Quesito 1. Sei processi, idendificati dalle lettere A-F, arrivano all'elaboratore agli istanti di tempo di valore 0, 2, 7, 9, 11 e 13 rispettivamente. I processi hanno un tempo stimato di esecuzione di 4, 6, 10, 5, 8 e 2 unità di tempo rispettivamente, mentre i loro valori di priorità statica (per le politiche che ne facciano uso) valgono rispettivamente 2, 3, 4, 1, 6 e 5 (dove 6 è il valore della priorità massima). Per ognuno dei seguenti algoritmi di ordinamento eseguiti dal *dispatcher*, determinare il tempo medio di completamento (*turnaround*) ed il tempo medio di attesa, trascurando i tempi dovuti allo scambio di contesto.

- Priorità senza prerilascio (un processo per volta, fino al completamento)
- Priorità con prerilascio
- *First-Come-First-Served* (FCFS) senza prerilascio (un processo per volta, fino al completamento, indipendentemente dalla priorità)
- *Shortest-Job-First* (SJF) senza prerilascio (un processo per volta, fino al completamento, indipendentemente dalla priorità).

Quesito 2. I sistemi operativi moderni hanno, tra gli altri, il problema di contenere il numero di pagine che ciascun processo ha chiamato in memoria principale. In generale, il problema si compone di due parti: localizzare le pagine che possono temporaneamente essere spostate su disco e rimuovere permanentemente (ove necessario prevedendo riscrittura su disco) quelle non più in uso. È prassi comune che entrambe tali attività vengano svolte da processi speciali di sistema, chiamati *daemon*. Si illustri, usando non più di 5 righe per ogni risposta:

1. cosa distingue un *daemon* da un normale processo
2. quale corrispondenza occorra preservare o creare tra le pagine che si trovano in memoria principale ed il corrispondente spazio su disco, precisando dove quest'ultimo sia inteso come permanente ovvero temporaneo
3. quali differenze intercorrano tra la tecnica della *paging partition* e quella del *paging file*, indicando quella che tra esse comporti il miglior rapporto costi/benefici.

Quesito 3. Alcune versioni di del sistema operativo UNIX, di cui Linux è derivato ed evoluzione, utilizzano l'algoritmo detto del *two-handed clock* (ovvero dell'orologio a doppia lancetta) per verificare empiricamente quali pagine attive in memoria principale siano buone candidate per il trasferimento temporaneo su disco. Tale algoritmo è una variante dell'"algoritmo dell'orologio". Si discutano, in non più di 10 righe, le differenze funzionali tra l'algoritmo di base e la sua variante "a doppia lancetta", illustrando la ragione per la quale la variante è generalmente preferibile.

Quesito 4. Il progettista di un sistema operativo ha deciso di usare nodi indice (*i-node*) per la realizzazione del proprio *file system*, stabilendo che essi abbiano la stessa dimensione di un blocco del disco, fissata a 512 *byte*. Il nodo indice primario è organizzato in modo da contenere 12 campi di indirizzo di blocchi (indirizzamento diretto), e tre campi puntatori a nodi indice di primo, secondo e terzo livello di indirezione rispettivamente. Utilizzando questa organizzazione e, sapendo che un indirizzo di blocco occupa 32 *bit*, si vuole allocare un *file* di 13.000 *record* da 84 *byte* ciascuno, imponendo che un *record* non possa essere suddiviso su due blocchi. Calcolare quanti blocchi verranno utilizzati per allocare il *file* e quanti per gestire la sua allocazione tramite *i-node*. Determinare inoltre l'occupazione totale in memoria secondaria.

Quesito 5. Il protocollo TCP di TCP/IP, che, come sappiamo si colloca, in ambito *Internet*, all'equivalente del livello Trasporto del modello OSI, deve gestire, per ogni comunicazione in uscita, il rischio di congestione della capacità trasmissiva della rete e della capacità recettiva del destinatario.

1. Si spieghi, in non più di 5 righe, se tali due problematiche di congestione siano da considerare indipendenti l'una dall'altra ovvero se esse abbiano dipendenze od influenze reciproche.

Al fine di gestire tali problematiche, il protocollo TCP usa un'euristica (ossia una forma di approssimazione costruttiva di una soluzione che potrebbe anche non esistere) che utilizza, in modo correlato, tre parametri fondamentali: un indicatore della capacità recettiva del destinatario (*receiver window*), che viene aggiornato grazie ad un apposito parametro incluso in ogni pacchetto di ritorno inviato dal destinatario sulla connessione; un indicatore della capacità trasmissiva della rete (*congestion window*), che viene aggiornato, euristicamente, mediante un algoritmo detto *slow start*; ed un indicatore di limite di rischio (*threshold*), che viene usato come fattore di limitazione nell'applicazione dell'algoritmo *slow start*.

2. Si illustri, in un formalismo a piacere (pseudo-codice, diagramma di flusso, testo libero, altro), il modo nel quale i tre parametri sopra indicati sono posti in relazione dall'euristica di controllo di congestione del protocollo TCP.

Quesito 6. Lo schema logico riportato in figura 1 rappresenta la rete dati di una piccola azienda composta da due reparti operativi ed una stanza per i gestori della rete.

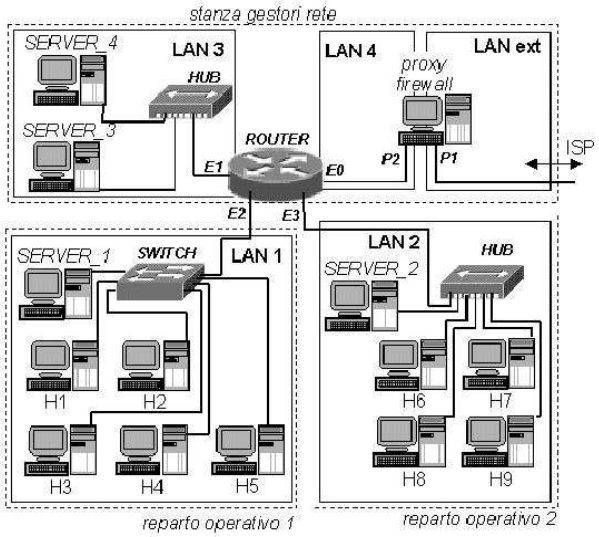

Figura 1: Schema della rete dati dell'azienda.

Il reparto operativo 1 è composto da 5 postazioni di lavoro (H1-H5) il cui traffico è prevalentemente di tipo utente-*server*, facenti capo al SERVER_1, mentre le 4 postazioni H6-H9 del reparto operativo 2 sono caratterizzate da un traffico prevalente tra utente e *server*, che fanno capo al SERVER_2. Gli ulteriori due *server* (3 e 4) posizionati nella stanza gestori rete offrono servizi *Web* accessibili rispettivamente ad utenti interni all'azienda (*Intranet Server*) e ad utenti esterni all'azienda (*Internet Server*). A tutti gli utenti interni, con esclusione dei server, deve essere anche garantito un traffico di accesso all'*Intranet Server* di 200 Kbps e alla rete *Internet* di 100 Kbps. Sapendo che tutti i dispositivi di rete dell'azienda sono di standard *Fast-Ethernet*, e quindi funzionano a 100 Mbps, si calcolino i flussi di traffico massimo determinati dalla configurazione *hardware* della rete nel caso peggiore di attività simultanea di tutti gli utenti.
L'azienda ha acquistato dal proprio ISP solo il seguente indirizzo statico di accesso ad *Internet*:

- *host address* = 122.141.11.241
- *subnet mask* = 255.255.255.128
- *default gateway* = 122.141.11.254

ed al suo interno vuole sfruttare la tecnologia NAT (*Network Address Translation* presente nel *proxy/firewall* e distribuire gli indirizzi della rete privata 192.168.0.0 tra le proprie sottoreti, così da assicurare ad ognuna di esse (tenendo presente eventuali sviluppi futuri) esattamente 30 indirizzi IP. Si proponga un possibile piano di assegnazione degli indirizzi IP a tutti i dispositivi della rete, individuando per ognuno la coppia *IP-address/subnet-mask*.

Soluzione 1 (punti 5).
Di seguito si presenta la soluzione in forma testuale, osservando però che essa ha varie rappresentazioni grafiche assai più compatte, pertanto generalmente preferibili.

Priorità senza prerilascio (un processo per volta, fino al completamento):
◇ T = 0 (arrivo del processo A, inizio esecuzione di A): a questo istante, da cui si inizia a valutare lo stato dello schedulatore di basso livello, nella lista dei pronti è presente solo il processo A, che lo schedulatore manderà pertanto in esecuzione.
◇ T = 2 (arrivo del processo B): arriva B, con priorità maggiore di A ($Pr_B = 3 > Pr_A = 2$), ma dato che la politica di ordinamento in esame esclude il prerilascio e prevede il completamento del processo in esecuzione, il S.O. lascia terminare il processo A.
◇ T = 4 (termine del processo A, inizio esecuzione di B): il processo A termina l'esecuzione passando allo stato di terminato. Nella coda dei processi pronti vi è solo il processo B, che lo schedulatore manda in esecuzione, senza valutare la sua priorità.
◇ T = 7 (arrivo del processo C): arriva il processo C, che viene posto al primo posto nella coda dei processi pronti che al suo arrivo è vuota, senza che la sua priorità sia comparata con quella del processo in esecuzione,dato che la politica non prevede prerilascio, cosa che avverrà per ogni altro successivo processo nelle stesse condizioni di C.
◇ T = 9 (arrivo del processo D): arriva il processo D, che viene posto al secondo posto nella coda dei processi pronti, dato che la sua priorità vale $Pr_D = 1 < Pr_C = 4$ dove C è già presente in lista.
◇ T = 10 (termine del processo B, inizio esecuzione di C): il processo B termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, per definizione quello a priorità maggiore tra quelli in attesa, ossia il processo C.
◇ T = 11 (arrivo del processo E): arriva il processo E, e viene posto al primo posto nella coda dei processi pronti, dato che la sua priorità vale $Pr_E = 6 > Pr_D = 1$, dove D è già presente nella lista.
◇ T = 13 (arrivo del processo F): arriva il processo F, e viene posto al secondo posto nella coda dei processi pronti, vista la sua priorità $Pr_F = 5 < Pr_E = 6$, che è però superiore a quella del processo D ($Pr_D = 1$), entrambi già presenti nella lista.
◇ T = 20 (termine del processo C, inizio esecuzione di E): il processo C termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, ossia il processo E.
◇ T = 28 (termine del processo E, inizio esecuzione di F): il processo E termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo F.
◇ T = 30 (termine del processo F, inizio esecuzione di D): il processo F termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti l'ultimo processo rimasto, cioè il processo D, e lo avvia in esecuzione.
◇ T = 35 (termine del processo D): il processo D termina l'esecuzione passando allo stato di terminato. Il S.O non trova più processi utente in attesa nella coda dei processi pronti, e quindi manda in esecuzione i processi di sistema in attesa.
Riepilogando otteniamo:

dispatcher con politica a priorità senza prerilascio

processo	arrivo	esecuzione		tempo di	tempo di
		inizio	fine	turn-around	attesa
A	0	0	4	4	0
B	2	4	10	8	2
C	7	10	20	13	3
D	9	30	35	26	21
E	11	20	28	17	9
F	13	28	30	17	15
			valori medi:	**14,16**	**8,33**

Priorità con prerilascio:
◇ T = 0 (arrivo del processo A, inizio esecuzione di A): a questo istante, nella lista dei pronti è presente solo il processo A, che lo schedulatore manderà pertanto in esecuzione.
◇ T = 2 (arrivo del processo B, sospensione di A, inizio esecuzione di B): arriva il processo B, la cui priorità ($Pr_B = 3$) è maggiore di quella del processo in esecuzione (processo A, con priorità $Pr_A = 2$); il S.O. forza pertanto il processo A nello stato di pronto, inserendolo al primo posto della relativa coda, in quanto unico processo in attesa, passando B in esecuzione.
◇ T = 7 (arrivo del processo C, sospensione di B, inizio esecuzione di C): arriva il processo C, la cui priorità ($Pr_C = 4$) è maggiore di quella del processo in esecuzione (processo B, con priorità $Pr_B = 3$); il S.O. forza

pertanto il processo B nello stato di pronto, inserendolo al primo posto della relativa coda, in quanto la sua priorità è superiore rispetto all'altro processo in attesa (processo A, $Pr_A = 2$), passando C in esecuzione.
◇ T = 9 (arrivo del processo D): arriva il processo D, la cui priorità ($Pr_D = 1$) è inferiore a quella del processo in esecuzione (processo C, $Pr_C = 4$); il S.O. lo pone pertanto in stato di pronto inserendolo al terzo posto della relativa coda, in quanto esso ha priorità inferiore rispetto agli altri due processi in attesa (processo B, $Pr_B = 3$; processo A, $Pr_A = 2$).
◇ T = 11 (arrivo del processo E, sospensione di C, inizio esecuzione di E): arriva il processo E, la cui priorità ($Pr_E = 6$) è maggiore di quella del processo in esecuzione (processo C, $Pr_C = 4$); il S.O. forza pertanto il processo E nello stato di pronto, inserendolo al primo posto della relativa coda, dato che esso ha priorità superiore rispetto agli altri tre processi in attesa (processo B, $Pr_B = 3$; processo A, $Pr_A = 2$; processo D, $Pr_D = 1$).
◇ T = 13 (arrivo del processo F): arriva il processo F, la cui priorità ($Pr_F = 5$) è inferiore a quella del processo in esecuzione (processo E, $Pr_E = 6$); il S.O. lo pone pertanto in stato di pronto e lo inserisce al primo posto della relativa coda, dato che ha priorità superiore rispetto agli altri quattro processi in attesa (processo C, $Pr_C = 4$; processo B, $Pr_B = 3$; processo A, $Pr_A = 2$; processo D, $Pr_D = 1$).
◇ T = 19 (termine del processo E, inizio esecuzione di F): il processo E termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, che è per definizione quello a priorità maggiore tra quelli attualmente presenti, ovvero il processo F.
◇ T = 21 (termine del processo F, ripristino esecuzione di C): il processo F termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, che è il processo C.
◇ T = 27 (termine del processo C, ripristino esecuzione di B): il processo C termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, che è il processo B.
◇ T = 28 (termine del processo B, ripristino esecuzione di A): il processo B termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, che è il processo A.
◇ T = 30 (termine del processo A, inizio esecuzione di D): il processo A termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti l'ultimo processo rimastovi, che è il processo D.
◇ T = 35 (termine del processo D): il processo D termina l'esecuzione passando allo stato di terminato. Il S.O non trova più processi utente in attesa nella coda dei processi pronti, e quindi manda in esecuzione i processi di sistema in attesa.
Riepilogando otteniamo:

dispatcher con politica a priorità con prerilascio

processo	arrivo	esecuzione				tempo di turnaround	tempo di attesa
		inizio	sospeso	riprende	fine		
A	0	0	2	28	30	30	26
B	2	2	7	27	28	26	20
C	7	7	11	21	27	20	10
D	9	30	-	-	35	26	21
E	11	11	-	-	19	8	0
F	13	19	-	-	21	8	6
					valori medi	**19,66**	**13,83**

First Come First Served (FCFS) senza prerilascio:
◇ T = 0 (arrivo del processo A, inizio esecuzione di A): a questo istante, nella lista dei pronti è presente solo il processo A, che lo schedulatore manderà pertanto in esecuzione.
◇ T = 2 (arrivo del processo B): consentendo la politica di ordinamento in esame di tipo FCFS il completamento del processo attualmente esecuzione, all'arrivo di B il S.O. lascia terminare il processo A ponendo B al primo posto della coda dei processi pronti.
◇ T = 4 (termine del processo A, inizio esecuzione del processo B): il processo A termina l'esecuzione passando allo stato di terminato. Poiché nella coda dei processi pronti vi è solo il processo B, lo schedulatore lo manda in esecuzione.
◇ T = 7 (arrivo del processo C): arriva il processo C, che viene inserito nella coda dei processi pronti, attualmente vuouta. ◇ T = 9 (arrivo del processo D): arriva il processo D, e viene inserito al secondo posto nella coda dei processi pronti, dato che essa gia contiene il processo C, che precede D per ordine di arrivo in stato di pronto.
◇ T = 10 (termine del processo B, inizio esecuzione di C): il processo B termina l'esecuzione passando allo stato di terminato. Lo schedulatore manda in esecuzione il processo C, in cima alla coda dei pronti, senza curarsi della sua priorità relativa.

◇ T = 11 (arrivo del processo E): arriva il processo E, che viene inserito al secondo posto nella coda dei processi pronti, dato che essa già contiene il processo D, che precede E per ordine di arrivo in stato di pronto.
◇ T = 13 (arrivo del processo F): arriva il processo F, che viene inserito al terzo posto nella coda dei processi pronti, dato che essa già contiene i processi D ed E.
◇ T = 20 (termine del processo C, inizio esecuzione di D): il processo C termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo D.
◇ T = 25 (termine del processo D, inizio esecuzione di E): il processo D termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo E.
◇ T = 33 (termine del processo E, inizio esecuzione di F): il processo E termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti l'ultimo processo presente, cioè il processo F.
◇ T = 35 (termine del processo F): il processo C termina l'esecuzione passando allo stato di terminato. Il S.O non trova più processi utente in attesa nella coda dei processi pronti, e quindi manda in esecuzione i processi di sistema in attesa.
Riepilogando otteniamo:

dispatcher con politica FCFS

processo	arrivo	esecuzione		tempo di turn-around	tempo di attesa
		inizio	fine		
A	0	0	4	4	0
B	2	4	10	8	2
C	7	10	20	13	3
D	9	20	25	16	11
E	11	25	33	22	14
F	13	33	35	22	20
			valori medi :	**14,16**	**8,33**

Shortest Job First (SJF) senza prerilascio:
◇ T = 0 (arrivo del processo A, inizio esecuzione di A): a questo istante, nella lista dei pronti è presente solo il processo A, che lo schedulatore manderà pertanto in esecuzione.
◇ T = 2 (arrivo del processo B): arriva B, e dato che la politica di ordinamento prevede il completamento del processo attualmente in esecuzione, il S.O. lo lascia terminare ponendo B al primo posto della coda dei processi pronti.
◇ T = 4 (termine del processo A, inizio esecuzione del processo B): il processo A termina l'esecuzione passando allo stato di terminato. Poiché la coda dei processi pronti contiene solo il processo B, lo schedulatore lo manda in esecuzione.
◇ T = 7 (arrivo del processo C): arriva il processo C, che viene inserito al primo posto nella coda dei processi pronti, essendo attualmente l'unico in attesa.
◇ T = 9 (arrivo del processo D): arriva il processo D, che viene inserito al primo posto nella coda dei processi pronti, dato che la sua durata è di 5 unità di tempo, inferiore a quella dell'altro processo in coda (processo C, 10 unità).
◇ T = 10 (termine del processo B, inizio esecuzione di D): il processo B termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo D.
◇ T = 11 (arrivo del processo E): arriva il processo E, che viene inseritoal primo posto nella coda dei processi pronti, dato che la sua durata è di 8 unità di tempo, inferiore a quella dell'altro processo in coda (processo C, 10 unità).
◇ T = 13 (arrivo del processo F): arriva il processo F, che viene inserito al primo posto nella coda dei processi pronti, dato che la sua durata è di 2 unità di tempo, inferiore a quella degli altri due processi in coda (processo E, 8 unità; processo C, 10 unità).
◇ T = 15 (termine del processo D, inizio esecuzione di F): il processo D termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo F.
◇ T = 17 (termine del processo F, inizio esecuzione di E): il processo F termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti il primo processo, cioè il processo E.
◇ T = 25 (termine del processo E, inizio esecuzione di C): il processo E termina l'esecuzione passando allo stato di terminato. Il S.O preleva dalla coda dei processi pronti l'ultimo processo presente, cioè il processo C.
◇ T = 35 (termine del processo C): il processo C termina l'esecuzione passando allo stato di terminato. Il S.O non trova più processi utente in attesa nella coda dei processi pronti, e quindi manda in esecuzione i processi di sistema in attesa.
Riepilogando otteniamo:

dispatcher con politica SJF

processo	arrivo	esecuzione		tempo di turn-around	tempo di attesa
		inizio	fine		
A	0	0	4	4	0
B	2	4	10	8	2
C	7	25	35	28	18
D	9	10	15	6	1
E	11	17	25	14	6
F	13	15	17	4	2
			valori medi:	**10,66**	**4,83**

Soluzione 2 (punti 5).

Risposta 1. Si definisce *daemon* un processo di sistema la cui attivazione è indipendente (e tipicamente periodica) dalle attività degli utenti. Il *garbage collector* è un esempio di *daemon* entro la JVM vista come sistema operativo.

Risposta 2. Lo spazio di memoria virtuale associato ad un processo tipicamente comprende: il suo codice, con i dati ad esso associati, i *file* in accesso, ed uno spazio di lavoro. Quest'ultimo, a differenza delle altre componenti elencate, è per definizione temporaneo e, pertanto, privo di corrispondente su disco. Quando porzioni (pagine) di tale spazio di lavoro devono essere temporaneamente spostate su disco, per esempio alla sospensione del processo possessore, il sistema operativo dovrà creare una corrispondenza fittizia con un *file* artificiale ovvero con una partizione apposita.

Risposta 3. Le tecniche menzionate nel quesito sono precisamente quelle generalmente utilizzate per assicurare il salvataggio temporaneo di pagine di lavoro prive di corrispondenze permanenti su disco. La tecnica della *paging partition* utilizza allo scopo una partizione riservata del disco, vista e gestita come una sequenza indistinta di *byte*. La tecnica del *paging file*, invece, utilizza un *file* speciale su disco come collettore di tutte le pagine temporanee attive provenienti dalla memoria principale. È evidente che quest'ultima tecnica comporti maggiori oneri e vincoli di utilizzo, che derivano dai limiti (per esempio dimensione massima e struttura interna) imposti su di esso dal *file system*.

Soluzione 3 (punti 4). L'algoritmo di base vede l'insieme delle pagine come una lista circolare, che viene periodicamente scandita da un solo indice. Alla prima passata, l'algoritmo azzera un *bit* di riferimento per ciascun indice di pagina. Alla passata successiva, l'algoritmo designa come rimuovibili tutte le pagine che non siano state nel frattempo riferite. L'intervallo che intercorre tra la prima e la seconda passata è ovviamente pari al tempo di una rivoluzione completa della lancetta, ossia di scansione dell'intera lista di pagine, il che può essere un tempo così lungo da consentire a tutte le pagine, anche quello di uso infrequente, di essere accedute almeno una volta e dunque di non essere identificate come rimuovibili. La variante "della doppia lancetta" si propone di evitare questo problema, usando la prima lancetta per l'azzeramento dei *bit* di controllo e la seconda per la verifica di avvenuto accesso, ponendo le due ad una distanza iniziale (e costante) configurabile al valore dell'intervallo temporale desiderato tra la prima e la seconda passata presso ciascuna pagina.

Soluzione 4 (punti 4). Richiamiamo i dati del problema:
N_R = numero di *record* che compongono il *file* = 13.000
D_R = dimensione di un *record* = 84 *byte*
D_I = dimensione di un indirizzo = 32 *bit* = 4 *byte*
D_B = dimensione di un blocco = 512 *byte*
N_{RB} = numero di *record* per blocco = $int(D_B/D_R) = int(512/84) = int(6,09) = 6$
N_{BF} = numero di blocchi occupati dal *file* = $\lceil N_R/N_{RB} \rceil = \lceil 13.000/6 \rceil = 2.167$
N_{IB} = numero di indirizzi in un blocco = $D_B/D_I = 512/4 = 128$.
I blocchi da indirizzare sono $N_{BF} = 2.167$:

- di questi, 12 possono essere indirizzati direttamente dall'*i-node* principale
- dei rimanenti $2.167 - 12 = 2.155$, N_{IB} (cioè 128) sono indirizzabili indirettamente tramite l'indirizzo di indirezione di primo livello dell'*i-node* principale
- dei rimanenti $2.155 - 128 = 2.027$, si possono utilizzare $1 + int(2027/128) = 16$ blocchi da 128 indirizzi indiretti ciascuno tramite il meccanismo di indirezione doppia, di cui i primi 15 saranno utilizzati

completamente (cioè per $128 \times 15 = 1.920$ indirizzi) mentre i rimanenti $2.027 - 1.920 = 107$ indirizzi andranno posizionati nel 16^o blocco, come riportato in figura 2.

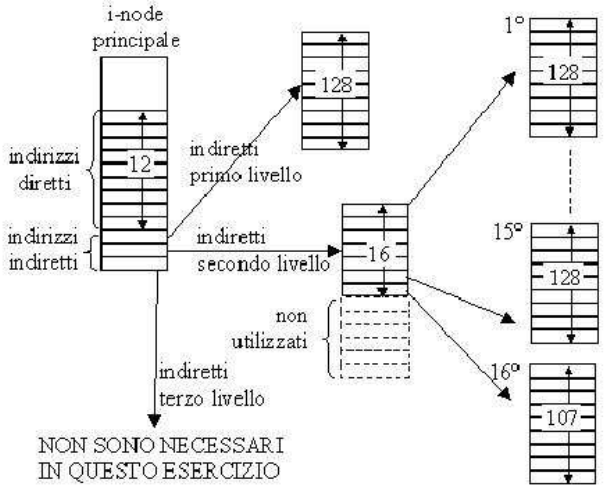

Figura 2: Allocazione del *file*.

Con le indicazioni riportate in figura possiamo concludere che:

- per allocare il *file* dati sono necessari N_{BF} blocchi, ossia 2.167 blocchi
- per gestire l'allocazione del *file* sono necessari:
  1 blocco per l'*i-node* principale
  1 blocco di indirizzi ad indirezione di primo livello
  $1 + 16$ blocchi per l'indirezione di secondo livello
  per un totale di $1 + 1 + 1 + 16 = 19$ blocchi
- l'occupazione totale in memoria secondaria vale $2.167 + 19 = 2.186$ blocchi da 512 *byte*, per un totale di $2.186 \times 512 = 1.119.232$ *byte* $= 1.093$ K*byte*, equivalenti a circa 1 M*byte*.

Soluzione 5 (punti 7).

Risposta 1. La capacità recettiva del destinatario in una qualunque connessione TCP non ha nulla a che fare con la capacità trasmissiva della rete, in quanto essa dipende esclusivamente dalla natura dell'applicazione destinataria (se, per esempio, tratta un solo carattere alla volta, indipendentemente dalla dimensione del pacchetto ricevuto) e con lo spazio di memoria assegnata al salvataggio dei pacchetti pervenuti al nodo, ma non ancora accettati dal destinatario.
Per contro, la capacità trasmissiva della rete *Internet* è legata a fenomeni prevalentemente asincroni ed impredicibili, e fortemente influenzata dall'uso di tecniche di "inondazione" (*flooding*) nell'emissione di duplicati di pacchetti in uscita.

Risposta 2. La figura 3 riporta lo schema di base dell'euristica di controllo di congestione utlizzata dal protocollo TCP, dove si vede come, iterativamente, si tenti di approssimare l'ampiezza dei pacchetti inviati alla capacità massima tollerata sia dal destinatario (in base a quanto da esso stesso dichiarato) che dalla rete (in base a quanto osservato). Si noti come, ad ogni evento *time-out*, che denota la mancata ricezione di conferma di un pacchetto precedentemente inviato sulla connessione, la soglia di correzione (*Threshold*) venga prudenzialmente dimezzata e l'ampiezza di pacchetto riportata al valore iniziale.

Soluzione 6 (punti 7). Il *router* separa le reti locali, isolandone i domini di diffusione distinti. Per il calcolo dei flussi nel caso peggiore possiamo pertanto analizzare il traffico separatamente per ogni LAN.
Per prima cosa, individuiamo i flussi utili, ossia quelli indicati dal testo. Detti:

```
main(){ /* i valori sono espressi in kbyte */
    int Threshold = 64; /* inizializzata alla dimensione massima di un pacchetto */
    int RW = 64; /* valore iniziale di Receiver Window, aggiornato ad ogni ricezione di pacchetti di ritorno dal destinatario */
    const CS = 1; /* valore iniziale di capacità trasmissiva della connessione */
    int CW = CS; /* valore iniziale di Congestion Window, da aggiornare mediante Slow Start, moderato da Threshold */
    int Exponential; /* variabile di appoggio per l'algoritmo Slow Start */
    int Linear;      /* variabile di appoggio per l'algoritmo Slow Start */
    int TimeOut = 0; /* variabile di servizio */
    int i;
    for(i=1;i<= ... ;i++){ /* in realtà, occorrerebbe un "while(...)" */
        /* applica Slow Start */
        Exponential = CW + CW;
        Linear = CW + CS;
        if ((Exponential <= RW) & (Exponential <= Threshold)) /* finchè possibile CW cresce esponenzialmente */
            CW = Exponential;
        else {if (Exponential <= RW) /* altrimenti cresce linearmente */
                CW = Linear;
            else {if (Linear <= RW)
                    CW = Linear;}}
        /* invia pacchetto di dimensione CW; ricevi conferma ed aggiorna RW */
        if (TimeOut) {
            /* se non arrivata ricezione di un pacchetto già inviato, adatta euristicamente i parametri di controllo */
            Threshold = Threshold / 2;
            CW = CS;}
}
```

Figura 3: Schema di principio dell'euristica di controllo di congestione.

X	flusso di dati gestito da un generico utente della rete LAN 1 (da H1 a H5)
Y	flusso di dati gestito da un generico utente della rete LAN 2 (da H6 a H9)
I	flusso di informazioni *Web* per ogni utente relative ad *Intranet* (200 Kbps)
J	flusso di informazioni *Web* per ogni utente relative ad *Internet* (100 Kbps)
ext	flusso di informazioni *Web* generato da utenti esterni per visitare il sito aziendale (ininfluente ai fini del calcolo dei flussi interni)

si ottiene facilmente la distribuzione rappresentata in figura 4.

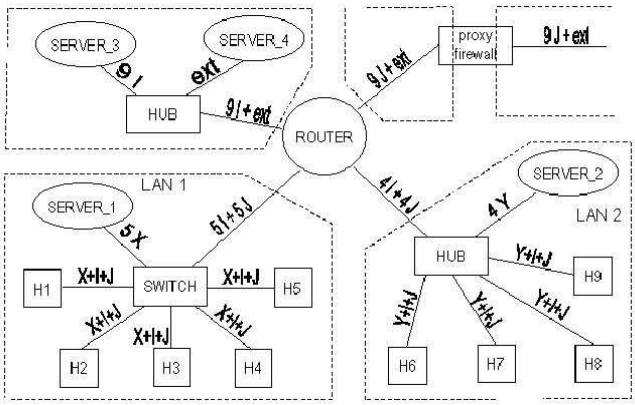

Figura 4: Distribuzione del flussi utili sulla rete.

Passiamo ora ad analizzare le singole reti.

LAN1

La rete LAN1 è gestita da uno *switch* che segmenta totalmente la rete. Poiché lo *switch* opera in modo da garantire per ogni segmento di rete la banda massima, potremmo scrivere le seguenti condizioni, una per ogni ramo dello *switch*:

$$\begin{array}{rcl} X+I+J & \leq & 100 \text{ Mbps} \\ 5X & \leq & 100 \text{ Mbps} \\ 5I+5J & \leq & 100 \text{ Mbps} \end{array}$$

Sostituendo i valori di I e J fissati dal testo otteniamo:

$$\begin{array}{rcl} X & \leq & 100-0,2-0,1=99,7 \text{ Mbps} \\ X & \leq & 100/5=20 \text{ Mbps} \\ 5\times 0,2+5\times 0,1 & \leq & 100 \text{ Mbps (sempre soddisfatta!)} \end{array}$$

Scegliendo la condizione più ristrettiva otteniamo che il valore di X non eccede 20 Mbps.

LAN2

La rete LAN2 è gestita da un *hub* che come sappiamo condivide la banda tra tutti i rami della stessa rete. Abbiamo quindi che in ogni porta dell'*hub* possiamo supporre la presenza di un traffico determinato dalla somma di tutti i traffici utili che fanno capo all'*hub*. Possiamo pertanto esprimere una sola condizione, valida per tutti i rami dell'*hub*:

$$4(Y+I+J)+4Y+(4I+4J)\leq 100 \text{ Mbps}.$$

Raccogliendo e sostituendo i valori di I e J fissati dal testo otteniamo:

$$8Y\leq 100-8I-8J=(100-1,6-0,8)/8=12,2 \text{ Mbps},$$

dal che determiniamo che il valore di Y non eccede $12,2$ Mbps.

Altre LAN

Il traffico nelle altre reti è limitato dagli utenti o da fenomeni esterni, quali il traffico proveniente dall'esterno verso il sito aziendale, e quindi non produce altri vincoli significativi.

Attribuzione indirizzi IP

L'indirizzo acquistato dall'azienda non è influente nella pianificazione degli indirizzi IP della rete aziendale, fatta eccezione per la configurazione della porta P1 del *proxy/firewall*. Per gli altri dispositivi abbiamo a disposizione un'intera classe C di indirizzi riservati 192.168.0.0.

Il dato di progetto da cui partire per l'individuazione del *subnetting* è il numero massimo di *host* per ogni sottorete, che viene fissato in 30 unità utili. Ricordando la suddivisione degli indirizzi e tenendo presente il fatto che in ogni rete o sottorete dobbiamo riservare 2 indirizzi (che quindi non sono attribuibili agli *host*), ricaviamo che il *subnetting* dovrà riservare uno spazio di indirizzamento di 32 unità per il campo *host*, pari a 5 *bit* ($2^5 = 32$). Di conseguenza, il campo *subnet* deve essere definito da 3 *bit* (dato che la rete di partenza è di classe C, e quindi mette a disposizione per le sottoreti un totale di 8 *bit*). Abbiamo quindi la condizione riportata in figura 5.

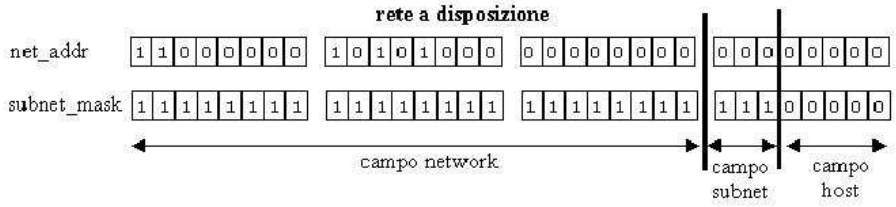

Figura 5: Struttura degli indirizzi utilizzabili dall'azienda.

Il numero di reti enumerabili con questa suddivisione dei campi d'indirizzo (3 *bit* per il campo *subnet*) è pari $2^3-2=6$. Tale quantità è sufficiente a soddisfare le nostre esigenze, che sono limitate all'identificazione di sole 4 sottoreti interne.

La figura 6 mostra le caratteristiche degli indirizzi IP delle prime quattro sottoreti utili:

Di seguito, infine, si riporta invece una possibile attribuzione di indirizzi IP per i dispositivi aziendali interni:

11000000 . 10101000 . 00000000 . 001 00000	192.168.0.32	inutilizzabile (indirizzo della 1ª sottorete)
11000000 . 10101000 . 00000000 . 001 00001	192.168.0.33	1° host utilizzabile nella 1ª sottorete
...	...	...
11000000 . 10101000 . 00000000 . 001 11110	192.168.0.62	ultimo host utilizzabile nella 1ª sottorete
11000000 . 10101000 . 00000000 . 001 11111	192.168.0.63	inutilizzabile (broadcast nella 1ª sottorete)
11000000 . 10101000 . 00000000 . 010 00000	192.168.0.64	inutilizzabile (indirizzo della 2ª sottorete)
11000000 . 10101000 . 00000000 . 010 00001	192.168.0.65	1° host utilizzabile nella 2ª sottorete
...	...	...
11000000 . 10101000 . 00000000 . 010 11110	192.168.0.94	ultimo host utilizzabile nella 2ª sottorete
11000000 . 10101000 . 00000000 . 010 11111	192.168.0.95	inutilizzabile (broadcast nella 2ª sottorete)
11000000 . 10101000 . 00000000 . 011 00000	192.168.0.96	inutilizzabile (indirizzo della 3ª sottorete)
11000000 . 10101000 . 00000000 . 011 00001	192.168.0.97	1° host utilizzabile nella 3ª sottorete
...	...	...
11000000 . 10101000 . 00000000 . 011 11110	192.168.0.126	ultimo host utilizzabile nella 3ª sottorete
11000000 . 10101000 . 00000000 . 011 11111	192.168.0.127	inutilizzabile (broadcast nella 3ª sottorete)
11000000 . 10101000 . 00000000 . 100 00000	192.168.0.128	inutilizzabile (indirizzo della 4ª sottorete)
11000000 . 10101000 . 00000000 . 100 00001	192.168.0.129	1° host utilizzabile nella 4ª sottorete
...	...	...
11000000 . 10101000 . 00000000 . 100 11110	192.168.0.158	ultimo host utilizzabile nella 4ª sottorete
11000000 . 10101000 . 00000000 . 100 11111	192.168.0.159	inutilizzabile (broadcast nella 4ª sottorete)

Figura 6: Ripartizione dello spazio di indirizzamento disponibile per le 4 sottoreti aziendali.

rete	dispositivo	indirizzo IP	*subnet-mask*	
LAN 1	H1	192.168.0.33	255.255.255.224	
	H2	192.168.0.34	255.255.255.224	
	H3	192.168.0.35	255.255.255.224	
	H4	192.168.0.36	255.255.255.224	
	H5	192.168.0.37	255.255.255.224	
	SERVER1	192.168.0.61	255.255.255.224	
	Router E2	192.168.0.62	255.255.255.224	porta *router*-LAN1
LAN 2	H6	192.168.0.65	255.255.255.224	
	H7	192.168.0.66	255.255.255.224	
	H8	192.168.0.67	255.255.255.224	
	H9	192.168.0.68	255.255.255.224	
	SERVER2	192.168.0.93	255.255.255.224	
	Router E3	192.168.0.94	255.255.255.224	porta *router*-LAN2
LAN 3	SERVER3	192.168.0.124	255.255.255.224	
	SERVER4	192.168.0.125	255.255.255.224	
	Router E1	192.168.0.126	255.255.255.224	porta *router*-LAN3
LAN 4	P2	192.168.0.157	255.255.255.224	porta *proxy-router*
	Router E0	192.168.0.158	255.255.255.224	porta *router*-LAN4
LAN ext	P1	122.141.11.241	255.255.255.128	porta *proxy*-ISP