# Appunti di Sistemi Operativi

---

**Enzo Mumolo**

e-mail address :mumolo@units.it
web address :www.units.it/mumolo

# Indice

# Capitolo 1

# Cenni su alcuni algoritmi del Kernel di Unix

## 1.1 Elementi di Unix Internals

Il kernel gestisce i file memorizzati sulla memoria di massa; quando un processo deve accedere ai dati contenuti in un file, il kernel porta i dati in memoria centrale. Consideriamo ad esempio il processo

```
cp file1 file2
```

che copia il primo nel secondo file. I dati del primo file vengono letti dal disco e messi in memoria, quindi vengono scritti nel secondo file. L'operazione di lettura dei dati in memoria centrale consente di minimizzare i tempi di accesso ai dato stessi, visto che l'accesso diretto al disco richiede un tempo molto alto. La struttura di bufferizzazione é chiamata **Buffer Cache** per quanto riguarda i blocchi di dati e **I-Node Cache** per quanto riguarda i descrittori di file. L'operazione di bufferizzazione si rifá al seguente concetto: quando il kernel legge dati da disco, il kernel cerca come prima cosa di leggere i dati dalla cache. Se i dati sono giá contenuti nel buffer cache, non deve leggerli dal disco. Se i dati non sono nella cache, il kernel li legge dal disco e li mette in cache.

### 1.1.1 Il Buffer Cache

I blocchi dati che vengono letti o scritti da o su disco risiedono piú a lungo possibile in memoria centrale. La struttura nella quale i dati sono organizzati durante la loro residenza in memoria é il Buffer Cache. Tale struttura é visualizzata in fig.1.1 . Il blocco di dati numero $N$ ($N$ fá riferimemnto al numero di settore di disco) é contenuto nel Buffer numero $N$, e tutti i buffer allocati in memoria centrale sono posti in code bidirezionali. Le code sono gestite mediante un indirizzamento Hash, basato cioé su operazioni di modulo. La disposizione della buffer cache é visibile in fig.1.1 , dove si vede che in ogni coda sono presenti i buffer con numero che fornisce un resto della divisione per 4 é pari a zero, uno e cosí via. Naturalmente il numero 4 deriva dal fatto che per semplicitá si sono indicate 4 code nella struttura.

L'algoritmo per l'allocazione in memoria di un buffer contenente un blocco dati é riportato qui di seguito.

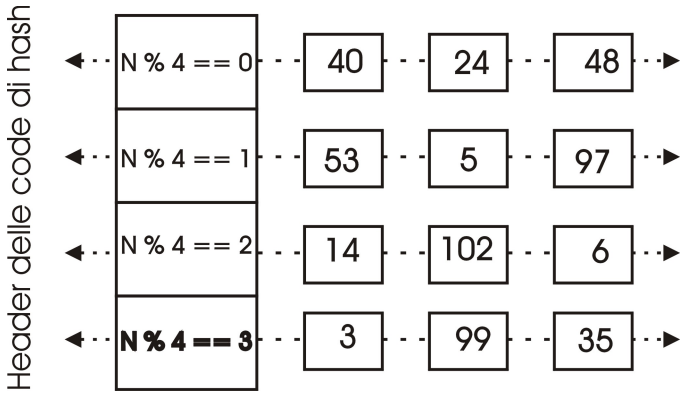


**Figura 1.1** Struttura della Buffer Cache

```
algoritmo getblk input: numero del file system e numero del blocco
dati output: buffer allocato in memoria {
    while (buffer non trovato) {
        if (blocco in coda di hash) {
            if (buffer occupato) {
                sleep(fino a quando il blocco si libera);
                continue;
            }
            stato buffer=occupato;
            rimuove il buffer dalla lista libera;
            return(buffer);
        }
        else
        {
            if (non ci sono più blocchi sulla lista libera)
            {
                sleep(fino a quando qualche blocco si libera);
                continue;
            }
            rimuove il buffer dalla lista libera;
            if (stato del buffer = scrittura ritardata) {
                scrittura asincrone del buffer sul disco;
                continue;
            }
            rimuove il buffer dalla vecchia posizione in coda di hash;
            posiziona il buffer nella nuove posizione;
            return(buffer);
            }
        }
```

I buffer usati precedentemente contengono essenzialmente i dati del corrispondente blocco dati su disco, ma non solo. Evidentemente per realizzare la strutture della Buffer Cache sono necessari anche dei puntatori agli altri buffer in coda di Hash. Inoltre esiste una Lista Libera che fá riferimento ai buffer riutilizzabili per allocare altri dati o, se hanno il numero di blocco richiesto e sono giá risiedenti in memoria centrale, utilizzabili con gli stessi dati (che sono giá quelli desiderati). La

struttura del buffer é riportata in fig.1.2

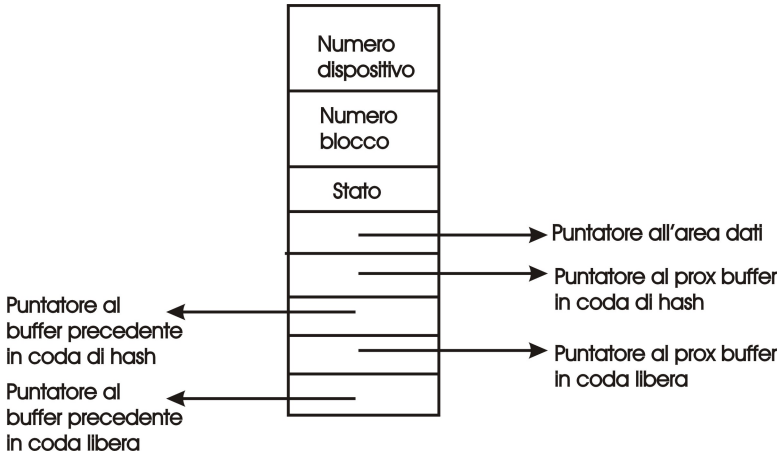


**Figura 1.2** Header della buffer cache

e le code del buffer sono strutturate come illustrato in fig.2.3.

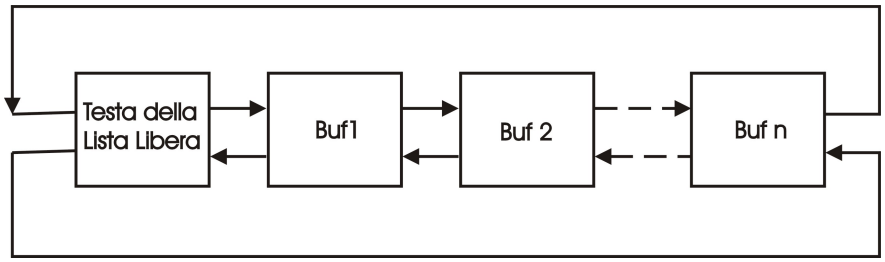


**Figura 1.3** una coda di hash

In figura 2.4 si mostra che il blocco A cercato da un processo si trova nel buffer di uguale numero nella Buffer Cache sulla Lista Libera; il buffer viene estratto dalla Lista Libera come mostrato in fig. 2.5

In virtú della cache, la lettura di un blocco di dati é velocizzata, visto che se il blocco risiede giá in memoria centrale non seve leggerlo da disco. Lo pseudocodice dell'algoritmo di lettura chiamato bread, é riportato di seguito.

```
bread(nr.blocco dati) {
    getblk(nr blocco);
    if(buffer valido)
      return(buffer);  //il buffer contiene gia' i dati giusti
    else{
      inizia a leggere il blocco da disco; //lettura sincrona
      sleep(fine lettura);
      return(buffer);
}
```

Quando un buffer non é piú necessario viene rilasciato, che vuol dire che viene posto nella Lista Libera. L'algoritmo per il rilascio dei buffer di dati é brelse, il cui pseudocodice é riportato.

```
brelse(nr.buffer) {
    Sveglia tutti i processi in attesa di un buffer;
```

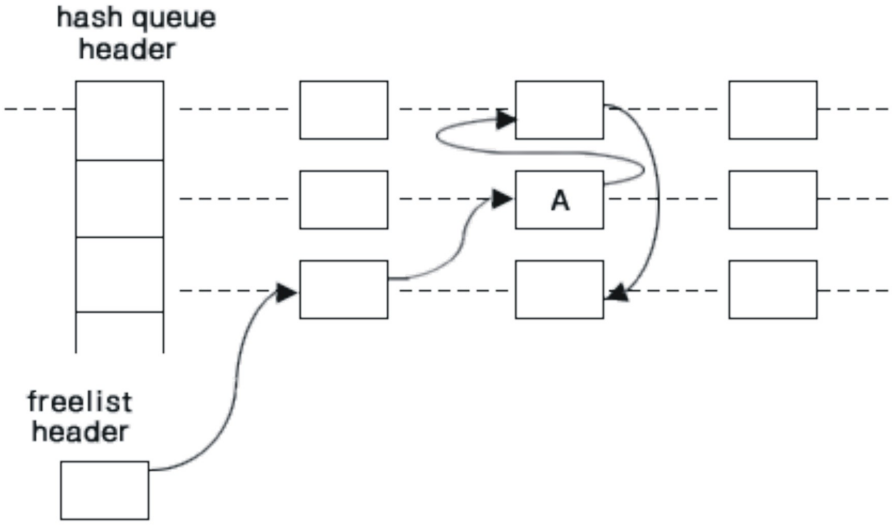


**Figura 1.4** Ricerca del blocco A sulla Buffer Cache

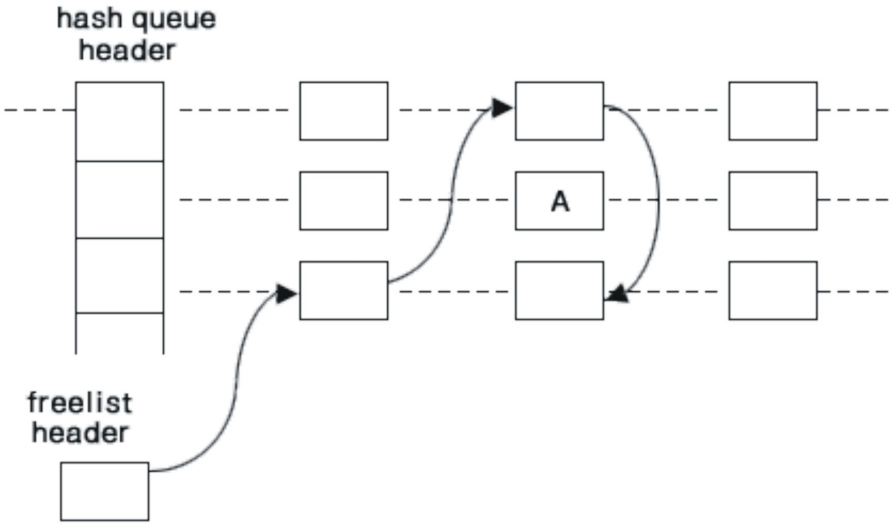


**Figura 1.5** Il buffer contenente i dati del blocco A viene estratto dalla Lista Libera

```
    if(buffer valido e buffer non vecchio)
      accoda il buffer alla fine della Lista Libera;
    else
      accoda il buffer in cima della Lista Libera;
    sblocca il buffer;
}
```

L'algoritmo di scrittura, bwrite, scrive solo che il blocco sará scritto in seguito dalla getblk, settando lo stato a scrittura ritardata,

```
bwrite(nr.buffer) {
    if(scrittura sincrona){
      scrivi il buffer su disco;
      sleep(fine scrittura);
      brelse(nr.buffer);
```

```
    }
    else
      marca il buffer a scrittura ritardata, non valido e vecchio;
}
```

### 1.1.2 L'Inode Cache

I blocchi del disco di Unix sono disposti come mostrato nella figura 2.6.

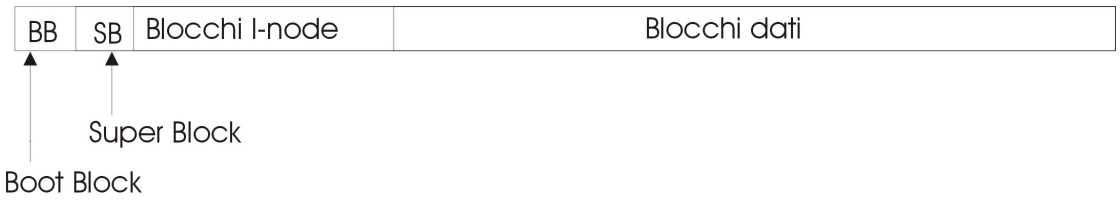


**Figura 1.6** Configurazione di una partizione logica del disco Unix

Il blocco di boot contiene il codice che serve per caricare il Sistema Operativo. Il Super Blocco (SB) contiene informazioni sull'intero disco. La lista degli Inode éla lista dei descrittori dei file Unix e la lista dei blocchi dati contiene i dati nella forma di directory e file.

In particolare, il SB contiene le seguenti informazioni relative all'intero file system:

- dimensione del file system
- numero di blocchi liberi
- lista dei blocchi liberi
- numero del prossimo blocco libero della lista
- dimensione della lista degli inode
- numero di inode liberi
- lista degli inode liberi
- numero del prossimo blocco libero
- dimensione della lista degli inode
- opzione di blocco delle liste dei blocchi liberi e degli inode liberi
- flag che indica modifiche del super blocco
- numero totale di blocchi

Il SB contiene un vettore di numeri di blocchi liberi, l'ultimo dei quali punta alla prossima lista. In altri termini, l'ultimo numero rappresenta un blocco di dati che contiene la lista dei prossimi blocchi liberi e il puntatore alla prossima lista. Quando un processo richiede un blocco, viene esaminata la lista dei blocchi liberi per recuperare un numero di blocco libero. Se il SB contiene solo un elemento, tutto il contenuto di quel blocco viene copiato nella lista dei blocchi liberi del SB e viene ritornato quel numero di blocco. Il rilascio di un blocco dati él'operazione inversa. In altri termini, se la lista dei blocchi liberi nel SB ha abbstanza spazio, il numero del blocco che si vuole rilasciare viene aggiunto alla lista. Se la lista non ha spazio, tutti gli elementi della lista dei blocchi

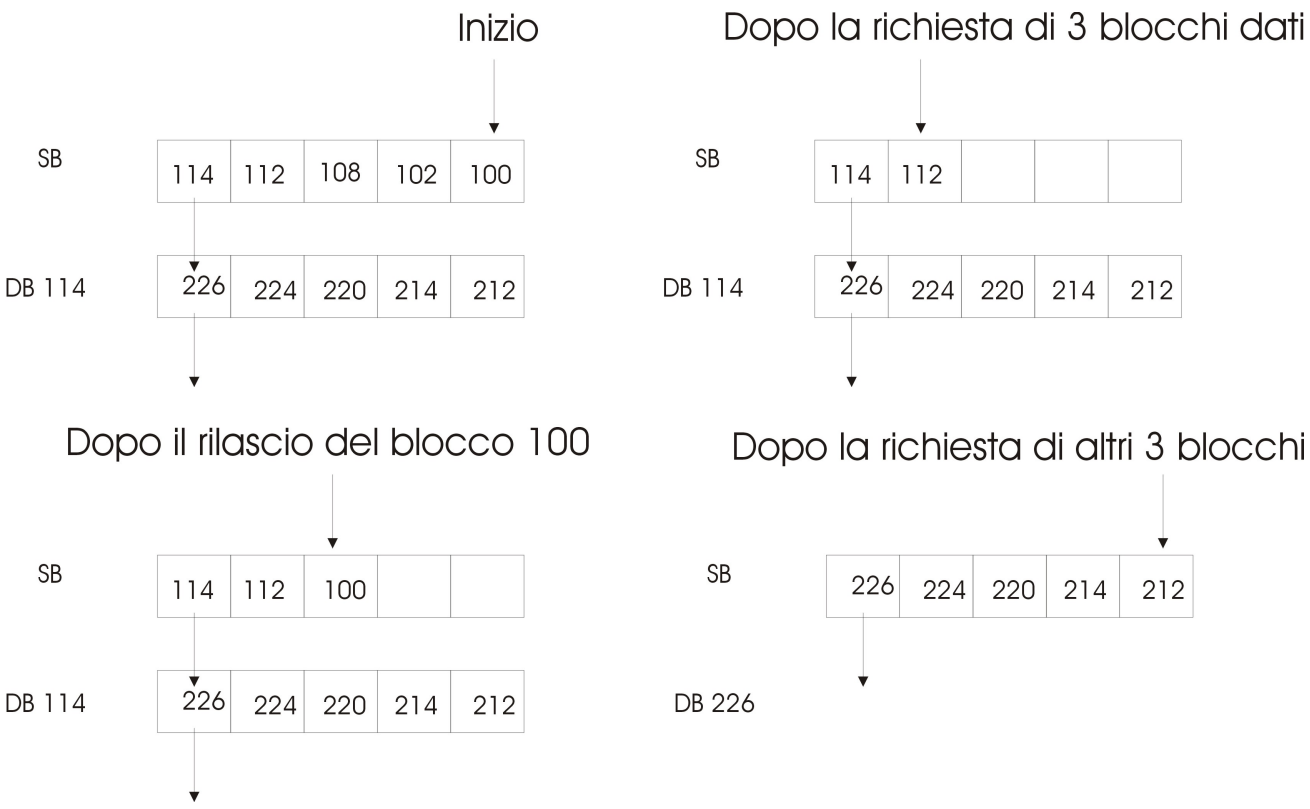


**Figura 1.7** una serie di richieste di blocchi dati

liberi vengono copiati nel blocco che si vuole rilasciare e il numero di quel blocco viene inserito nella lista del SB, che quindi contiene solo quel numero.

La figura 1.7 mostra una serie di richieste e rilasci di blocchi di dati.

Quando il blocco richiesto é l'ultimo della lista, il suo contenuto viene letto e utilizzato per riempire la lista nuovamente, dato che i blocchi liberi sono strutturati in Unix secondo una modalitá a raggruppamento.

Viceversa, quando si rilascia un blocco, si ripete l'operazione descritta, nel senso che la lista - se piena - viene scritta nel blocco che diventa l'unico della lista, come si vede nella figura 1.8.

Gli algoritmi di allocazione e rilascio blocchi sono descritti in forma di pseudocodice e sono chiamati rispettivamente alloc e free.

```
alloc() {
    while(SB occupato)
        sleep(SB libero);
    preleva il prossimo numero di blocco dati libero;
    if(ultimo numero della lista)
    {
       occupa SB;
       bread(blocco appena tolto dal SB);
       copia numeri letti nella lista  del SB dei blocchi dati liberi;
       brelse(blocco appena tolto dal SB);
       libera SB;
       sveglia i processi che aspettano che si liberi il SB;
     }
     getblk(numero prelevato dal SB);
```

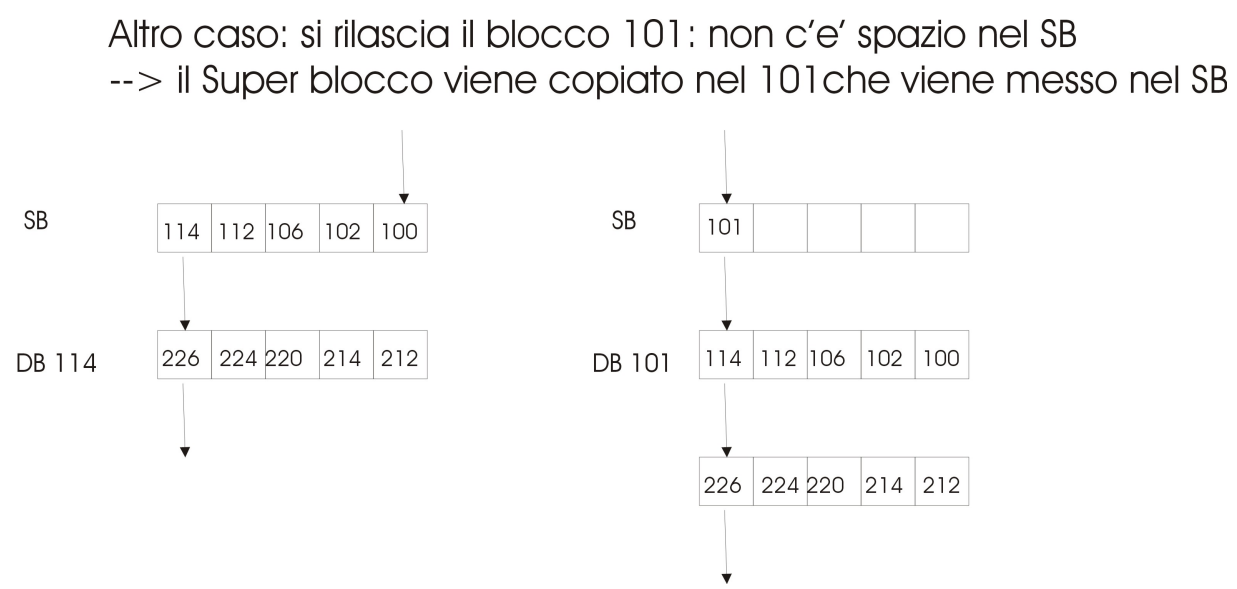


**Figura 1.8** rilascio di blocchi dati.

```
     azzera il buffer;
     return(buffer);
}

free(nr.blocco)   //blocco da liberare {
    while(SB occupato)
        sleep(SB libero);
    if(lista nel SB dei blocchi liberi piena)
    {
       occupa SB;
       copia la lista nel SB dei blocchi liberi nel buffer del blocco da liberare;
       bwrite(nr.buffer); //scrive la lista sul disco
       metti nella lista del SB il nr blocco da liberare; //e' l'unico della lista!
     }
     else
        inserisci il numero del blocco nella lista;
 }
```

Il SB contiene anche la lista degli inode liberi. Durante l'assegnazione di un Inode ad un nuovo file, il kernel analizza la lista degli Inode liberi. Se si trova un Inode libero, viene ritornato quel numero. Se la lista evuota, il kernel cerca gli Inode liberio (filetype=0) tra quelli presenti su disco. Il kernel allora riempie la lista degli inode liberi piuche puó, e ritorna uno di questi numeri. Il numero piú alto viene memorizzato. La prossima volta che vengono esaminati gli Inode su disco per cercare quelli liberi, non é necessario ripartire da zero ma dal numero memorizzato, per non rianalizzare Inode giaá analizzati, allo scopo evidentemente di migliorare l'efficienza.

Quando viene rilasciato un Inode, viene messo nella lista libera in SB se c'é spazio, altrimenti, se il numero é minore del numero memorizzato viene aggiornato il numero dell'Inode memorizzato con il numero dell'Inode che si vuole rilasciare. Se il numero dell'Inode che si vuole rilasciare é maggiore dell'Inode memorizzato, non é necessario aggiornare il numero memorizzato perché quell'Inode verrá trovato alla prossima ricerca.

Un blocco di Inode é un descrittore di file, e contiene le seguenti informazioni:

- Informazioni sul possesso del file (Proprietario, gruppo )
- Tipo del file
- Permessi d'accesso (nel formato rwx rwx rwx)
- Istante dell'ultima modifica
- Istante dell'ultimo accesso
- Numero di link al file
- Dimensione del file
- I blocchi di cui é composto il file

I blocchi di cui é composto il file sono indicati in una lista contenuta nell'Inode e, se la dimensione del file eccede il numero massimo di blocchi che possono essere messi in un Inode, si usa un indirizzamento multiplo, come illustrato in fig.1.9.

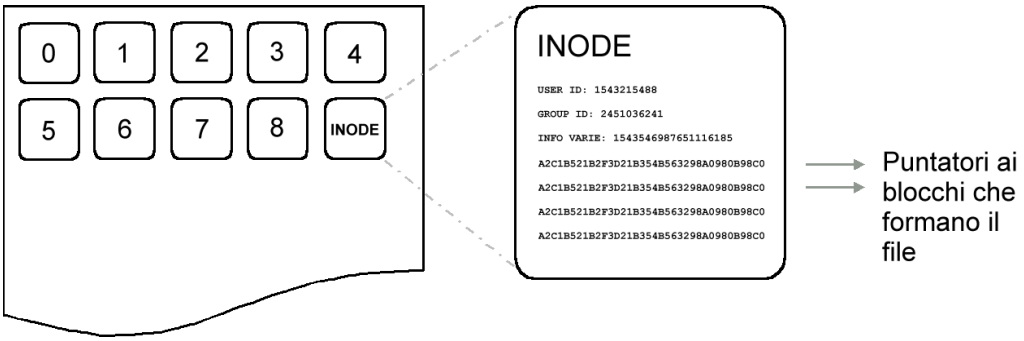


**Figura 1.9** inode

L'Inode é dunque un blocco di disco. Analogamente ai blocchi dati (struttura Buffer Cache) gli Inode sono memorizzati in memoria in strutture simili a quelle della Buffer Cache: si parla di Inode Cache. Oltre ai campi visti per gli Inode su disco, un Inode in memoria contiene anche i seguenti campi:

- Stato dell'Inode: Inode occupato, un processo sta aspettando che l'Inode si liberi, la copia in memoria é diversa dall'Inode su disco
- identificatore logico del disco su cui sta l'Inode
- Il numero di Inode (l'Inode su disco non ha bisogno di questo campo)
- Puntatori alla coda degli Inode e alla Lista Libera di Inode liberi
- contatore di riferimenti all'Inode (quanti processi stanno condividendo il file)

Gli algoritmi per assicurare un buffer contenente un Inode in Cache sono iget (se si conosce il numero di Inode) e ialloc (se non si conosce il numero di Inode perché viene creato un nuovo file, mentre iput é l'algoritmo che rimette i buffer degli Inode in memoria sulla lista libera e ifree é l'algoritmo che rilascia il blocco di Inode su disco, cioé rende nuovamente utilizzabile un numero di Inode.

```
iget(nr.Inode su disco) {
    while(l'Inode su disco non si trova nella Inode Cache){
      if(Inode nella Inode Cache)
          {
              if(Inode bloccato)
              {
                 sleep(Inode libero);
                 continue;
              }
      // qui l'Inode e' libero ma non e' detto che si trovi nella Lista Libera!
      // Gli Inode possono essere condivisi da piu' processi
              if(Inode nella Lista Libera)
                  togli Inode dalla Lista Libera;
              incrementa il numero di riferimenti;
              return(Inode in memoria)
       }
       else
          {
              if(Lista Libera vuota)
                  return(errore)
              togli ll buffer dalla Lista Libera;
              azzera numero di Inode e di File System;
              sposta il buffer dalla vecchia coda di hash alla nuova;
              bread(nr. Inode); //leggi Inode da disco e mettilo nel buffer in Inode Cache
              nr.riferimenti=1;
              return(Inode in memoria)
           }
    }
}

ialloc() {
    while(non allocato nuovo Inode in Inode Cache){
    {
       if(SB occupato)
       {
          sleep(SB libero);
          continue;
       }
       if(lista degli Inode liberi nel SB vuota)
       {
          occupa SB;
          recupera Inode memorizzato;
          cerca su disco gli Inode liberi partendo dall'Inode memorizzato;
          //legge gli Inode con bread e brelse e vede se il tipo di file e' zero
          libera SB;
          Inode memorizzato=massimo dei numeri di Inode liberi letti da disco;
        }
        leggi il prossimo numero di Inode libero dal SB;
        iget(nr.Inode);
        inizializza Inode;
```

```
        scrive Inode su disco;
        return(Inode);
}
```

In figura 1.10 si riporta una sequenza di richieste e rilasci di Inode.

```
iput(nr.buffer) {
    blocca Inode in memoria;
    nr.riferimenti--;
    if(nr.riferimenti == 0)
    {
       if(nr.link == 0){
          free; //libera i blocchi di disco del file
          tipo di file=0;
          ifree(nr.buffer); //libera Inode
        }
        if(file o Inode cambiati)
           aggiorna Inode su disco;
        metti l'Inode in memoria (buffer) nella Lista Libera;
     }
     sblocca Inode;
}

ifree(nr.Inode da liberare) {
    if(SB occupato)
        return;
    if(lista Inode piena)
    {
       if(nr.di Inode < Inode memorizzato)
          aggiorna l'Inode memorizzato;
     }
     else
        memorizza il numero Inode nella lista degli Inode liberi in SB;
}
```

Durante queste allocazioni, c'é sempre la possibilitá che gli Inode siano modificati se sono condivisi. Durante l'accesso ad u Inode da parte di una system call, l'Inode stesso viene quindi bloccato.