Quesito 1 (punti 5). Si consideri un elaboratore dotato di un sistema di memoria virtuale paginata equipaggiato con un processore il cui ciclo di *clock* sia 1 MHz. Si assuma che il processore impieghi un ciclo di *clock* per eseguire istruzioni che non comportino riferimenti a pagine di memoria diverse da quella corrente. Si assuma inoltre che valgano le seguenti ipotesi:

- l'accesso a una pagina di memoria diversa dalla corrente comporti un costo temporale aggiuntivo di 1 μs;
- ciascuna pagina di memoria sia composta di 1.000 B;
- il disco fisso ruoti alla velocità di 3.000 rpm (rotazioni/minuto);
- il trasferimento tra disco e memoria avvenga a 1.000.000 B/secondo;
- il tempo medio per spostare la testina dalla posizione corrente fin sopra la traccia dove si trova il punto di lettura/scrittura su disco sia di 10 ms;
- un blocco su disco corrisponda in dimensione esattamente a una pagina di memoria virtuale;
- 99% delle istruzioni eseguite facciano riferimento alla pagina corrente;
- 80% delle pagine accedute (diverse dalla corrente) si trovi già in memoria;
- quando una nuova pagina debba essere caricata in memoria, nel 50% dei casi quella rimpiazzata sia stata precedentemente modificata.

Si calcoli il tempo medio effettivo di esecuzione di ciascuna istruzione su tale elaboratore assumendo che il sistema stia eseguendo un unico processo e che il processore rimanga inattivo (stato *idle*) durante i trasferimenti di dati.

Quesito 2 (punti 7). Cinque processi *batch*, identificati dalle lettere $A - E$ rispettivamente, arrivano all'elaboratore agli istanti $0, 2, 3, 6, 10$ rispettivamente. Tali processi hanno un tempo di esecuzione stimato di $3, 6, 4, 5, 1$ unità di tempo rispettivamente. Per ognuna delle seguenti politiche di ordinamento:

1. RR (divisione di tempo, senza priorità e con quanto di tempo di ampiezza 2)
2. RR (divisione di tempo, con priorità e prerilascio, e quanto di tempo di ampiezza 2)
3. $\mathsf{SJF}_{\mathsf{SRTN}}$ (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 3 (punti 4). Si consideri un sistema dotato di memoria virtuale, con memoria fisica divisa in 8 *page frame*, condivisa da 4 processi contemporaneamente attivi e denominati A, B, C e D rispettivamente. Si supponga che all'istante 100 lo stato della memoria sia quello riportato in tabella 1:

processo	pag. logica	pag. fisica	istante di caricamento	*bit* di riferimento
A	0	7	50	1
A	1	6	37	0
B	5	5	30	1
B	3	4	97	0
C	2	0	15	1
C	5	2	70	0
C	9	1	92	0
D	8	3	27	0

Tabella 1: Stato della memoria all'istante 100.

Si supponga che il sistema utilizzi un algoritmo di rimpiazzo delle pagine *second chance* (globale), e che la lista delle pagine accedute all'istante 100 sia:

C2(15,1) - D8(27,0) - B5(30,1) - A1(37,0) - A0(50,1) - C5(70,0) - C9(92,0) - B3(97,0)

con C2(15,1) primo elemento della lista; in tale elemento, C2 indica che si tratta della pagina logica 2 del processo C, mentre (15,1) indica che tale pagina è stata caricata in memoria all'istante 15 e che il suo *bit* di riferimento è uguale a 1.
Si considerino i due casi seguenti, eseguiti in alternativa:

[1] Esclusi ovviamente i valori di priorità impliciti determinati dalla durata (residua) dei processi.

1. all'istante 101, C riferisce la pagina logica 5
2. all'istante 101, A riferisce la pagina logica 9

Assumendo che, a parte quelle sopra elencate, non vi siano altre operazioni che modifichino i *bit* di riferimento, si scriva la lista delle pagine accedute aggiornata nei due casi dopo aver eseguito, alternativamente, le operazioni di cui ai punti 1 e 2.

Quesito 4 (punti 8). Un sistema multiprogrammato che includa risorse condivise dotate di istanze multiple è pericolosamente esposto al rischio dello stallo perchè le tecniche di prevenzione utilizzabili risultano essere di realizzazione complessa e onerosa e dunque poco attraenti sul piano prestazionale.
Una nota tecnica di prevenzione per tali sistemi utilizza il cosiddetto *algoritmo del banchiere*, il cui fine ultimo è assicurare che il sistema non assegni mai tutte le risorse a sua disposizione così da non restarne mai sguarnito e poter continuare a soddisfare le richieste dei propri processi.
Lo studente discuta brevemente, sulla base delle indicazioni fornite nel quesito, se e come l'*algoritmo del banchiere* sia effettivamente un mezzo di prevenzione dello stallo e poi progetti schematicamente le strutture dati e le procedure necessarie per realizzarlo.

Quesito 5 (punti 8). Un aspirante informatico, dotato di un elaboratore personale non recentissimo ma di buona qualità, con disco fisso ampio 80 GB a doppia partizione, decide di installare nella partizione predefinita (quella che il fornitore assegna d'autorità a MS Windows) un disco USB rimuovibile di ultima generazione con capienza 120 GB configurandolo a FAT, come consigliato dalle istruzioni del fornitore. L'operazione malauguratamente provoca un guasto di sistema che corrompe la tabella di configurazione della partizione (ciò che tecnicamente si chiama *partition boot sector*). All'avvio successivo l'aspirante informatico nota con sgomento che la partizione MS Windows non è più riconosciuta né dal programma di avvio (GRUB nel suo caso) né dall'interno della partizione GNU/Linux e dunque teme di aver perso per sempre tutti i programmi e i dati memorizzati nella partizione guasta. Studiando però la documentazione del suo sistema e facendo verifiche sul *Web* apprende che il disco di installazione di MS Windows contiene una utilità di recupero che include il comando FIXBOOT che potrebbe fare al caso suo.
Lo studente ipotizzi quale guasto possa essersi verificato in relazione all'installazione tentata dall'aspirante informatico del quesito, spieghi poi concisamente se e perchè la corruzione di informazione nel *partition boot sector* possa causare il guaio presentato nel quesito e indichi infine se, sotto quali ipotesi, e attraverso quali operazioni il comando FIXBOOT possa effettivamente ripristinare la situazione.

Soluzione 1 (punti 5). Il processore considerato riesce a eseguire una istruzione/μs qualora essa non comporti riferimenti a pagine di memoria diverse da quella corrente. L'esecuzione di ogni istruzione richiede quindi $\geq 1\mu s$.
Per ipotesi, l'1% delle istruzioni richiede 1 μs in più per accedere ad aree di memoria diverse da quella corrente; di queste, il 20% richiede tempo ulteriore per leggere dal disco un pagina di memoria virtuale e, nel 50% dei casi, la pagina rimpiazzata dovrà essere copiata su disco, aumentando quindi il tempo di esecuzione della istruzione corrispondente.
Il tempo medio di esecuzione risulta quindi dalla somma delle seguenti componenti:

- tempo di esecuzione su processore: 1 μs per tutte le istruzioni;
- tempo di accesso ad area di memoria diversa da quella corrente: 1 μs per l'1% delle istruzioni;
- tempo di lettura da disco della pagina referenziata: tempo medio di posizionamento, rotazione e trasmissione, per il 20% delle istruzioni che fanno riferimento a pagine diverse da quella corrente;
- tempo di scrittura su disco della pagina rimpiazzata: come per il punto precedente, ma solo nel 50% dei casi in cui si deve accedere al disco;

Il disco ruota a 3.000 rpm, ovvero 50 rotazioni/secondo; una rotazione completa richiede quindi 20 ms. Statisticamente, quando la testina del disco si muove per leggere o scrivere su un'altra traccia, si troverà a mezzo giro di distanza dalla posizione cercata, aggiungendo quindi un ulteriore ritardo medio di 10 ms (equivalmente a mezzo giro) a ogni spostamento di testina. A ogni lettura o scrittura su disco occorre dunque aggiungere il tempo di spostamento della testina (per ipotesi, in media 10 ms), il tempo di rotazione del disco necessario per posizionare la testina sul punto iniziale di lettura o scrittura (in media 10 ms) e infine il tempo di trasferimento della pagina, pari a $1.000\text{ B}/1.000.000\text{ B/s} = 1$ ms.
In totale quindi, ogni accesso a disco comporta un ulteriore ritardo di $10\text{ ms} + 10\text{ ms} + 1\text{ ms} = 21$ ms.
In conclusione, il tempo medio di esecuzione di un'istruzione risulta essere:

$$1\mu s + 0,01 \times [1\mu s + 0,2 \times (21\text{ ms} + 0,5 \times 21\text{ ms})] = 64,01\mu s.$$

Soluzione 2 (punti 7).

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

```
processo A  AAA                     LEGENDA DEI SIMBOLI
processo B  --bBBbbBBbbbbBB         - non ancora arrivato
processo C  ---ccCCccccCC           x (minuscolo) attesa
processo D  ------dddDDddddddDDD    X (maiuscolo) esecuzione
processo E  ----------eeeeeE        . coda vuota

CPU         AAABBCCBBDDCCBBEDDD
coda        ..bccbbddccbbeed...
            ......dccbbeedd....
            ..........edd......
```

processo	tempo di risposta	tempo di attesa	tempo di *turn-around*
A	0	0	0+3= 3
B	1	7	7+6=13
C	2	6	6+4=10
D	3	8	8+5=13
E	5	5	5+1= 6
medie	2,20	5,20	9,00

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

```
processo A  AAaaaaaaaaaaaaaaA       LEGENDA DEI SIMBOLI
processo B  --BbbbbBBbbBBbbB        - non ancora arrivato
processo C  ---CCCC                 x (minuscolo) attesa
processo D  ------dddDDddDDdD       X (maiuscolo) esecuzione
processo E  ----------eeeeeeeeE     . coda vuota
```

```
CPU         AABCCCCBBDDBBDDBDAE
coda        ..abbbbddbbddbbdae.
            ...aaadaaaaaaaaae..
            ......a...eeeeee...
```

processo	tempo di risposta	tempo di attesa	tempo di *turn-around*
A	0	15	15+3=18
B	0	8	8+6=14
C	0	0	0+4= 4
D	3	6	6+5=11
E	8	8	8+1= 9
medie	2,20	7,40	11,20

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

```
            0123456789012345678
processo A  AAA                     LEGENDA DEI SIMBOLI
processo B  --bbbbbbbbbbbBBBBBB    - non ancora arrivato
processo C  ---CCCC                 x (minuscolo) attesa
processo D  ------dDDDdDD           X (maiuscolo) esecuzione
processo E  ----------E             . coda vuota

CPU         AAACCCCDDDEDDBBBBBB
coda        ..bbbbdbbbdbb......
            ......b...b........
```

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

Soluzione 3 (punti 4).
Caso 1: La pagina C5 è già in memoria (nella pagina fisica 2) e quindi viene marcata come riferita
La lista risultante è dunque:
C2(15,1) - D8(27,0) - B5(30,1) - A1(37,0) - A0(50,1) - C5(70,1) - C9(92,0) - B3(97,0)
Caso 2: La pagina logica A9 non si trova in memoria, l'algoritmo *second chance* esamina C2: siccome è stata riferita, viene inserita in coda alla lista e il *bit* di riferimento viene posto a 0; dopodiché, si esamina D8, che non risulta riferita e viene dunque eliminata. La lista risultante è dunque:
B5(30,1) - A1(37,0) - A0(50,1) - C5(70,0) - C9(92,0) - B3(97,0) - C2(15,0) - A9(101,1)

Soluzione 4 (punti 8). Vediamo subito se e come il principio guida dell'*algoritmo del banchiere* come illustrato nel quesito costuitisca un mezzo di prevenzione dello stallo: se il sistema di gestione delle risorse riesce ad assicurare che vi siano sempre risorse disponibili per soddisfare le richieste di qualche processo (naturalmente non di tutti, visto che qualche processo potrebbe necessitare di più risorse di quante al momento disponibili) i processi soddisfatti completeranno il loro lavoro e rilasceranno (a meno di guasti o male intenzioni, che però esulano dal problema in esame) le risorse impiegate. La maggior disponibilità di risorse che consegue da tale rilascio consentirà il soddisfacimento delle richieste di altri processi continuando così ad alimentare l'attività del sistema impedendo di conseguenza che si possano creare situazioni di stallo (*deadlock*)[2].
L'algoritmo in se è di semplice concezione, mentre è la sua realizzazione è assai onerosa sia per i processi coinvolti che per la gestione delle risorse. Vediamo brevemente ciò che occorre:

[2]Sarà utile notare che questo algoritmo, senza speciali correttivi, non potrà invece impedire il formarsi di situazioni di *starvation* in virtù delle quali processi che necessitino di molte risorse non riescano mai a veder soddisfatte le proprie richieste perchè l'arrivo continuo di altri processi con basse pretese, immediatamente soddisfacibili, impedisce il formarsi di disponibilità sufficienti.

- informazione sulle disponibilità correnti di istanze per tipo di risorsa; questa struttura, che chiamiamo FondoDisponibile[] è un vettore di n posizioni, per n tipi di risorse nel sistema, a contenuto variabile, inizializzato alla massima disponibilità del sistema e poi aggiornato con l'andamento delle assegnazioni e dei rilasci;
- informazione sul numero massimo di risorse necessarie a ogni processo che si presenti al sistema, utile quindi per impedire che un processo possa richiedere e eventualmente cumulare più risorse del permissibile: chiamiamo FidoBancario[][] questa struttura dati a valori costanti per processo, che sarà una matrice $m \times n$, dove m sono i processi e n i tipi di risorsa; naturalmente il sistema non ammetterà il processo i con richieste $[i][j]$ (per qualche tipo j di risorsa) superiori alla disponibilità massima stipulata in FondoDisponibile[j];
- informazione sulla quantità di istanze di risorse attualmente conferite in prestito a ogni processo operante nel sistema; questa struttura dati, che chiamiamo PrestitoCorrente[][], sarà una matrice $m \times n$ il cui valore $[i][j]$ denota quante istanze della risorsa j siano attualmente in uso al processo i; questo valore non potrà mai eccedere il limite fissato in FidoBancario[i][j];
- informazione sul residuo di prestito di risorse potenzialmente esigibile da ogni processo entro il proprio tetto di fido; questa struttura dati, che chiamiamo LimiteResiduo[][], sarà una matrice $m \times n$, il cui valore $[i][j]$ risulta da: FidoBancario[i][j] $-$ PrestitoCorrente[i][i].

Il sistema dovrà assicurare che, a ogni istante, per ogni tipo di risorsa, vi sia almeno un processo i, per il quale valga: FondoDisponibile[j] $\geq$ LimiteResiduo[i][j]. In tal modo infatti, (almeno) il processo i potrà ottenere il prestito, terminare il proprio lavoro e restituire tutte le risorse in suo possesso, aumentandone quindi la disponibilità a vantaggio dei processi residui. È immediato vedere che l'onerosità di questo algoritmo risulta dall'esigenza di tenere aggiornate in corso di esecuzione le 4 strutture sopra descritte e nel penalizzare potenzialmente i processi con richieste elevate a vantaggio di quelli con esigenze più minute.

Soluzione 5 (punti 8). L'operazione tentata dall'aspirante informatico evocato dal quesito comporta una modifica importante dei dati di configurazione del sistema (all'interno della partizione MS Windows) per recepire la presenza di un nuovo disco esterno. Questa modifica ha effetto sul *boot record* della partizione MS Windows che deve essere capace da ora in poi di riconoscere il nuovo dispositivo come disco esterno aggiuntivo. Il guasto che si è presumibilmente verificato è dunque stato la scrittura di valori non congruenti nel *partition boot sector*. Questa scrittura rende conseguentemente incongruente e dunque illeggibile dall'esterno il *partition boot sector*, così che né il programma di avvio né l'altra partizione attiva sul disco dell'elaboratore possono decrifrarne il contenuto.
A lezione abbiamo studiato due architetture di *file system* in ambito MS Windows: FAT e NTFS. FAT non prevede alcun tipo di ridondanza interna e dunque la corruzione del suo *boot sector* comporta la perdita definitiva di tutto il suo contenuto. Non a caso infatti il comando FIXBOOT non è utilizzabile per il ripristino di partizioni FAT. Per fortuna invece, NTFS prevede una preziosa forma di ridondanza che comporta la duplicazione dell'intera MFT, e dunque anche delle informazioni contenute nel *boot sector*, in un'altra posizione della partizione (per esempio a metà). È proprio grazie a questa duplicazione che il comando FIXBOOT può ripristinare il *partition boot sector* di competenza, consentendo così di nuovo l'avviamento del sistema, (quasi) come se nulla fosse accaduto. Nell'interesse della salute fisica e mentale dell'aspirante informatico del quesito dobbiamo dunque augurargli che la sua partizione MS Windows fosse configurata come NTFS.