Quesito 1 (punti 8). Cinque processi *batch*, identificati dalle lettere $A - E$ rispettivamente, arrivano all'elaboratore agli istanti $0, 1, 3, 5, 8$ rispettivamente. Tali processi hanno un tempo di esecuzione stimato di $3, 7, 4, 6, 2$ unità di tempo rispettivamente. Per ognuna delle seguenti politiche di ordinamento:

1. FCFS (un processo per volta, sino al completamento)
2. RR (divisione di tempo, senza priorità e con quanto di tempo di ampiezza 2)
3. RR (divisione di tempo, con priorità e prerilascio, e quanto di tempo di ampiezza 2)
4. SJF_{SRTN} (senza considerazione di valori di priorità espliciti[1] e con prerilascio)

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*.
Ove la politica di ordinamento in esame consideri i valori di priorità, tali valori, mantenuti staticamente per l'intera durata dell'esecuzione, sono rispettivamente: $2, 3, 5, 3, 2$ (con 5 valore maggiore).
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.

Quesito 2 (punti 10).
In una pubblicazione del 1968 sulla rivista IBM Systems Journal, lo studioso James W. Havender propose una strategia per prevenire l'insorgere di situazioni di stallo (*deadlock*) in presenza di condivisione di risorse da parte di processi concorrenti. La strategia di Havender imponeva al sistema l'adozione di almeno una tra le seguenti condizioni:

1. ogni processo deve richiedere tutte le risorse necessarie in una sola volta e non può procedere finché non gli siano state tutte garantite
2. un processo che già detenga risorse a cui venga negato l'accesso a una ulteriore risorsa, deve rilasciarle tutte e, se necessario, richiederle tutte (inclusa quella aggiuntiva) di nuovo come stipulato al punto 1
3. a tutti i processi viene imposto un ordinamento lineare (p.es.: identificatore crescente) delle risorse: un processo che già detenga risorse, potrà richiederne altre solo in modo conforme all'ordinamento fissato (p.es.: il processo P_a che già possieda le risorse $\{R_1, R_3, R_6\}$ potrà richiedere la risorsa R_7 ma non la risorsa R_2).

Si discuta se e in che modo le condizioni proposte da Havender concorrano effettivamente a evitare strutturalmente lo stallo. In caso affermativo si discutano eventuali controindicazioni all'applicazione della strategia proposta.

Quesito 3 (punti 4). Si discuta concisamente come il fenomeno della frammentazione si manifesta in presenza di ciascuna delle seguenti strategie di organizzazione della memoria virtuale:

- segmentazione
- paginazione
- segmentazione paginata.

Quesito 4 (punti 10). Sia data una partizione di disco ampia 64 GB[2] organizzata in blocchi dati di ampiezza 1 kB. Sotto queste ipotesi si determini l'ampiezza massima di *file* ottenibile per l'architettura di *file system ext2fs* nel caso pessimo di contiguità nulla, 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 delle strutture di rappresentazione rispetto a quella dei dati veri e propri.
Effettuati tali calcoli si discuta se e con quale rapporto inflattivo le architetture *FAT* e *NTFS* rispettivamente possano rappresentare *file* di tale ampiezza nella partizione data, sotto le medesime ipotesi di contiguità nulla. Per l'architettura *NTFS* si assumano *record* ampi 1 kB, 408 B riservati all'attributo dati nel *record* principale e 800 B nei *record* di estensione.

[1] Esclusi ovviamente i valori di priorità impliciti determinati dalla durata (residua) dei processi.
[2] Nel quesito useremo la notazione informatica tradizionale, con prefissi che denotano potenze di 2.

Soluzione 1 (punti 8).

- FCFS (un processo per volta, sino al completamento)

```
processo A  AAA                      LEGENDA DEI SIMBOLI
processo B  -bbBBBBBBB               - non ancora arrivato
processo C  ---ccccccCCCC            x (minuscolo) attesa
processo D  -----ddddddddDDDDDD      X (maiuscolo) esecuzione
processo E  --------eeeeeeeeeeeeEE   . coda vuota

CPU         AAABBBBBBBCCCCDDDDDDEE
coda        .bbccccccddddeeeeee..
            .....dddddeeee........
            ........ee............
```

processo	tempo di risposta	tempo di attesa	*turn-around*
A	0	0	0+3= 3
B	2	2	2+7= 9
C	7	7	7+4=11
D	9	9	9+6=15
E	12	12	12+2=14
medie	6,00	6,00	10,40

- RR (divisione di tempo, senza priorità e con quanto di tempo di ampiezza 2)

```
processo A  AAaaA                    LEGENDA DEI SIMBOLI
processo B  -bBBbbbBBbbbbbbBBbbB     - non ancora arrivato
processo C  ---ccCCccccCC            x (minuscolo) attesa
processo D  -----ddddDDddddddDDdDD   X (maiuscolo) esecuzione
processo E  --------eeeeeEE          . coda vuota

CPU         AABBACCBBDDCCEEBBDDBDD
coda        .baacbbddcceebbddbbd..
            ...cbddcceebbdd.......
            ........ebbdd.........
```

processo	tempo di risposta	tempo di attesa	*turn-around*
A	0	2	2+3= 5
B	1	1+3+6+2=12	12+7=19
C	2	2+4= 6	6+4=10
D	4	4+6+1=11	11+6=17
E	5	5	5+2= 7
medie	2,40	7,20	11,60

- RR (divisione di tempo, con priorità e prerilascio, e quanto di tempo di ampiezza 2)

```
processo A  AaaaaaaaaaaaaaaaaaAA     LEGENDA DEI SIMBOLI
processo B  -BBbbbbBBbbBBbbB        - non ancora arrivato
processo C  ---CCCC                  x (minuscolo) attesa
processo D  -----ddddDDddDDdDD       X (maiuscolo) esecuzione
processo E  --------eeeeeeeeeeeeEE   . coda vuota

CPU         ABBCCCCBBDDBBDDBDDAAEE
coda        .aabbbbddbbddbbdaaee..
            ...aaddaaaaaaaaaee....
            .....aa.eeeeeeee......
```

processo	risposta	tempo di attesa	*turn-around*
A	0	17	17+3=20
B	0	4+2+2= 8	8+7=15
C	0	0	0+4= 4
D	4	4+2+1= 7	7+6=13
E	12	12	12+2=14
medie	3,20	8,80	13,20

- SJF_{SRTN} (senza considerazione di valori di priorità espliciti e con prerilascio)

```
processo A  AAA                     LEGENDA DEI SIMBOLI
processo B  -bbbbbbbbbbbbbbbBBBBB   - non ancora arrivato
processo C  ---CCCC                 x (minuscolo) attesa
processo D  -----ddDddDDDDD         X (maiuscolo) esecuzione
processo E  --------EE              . coda vuota

CPU         AAACCCCDEEDDDDDBBBBBB
coda        .bbbbbbbddbbbbb......
            .....dd.bb..........
```

processo	risposta	tempo di attesa	*turn-around*
A	0	0	0+3= 3
B	14	14	14+7=21
C	0	0	0+4= 4
D	2	2+2=4	4+6=10
E	0	0	0+2= 2
medie	3,20	3,60	8,0

Soluzione 2 (punti 10). Dalla diapositiva 33 del corso conosciamo le 4 condizioni che concorrono all'insorgere di situazioni di stallo. Per essere efficace, una strategia di prevenzione, e dunque anche quella proposta da Havender, deve essere capace di impedire il verificarsi di almeno una di tali condizioni. Per risolvere il quesito dobbiamo dunque studiare se e quale condizione di stallo ciascuno dei 3 requisiti proposti da Havender effettivamente previene:

- il requisito 1 previene l'accumulo parziale di risorse, perchè stipula che un processo possa eseguire (già dall'inizio) solo avendo acquisito tutte le risorse necessarie; naturalmente questa strategia è pesantemente inefficiente per almeno due motivi: (1) perché assegna più risorse di quante ne siano probabilmente necessarie, visto che l'assegnazione deve contemplare il caso di massima domanda, e per un tempo probabilmente più lungo del necessario; e (2) perché comporta per i processi il serio rischio di posticipazione indefinita
- il requisito 2 di fatto realizza il prerilascio preventivo delle risorse perché impegna il processo cui venisse negata una risorsa supplementare a rilasciare immediatamente tutte quelle in suo possesso; questa strategia ínefficiente perchè rischia di impedire al processo in questione di completare il proprio lavoro sulle risorse in proprio possesso e di doverlo ricominciare, eventualmente anche daccapo (a seconda del tipo di risorsa e dell'operazione interrotta) in un momento successivo
- il requisito 3 impedisce il formarsi di condizioni di attesa circolare; si lascia come esercizio al lettore la semplice verifica di questa asserzione. È facile comprendere che anche in questo caso l'applicazione di una tale strategia risulta troppo rigida per essere utilmente applicabile in un sistema reale.

Soluzione 3 (punti 4). Dalla diapositiva 160 del corso sappiamo che la scelta di un sistema di paginazione comporta il rischio di frammentazione interna poiché l'ampiezza di pagina scelta a priori può rivelarsi troppo grande per diversi tipi di contenuto.
Da pagina 168 invece sappiamo che la scelta di segmentazione comporta il rischio opposto, di frammentazione esterna perchè i segmenti, che hanno dimensione variabile e che debbono essere riversati interamente in memoria principale, potrebbero non trovare spazio sufficiente per ospitarli, con ciò lasciando inutilizzate porzioni preziose di memoria.

La segmentazione paginata riduce (fino ad annullarla) l'incidenza della frammentazione esterna, perché consente il caricamento parziale delle pagine, ma incorre nel rischio di frammentazione interna per via del ricorso alla paginazione.

Soluzione 4 (punti 10). Essendo la memoria secondaria ampia 64 GB e i blocchi dati ampi 1 kB, è immediato calcolare che sono necessari: $\lceil \frac{64\ GB}{1\ kB} \rceil = 64\ M = 2^6 \times 2^{20} = 2^{26}$ 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
- $\lfloor \frac{128B}{4B} \rfloor = 32$ 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: $33 \times 128B = (4.096 + 128)B = 4$ 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: $1.057 \times 128B = 128$ 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 $1 + 33 + 33 + 32^2 = 1.091$ *i-node* ampi 128 B, per un rapporto inflattivo di: $\frac{1.091 \times 128\ B}{33.836 \times 1\ kB} = \frac{1.091}{33.836 \times 8} = 0,40\%$.
Vediamo ora di determinare se e in che modo le architetture di *file system FAT* e *NTFS* siano in grado di rappresentare *file* di tale ampiezza sotto le ipotesi fissate dal quesito.

File system* di tipo *FAT : La struttura FAT, che rappresenta la vista dell'intera partizione in termini di blocchi dati, sarà composta da $\lceil \frac{64\ GB}{1\ kB} \rceil = 64\ M$ celle ampie 4 B, una per indice di blocco: di queste, il *file* che dobbiamo rappresentare ne occuperà 33.836, per un rapporto inflattivo — calcolato considerato che l'architettura *FAT* concettualmente usa l'intera struttura per ogni singolo *file* — pari a: $\frac{64\ M \times 4\ B}{33.836\ kB} = \frac{256\ MB}{33.836\ kB} = 774,75\%$, un onere piuttosto imponente.

File system* di tipo *NTFS : Dei 408 B riservati all'attributo dati nel *record* principale, $2 \times 4 = 8$ B saranno riservati alla coppia $\{$base, indice$\}$, mentre i rimanenti $408 - 8 = 400$ B potranno essere utilizzati per denotare le sequenze contigue che, sotto le ipotesi di contiguità nulla fissate del quesito, sono tutte ampie 1 blocco. Poiché ciascuna sequenza di tipo $\{$inizio, fine$\}$ richiede 8 B, il *record* principale potrà ospitare: $\lfloor \frac{400B}{8B} \rfloor = 50$, mentre un singolo *record* di estensione dispone di 800 B per la memorizzazione di: $\lfloor \frac{800B}{8B} \rfloor = 100$ ulteriori sequenze. Ne segue che, per rappresentare un *file* dell'ampiezza data, l'architettura *NTFS* necessiterà, in prima approssimazione, di: $1 + \lceil \frac{33.836 - 50 \text{ blocchi}}{100 \text{ blocchi/}\textit{record}} \rceil = 1 + \frac{33.786}{100} = 1 + 338 = 339$ *record*. Dobbiamo però tener conto del fatto che il *record* principale dovrà ritenere i puntatori a 338 *record* di estensione (in forma di attributi non residenti). Per stabilire l'ampiezza di spazio occupata da questa informazione supplementare dobbiamo determinare l'ampiezza di un indice di *record* MFT. In generale, Windows NT assegna $\frac{1}{8}$ della partizione alla MFT, nel nostro caso dunque $64\ GB \times \frac{1}{8} = 8\ GB$, che pertando contiene $\frac{8\ GB}{1\ kB/\textit{record}} = 8\ M$ *record*, indirizzabili con 23 *bit*, ma che diventano 4 B per il vincolo architetturale considerato più sopra. Per rappresentare 338 indici di estensione saranno necessari $338 \times 4 = 1.352$ B $> 1\ kB$, dunque certamente almeno 1 *record* aggiuntivo a quello principale. In conclusione, il *file* dato sarà rappresentato con $339 + 1 = 340$ *record*, per un rapporto inflattivo pari a: $\frac{340 \times 1\ kB}{33.836\ kB} = 1,00\%$.