Medie			

Quesito 5:

Si consideri un sistema dotato di memoria virtuale, con memoria fisica divisa in 4 page frame. Il tempo di caricamento, tempo di ultimo accesso e i *bit* di R (*Referred*) e M (*Modified*) per ogni pagina sono come mostrato nella tabella sottostante.

pagina	tempo caricamento	ultimo riferimento	R	M
0	132	286	1	0
1	236	251	0	1
2	154	267	0	0
3	91	301	1	1

Si supponga che il sistema abbia bisogno di caricare in memoria una nuova pagina. Giustificando (molto brevemente) la risposta, quale di quelle in tabella sarà rimpiazzata se si adotta una politica:

a) NRU

b) FIFO

c) LRU

d) second chance

Quesito 6:
[6.A] La seguente soluzione del problema dei lettori-scrittori contiene alcuni errori e mancanze. Lo studente ne modifichi il codice tramite aggiunte, cancellazioni e correzioni. Il risultato dovrà rappresentare una versione corretta, realizzata apportando il minor numero possibile di modifiche all'originale qui di seguito.
(Per coloro che avessero studiato solo sul libro di testo: *P*, corrisponde a *down*, *V* corrisponde a *up*)

```
void Lettore (void) {
  while (true) {
    numeroLettori++;
    if (numeroLettori==1) P(mutex);
    // leggi il dato
    numeroLettori--;
    if (numeroLettori==0) V(mutex);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    P(dati);
    // scrivi il dato
    V(dati);
  }
}
```

[6.B] Lo studente riporti qua sotto l'indicazione del tipo e del valore iniziale di ciascuna variabile.

Soluzione

Soluzione al Quesito 1

DOMANDA	Vero/Falso
In un sistema di memoria a paginazione, il Translation Lookaside Buffer (TLB) velocizza la traduzione di indirizzi virtuali in indirizzi fisici	V
La segmentazione consente a due processi di condividere un segmento	V
Se non vi sono percorsi chiusi in un grafo di allocazione allora non vi è situazione di stallo	V
FAT è un file system ad allocazione concatenata	F
La politica di scheduling round robin minimizza il tempo medio di attesa dei processi	F
Se un processo è in blocco da 10 ms significa che 10 ms fa ha eseguito una system call	V
Ogni interrupt può essere associato ad un processo che ha richiesto una operazione di I/O	F
Con NTFS è possibile che il file system scriva il contenuto di file di piccola dimensione (es. <1KB) direttamente nell'inode	F
`pwd` è un comando GNU/Linux per modificare la password	F

Soluzione al Quesito 2

Varie soluzioni possibili, ad esempio quella del filosofo mancino:

```
int semaforo f[i] = 1;

Filosofo(i) {
  while(1) {
     <pensa>
     if(i == X) {
        P(f [(i+1)%N]);
        P(f [i]);
     } else {
        P(f [i]);
        P(f [(i+1)%N]);
     }
     <mangia>
     V(f [i]);
     V(f [(i+1)%N]);
  }
}
```

Soluzione al Quesito 3

[3.A] No (per il perché si studi l'argomento su slide e/o libro)
[3.B] No (per il perché si studi l'argomento su slide e/o libro)
[3.C] Sì (per il perché si studi l'argomento su slide e/o libro)
[3.D] In questa soluzione useremo la notazione informatica tradizionale, con prefissi che denotano potenze di 2.
Essendo la memoria secondaria ampia 64 GB e i blocchi dati ampi 1 KB, è immediato calcolare
che sono necessari: $\left\lceil \frac{64GB}{1KB} \right\rceil$ = 64 M = $2^6 \times 2^{20} = 2^{26}$ indici, la cui rappresentazione binaria banalmente richiede 26 *bit*.
Stante l'ovvio vincolo che la dimensione dell'indice debba essere un multiplo di un "ottetto" (8 *bit*), otteniamo la dimensione di 32 *bit* (4 B).
Sotto queste ipotesi, il *file* di massima dimensione rappresentabile dall'architettura ext2fs fissata dal quesito sarà composto da:

- 12 blocchi, risultanti dall'utilizzo dei corrispondenti indici diretti presenti nell'i-node principale, al costo di 1 i-node, pari a 128 B
- $\left\lfloor \frac{128B}{4B} \right\rfloor$ = 32 blocchi, risultanti dall'utilizzo dell'intero i-node secondario denotato dall'indice di I indirezione presente nell'i-node principale, al costo di 1 i-node, pari a 128 B
- $32^2 = 2^{10}$ = 1 K blocchi, risultanti dall'utilizzo dell'indice di II indirezione, al costo di 1 + 32 = 33 i-node, pari a: 33 × 128B = (4.096 + 128)B = 4 KB + 128 B

- $32^3 = 2^{15} = 32$ K blocchi, risultanti dall'utilizzo dell'indice di III indirezione, al costo di $1 + 32 + 32^2 = 1.057$ i-node, pari a: 1.057 × 128 B = 128 KB + 4 KB + 128 B = 132 KB + 128 B corrispondenti a 12 + 32 + 1.024 + 32.768 = 33.836 blocchi ampi 1 KB, al costo complessivo di $1 + 1 + 33 + 33 + 32^2 = 1.092$

per un rapporto inflattivo di: $\frac{1.092 \times 128\,B}{33.836 \times 1\,KB} = \frac{1.092}{33.836 \times 8} \approx 0{,}40\%$.

Soluzione al Quesito 4

a) • RR con quanto di tempo di ampiezza 2

```
processo A      AAaaaaaaaaaaA                LEGENDA DEI SIMBOLI
processo B      -bBBBBBBB                    - non ancora arrivato
processo C      --ccccccccccCC               x (minuscolo) attesa
processo D      ------dddDDD                 X (maiuscolo) esecuzione
processo E      -------eeeeeeeeE             . coda vuota

CPU             AABBBBBBBDDDACCE
coda            .baaaadddaaacee.
                ..ccccaaaccce...
                ......ccceee....
                .......ee.......
```

processo	risposta	tempo di attesa	*turn-around*
A	0	10	10 + 3 = 13
B	1	1	1 + 7 = 8
C	11	11	11 + 2 = 13
D	3	3	3 + 3 = 6
E	8	8	8 + 1 = 9
medie	**4,60**	**6,60**	**9,80**

b) • Fair Priority Scheduling

```
processo A      AaaaaaaaaAA                  LEGENDA DEI SIMBOLI
processo B      -BBBBBbbbbbBB                - non ancora arrivato
processo C      --ccccccccccCC               x (minuscolo) attesa
processo D      ------DDD                    X (maiuscolo) esecuzione
processo E      -------eeeeeeeeE             . coda vuota

CPU             ABBBBBDDDAABBCCE
coda            .aaaaaaaabbccee.
                ..ccccbbbccee...
                ......cccee.....
                .......ee.......
```

Nota:
- All'istante 3, il valore di priorità di B passa da 5 a 4.
- All'istante 5, il valore di priorità di B passa da 4 a 3 (e viene dunque prerilasciato all'ingresso di D).
- All'istante 8, il valore di priorità di D passa da 4 a 3.

processo	risposta	tempo di attesa	*turn-around*
A	0	8	8 + 3 = 11
B	0	5	5 + 7 = 12
C	11	11	11 + 2 = 13
D	0	0	0 + 3 = 3
E	8	8	8 + 1 = 9
medie	**3,80**	**6,40**	**9,60**

Soluzione al Quesito 5

- NRU rimuove ovvero la pagina 2 perché è l'unica che abbia R = 0 e M = 0.
- FIFO rimuove la prima pagina che è stata caricata, ovvero la pagina 3.
- LRU rimuove la pagina 1 perché è quella riferita meno di recente tra quelle con R = 0.
- second chance rimuove la pagina più vecchia tra quelle con R = 0, ovvero la pagina 2.

Soluzione al Quesito 6

[6.A]

```
void Lettore (void) {
  while (true) {
    P(mutex);
    numeroLettori++;
    if (numeroLettori==1) P(dati);
    V(mutex);
    // leggi il dato
    P(mutex);
    numeroLettori--;
    if (numeroLettori==0) V(dati);
    V(mutex);
    // usa il dato letto
  }
}
```

```
void Scrittore (void) {
  while (true) {
    // prepara il dato da scrivere
    P(dati);
    // scrivi il dato
    V(dati);
  }
}
```

[6.B]

Non importa la sintassi…

int numeroLettori = 0
semaforo mutex = 1
semaforo database = 1