- 1. Spiegare le differenze fra le seguenti modalità di binding degli indirizzi:
     - compile time,
     - load time,
     - execution time.

     Quale delle modalità precedenti necessita di un supporto hardware per poter essere implementata?
  2. Illustrare lo schema di traduzione degli indirizzi nel caso della paginazione.
  3. (*) Supponendo di avere un sistema con tre frame e sette pagine, addottando una politica di rimpiazzamento basata sul working set model, quanti page fault si verificheranno con la reference string seguente, assumendo $\Delta = 3$ e di mantenere in memoria esattamente il solo working set?

     3 4 0 1 0 2 4 5 1 2 6

     (Si assuma che i tre frame siano inizialmente vuoti.)

  Risposta:

  1. L'associazione di istruzioni e dati a indirizzi di memoria può avvenire in vari momenti:
     - compile time: l'associazione avviene a tempo di compilazione e quindi si produce del codice assoluto (la locazione di esecuzione è nota a priori); nel caso in cui si voglia cambiare locazione di esecuzione, bisogna ricorrere ad una ricompilazione del codice;
     - load time: l'associazione avviene a tempo di caricamento; siccome la locazione di esecuzione non è nota a priori, il compilatore produce del codice rilocabile la cui posizione in memoria viene decisa al momento del caricamento e non può essere cambiata durante l'esecuzione del programma;
     - execution time: siccome l'associazione è stabilita a tempo di esecuzione, il programma può essere spostato da una zona all'altra della memoria mentre viene eseguito; questa modalità necessita di un supporto hardware (ad esempio registri base e limite).
  2. Nel caso della paginazione la memoria fisica è suddivisa in blocchi di dimensione prefissata chiamati *frame*, mentre la memoria fisica è suddivisa in blocchi delle stesse dimensioni chiamati *pagine*. Ogni indirizzo generato dalla CPU si suddivide in un numero di pagina p e uno spiazzamento (offset) d all'interno della pagina. Il numero di pagina viene utilizzato come indice nella *page table* del processo correntemente in esecuzione: ogni entry della page table contiene l'indirizzo di base del frame fisico f che contiene la pagina in questione. Tale indirizzo di base f viene combinato (giustapposto come prefisso) con lo spiazzamento d per definire l'indirizzo fisico da inviare all'unità di memoria.
  3. Simuliamo il funzionamento dell'algoritmo basato sul working set model:

3	4	0	1	0	2	4	5	1	2	6
3	4	0	1	0	2	4	5	1	2	6
	3	4	0	1	0	2	4	5	1	2
		3	4		1	0	2	4	5	1
P	P	P	P		P	P	P	P	P	P

Si verificano quindi dieci page fault.

- 1. Illustrare brevemente le seguenti strutture dati utilizzate per la gestione dello spazio libero su disco, evidenziandone pregi e difetti:
   - bitmap o vettore di bit,
   - lista concatenata (linked list).
  2. Descrivere le differenze fra le seguenti modalità di bufferizzazione:
   - bufferizzazione in spazio utente,
   - bufferizzazione in kernel,
   - doppia bufferizzazione.
  3. Spiegare brevemente come funziona il meccanismo di allocazione contigua nel filesystem, evidenziandone vantaggi e svantaggi.

  Risposta:

  1. Nella bitmap ogni singolo bit rappresenta un blocco del disco: se il blocco è allocato ad un file il bit è 0, se il blocco è libero il bit è 1. Il vantaggio di questa codifica è essenzialmente l'efficienza nel trovare il primo blocco libero dato che la maggior parte delle CPU più diffuse mettono a disposizione delle istruzioni macchina che forniscono l'offset del primo bit a 1 in una parola. In questo modo il numero del primo blocco libero può essere calcolato come segue:

     (numero di bit in una parola)×(numero delle parole con tutti i bit 0)+offset del primo bit a 1

     Lo svantaggio è che la bitmap deve essere tenuta in memoria per un utilizzo veloce e quindi può portare ad uno spreco di quest'ultima se il disco è di dimensioni ragguardevoli.
     Una soluzione che non richiede un impiego considerevole di memoria è mantenere una lista concatenata dei blocchi del disco liberi (in cui ogni blocco contiene un puntatore al blocco libero successivo). Così facendo è sufficiente mantenere in memoria il puntatore al primo blocco della lista per essere in grado di reperire un blocco libero all'occorrenza. Lo svantaggio è che se si rende necessario attraversare la lista occorre leggere ogni singolo blocco, degradando le prestazioni del sistema in modo considerevole.
  2. Nel caso della bufferizzazione in spazio utente, il processo riserva un certo numero di locazioni nel suo spazio di indirizzi a beneficio della system call che effettua l'operazione di I/O. Nel caso in cui le pagine contenenti il buffer vengano scaricate su disco nel momento in cui arrivano dei dati dal dispositivo di I/O, questi ultimi potrebbero andare (parzialmente) persi a causa del tempo necessario a ricaricarle in memoria.
     Per ovviare a tale problema si può ricorrere ad un buffer in spazio kernel (che non è soggetto ad essere swappato su disco) ed il cui

contenuto viene ricopiato nel buffer in spazio utente quando è pieno. In questo caso potrebbe nuovamente verificarsi una perdita di dati se, al momento di ricopiare il contenuto del buffer in spazio kernel nel buffer in spazio utente, le pagine contenenti quest'ultimo dovessero essere caricate in memoria e contemporaneamente arrivassero nuovi dati dal dispositivo di I/O (questi non potrebbero essere salvati nel buffer in spazio kernel che risulta pieno).

La soluzione con doppia bufferizzazione permette di risolvere anche l'ultimo problema in quanto, utilizzando due buffer in spazio kernel, nel momento in cui uno dei due risulti pieno, può essere ricopiato in spazio utente (anche con i tempi necessari per ricaricare in memoria le pagine contenenti il buffer del processo) senza il rischio di perdere informazioni, dato che eventuali nuovi dati che dovessero arrivare dal dispositivo di I/O potrebbero essere salvati nel secondo buffer. I due buffer in spazio kernel si scambiano quindi i ruoli di buffer destinato alla memorizzazione di quanto arriva dal dispositivo di I/O e di buffer "di sicurezza" nel caso arrivino nuovi dati durante l'operazione di copia.

3. Se la modalità di allocazione è quella contigua, ogni file occupa un insieme di blocchi contigui (adiacenti) su disco. Quindi l'entry di ogni file nelle directory è sufficiente che contenga l'indirizzo del blocco di partenza e la lunghezza dell'area riservata al file. Come conseguenza si ha che l'accesso al file (sia sequenziale che random) è molto efficiente e richiede al massimo spostamenti della testina di una traccia per volta (quando si deve passare dall'ultimo settore di una traccia al primo della successiva). I problemi si hanno quando bisogna allocare spazio per un nuovo file, dato che l'allocazione contigua soffre del problema della frammentazione esterna. Inoltre può essere difficile allocare dello spazio aggiuntivo per un file dato che fra la zona correntemente riservata al file in questione e quella riservata al prossimo file potrebbe esserci uno spazio insufficiente o nullo. Bisognerebbe quindi ricorrere alla compattazione del file system, che è un'operazione costosa.

- 1. Cosè una socket? Dopo aver dato la definizione si elenchino alcuni tipi di socket.
  2. Cosa distingue un sistema debolmente accoppiato da uno strettamente accoppiato? Si faccia qualche esempio di sistema delle due tipologie.

  Risposta:

  1. Una socket rappresenta un *endpoint* di comunicazione e solitamente è associata ad un indirizzo la cui natura dipende dal dominio di comunicazione scelto (processi che comunicano nello stesso dominio utilizzano lo stesso formato di indirizzi). I domini più comuni sono il dominio Unix (`AF_UNIX`), il dominio Internet (`AF_INET`, `AF_INET6`) il dominio Novell (`AF_IPX`) e il dominio AppleTalk (`AF_APPLETALK`).

     Esistono diversi tipi di socket che rappresentano diverse classi di servizi (non tutti sono disponibili in ogni dominio di comunicazione):

- stream socket: forniscono stream di dati affidabili, duplex, ordinati,
- socket per pacchetti in sequenza: forniscono stream di dati, ma i confini dei pacchetti sono preservati,
- socket a datagrammi: trasferiscono messaggi di dimensione variabile, preservando i confini, ma senza garantire ordine o arrivo dei pacchetti,
- socket per datagrammi affidabili: come quelle a datagrammi, ma l'arrivo è garantito,
- socket raw: permettono di accedere direttamente ai protocolli che supportano gli altri tipi di socket: sono utili per sviluppare nuovi protocolli.

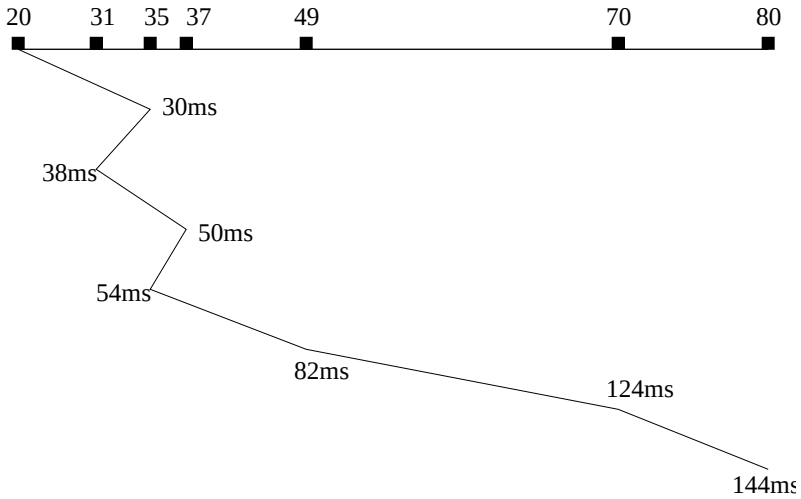
2. I sistemi strettamente accoppiati sono caratterizzati dalla condivisione di clock e/o memoria (es.: sistemi UMA, NUMA) oppure dalla presenza di linee di comunicazione molto veloci fra i vari nodi del sistema che solitamente risiedono nello stesso rack o stanza e che sono amministrati da una singola organizzazione (es.: multicomputer, cluster of workstations). Viceversa i sistemi debolmente accoppiati sono costituiti da sistemi completi (in cui ogni nodo è sostanzialmente una macchina completa) sparsi in regioni geograficamente anche molto distanti fra loro, controllati da diverse organizzazioni e connessi da reti con linee di comunicazione molto più lente (es.: Internet).

- (*) Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 20, ascendente; lo spostamento ad una traccia adiacente richiede 2 ms. Al driver di tale disco arrivano richieste per i cilindri 70, 31, 49, 35, 80, rispettivamente agli istanti 0 ms, 30ms, 40 ms 50 ms, 70 ms. Si trascuri il tempo di latenza.

  1. In quale ordine vengono servite le richieste?
  2. Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene effettivamente servita. Qual è il tempo di attesa medio per le quattro richieste in oggetto?

  Risposta:

  1. L'ordine in cui vengono servite le richieste è 31, 35, 49, 70, 80 come illustrato dal seguente diagramma:



2. Il tempo d'attesa medio è $\frac{(38-30)+(54-50)+(82-40)+(124-0)+(144-70)}{5} = \frac{8+4+42+124+74}{5} = 252/5 = 50,4\ ms$.

Il punteggio attribuito ai quesiti è il seguente: 3, 3, 5, 3, 3, 3, 3, 3, 6 (totale: 32).