Sistemi Operativi (Laboratorio)

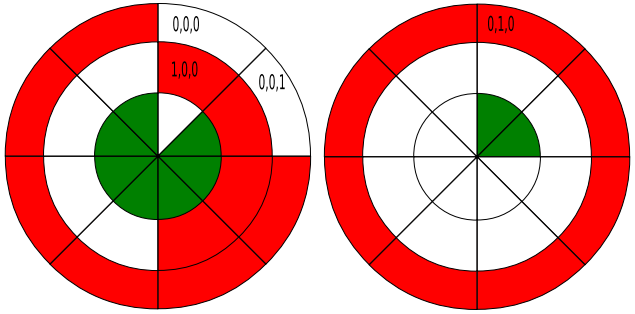
Lorenzo Martignoni

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano, Italia
lorenzo@security.dico.unimi.it

a.a. 2008/09

Lezione IX: Memoria di massa e system call

CHS



- $C = 3$ $H = 2$ $S = 8$, totale blocchi 48
- Zona (partizione) rossa 0,0,2 $\rightsquigarrow$ 1,0,3

$(1*(2*8)+0*8+3*1)-(0*(2*8)+0*8+2*1) = 19-2 = 17$

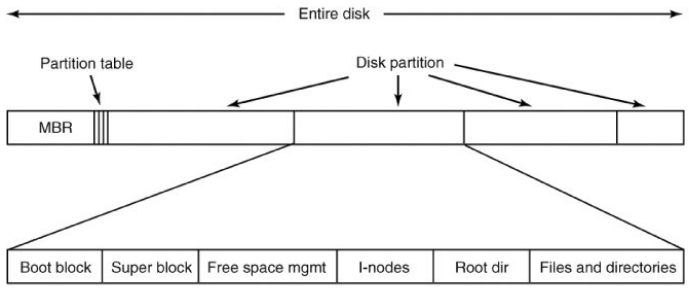
(in realtà 18 perché contiamo da zero)

Partizioni

Lo spazio di memoria di uno hard-disk è ripartito in porzioni indipendenti (partizioni): in linea di principio possono contenere anche sistemi differenti. Generalmente contengono sotto-file-system il cui backup e/o aggiornamento è indipendente.

- Partition table sector: contiene la descrizione di 4 partizioni (primarie) agli offset 446, 462, 478, 494
- Partizione: una zona contigua del disco (CHS)
- Partizione estesa: una partizione che permette una nuova suddivisione (partizioni logiche) grazie ad un nuovo PTS

Disk layout



Creare e usare un fs

- Un file system va creato (`mkfs`)
- Un file system va montato (`mount`)
- Corrispondentemente va smontato (`umount`)
- Ogni file è caratterizzato da un i-node e conosciuto tramite uno o piú link o nomi (`ln`)

Esercizio (esame febbraio 2009)

1. Creare un disco virtuale (`qemu-img`) e connetterlo alla macchina virtuale (come `hdb`)
2. Partizionare il nuovo disco (`part`):

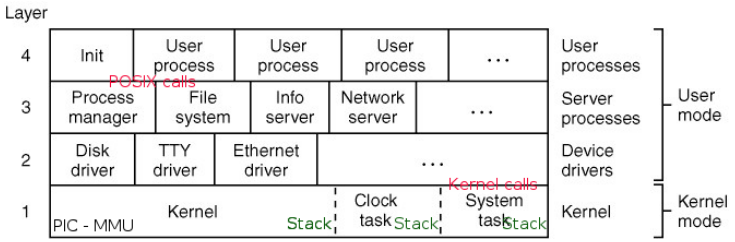
#	Tipo	Dimensione
1	MINIX	10MB
2	WINDOWS (FAT32)	20MB
3	SWAP	30MB
4	WINDOWS (FAT32)	10MB
5	MINIX	30MB

3. Creare un file system MINIX sulla partizione 5
4. Montare la partizione 5 in /tmp/esame

Attenzione

Per creare più di 4 partizioni è necessario creare delle partizioni con sotto-partizioni. Lasciare lo spazio per il boot sector.

MINIX



Come aggiungere una system call `foo()`

Supponendo che il servizio sia gestito gestito da PM

1. Aggiornare `/usr/src/include/minix/callnr.h`
2. Aggiungere la entry `do_foo` al `call_vec` di PM
3. Aggiungere la entry `no_sys` al `call_vec` di FS
4. Aggiungere `do_foo()` (p.es. in `pm/misc.c`)
5. Aggiungere il prototipo di `do_foo()` (in `pm/proto.h`)
6. Creare la libreria wrapper e aggiornare `unistd.h`

`http://www.cis.syr.edu/~wedu/seed/Documentation/Minix3/How_to_add_system_call.pdf`

Esercizio

- Aggiungere una syscall che stampa "ciao"
- Aggiungere una syscall che modifica il proprietario di un processo in esecuzione (trascurare la task call a FS)
- Aggiungere una syscall che stampa il numero di processi nella coda ready (richiesta una kernel call)

Un semplice semaforo

Primitive da implementare:

1. sem_init
2. sem_status
3. sem_up
4. sem_down

Aggiungiamo queste primitive (system call) ad un server esistente (FS).

Un semplice semaforo

```
int do_down(message *m_ptr) {
        /* Resource available. Decrement semaphore and reply. */
        if (s > 0) {
                s = s  1; /* take a resource */
                return(OK); /* let the caller continue */
        }
        /* Resource taken. Enqueue and block the caller. */
        enqueue(m_ptr->m_source); /* add process to queue */
        return(EDONTREPLY); /* do not reply in order to block
the caller */
}
int do_up(message *m_ptr) {
    message m; /* place to construct reply message */
        /* Add resource, and return OK to let caller continue. */
        s = s + 1; /* add a resource */
        /* Check if there are processes blocked on the semaphore. */
        if (queue_size() > 0) { /* are any processes blocked? */
                m.m_type = OK;
                m.m_source = dequeue(); /* remove process from queue */
                s = s  1; /* process takes a resource */
                send(m.m_source, m); /* reply to unblock the process */
        }
        return(OK); /* let the caller continue */
    }
```

Semaphore Server

Creare un nuovo server che implementa le primitive per la gestione dei semafori

Semaphore Server

Creare un nuovo server che implementa le primitive per la gestione dei semafori

Semaphore Server

Aggiungere un server SS che gestisca un semaforo (s) secondo lo schema indicato.

- Partire da un server esistente (p.es. DS o IS)
- Aggiornare con i messaggi possibili `include/minix/com.h`
- Aggiungere il server alla boot image (è piú semplice che caricarlo poi)
- Aggiornare i makefile

Semaphore Server

```
void semaphore_server( ) {
    message m;
    int result;
    /* Initialize the semaphore server. */
    initialize( );
    /* Main loop of server. Get work and process it. */
    while(TRUE) {
        /* Block and wait until a request message arrives. */
        get_work(&m);
        /* Caller is now blocked. Dispatch based on message type. */
        switch(m.m_type) {
            case UP: result = do_up(&m); break;
            case DOWN: result = do_down(&m); break;
            default: result = EINVAL;
        }
        /* Send the reply, unless the caller must be blocked. */
        if (result != EDONTREPLY) {
            m.m_type = result;
            send(m.m_source, &m);
        }
    }
}
```