Micro Filesystem (ufs): specifiche per il progetto d'esame (include esonero n.2)

Laboratorio di Sistemi Operativi I
Anno Accademico 2007-2008

Francesco Pedullà
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Il progetto - Obiettivo

- Questo modulo descrive le specifiche del progetto da svolgere per l'esame di Laboratorio di Sistemi Operativi I - AA 2007-2008.
- Tali specifiche si applicano anche al **compito di esonero n.2**, il quale pero' richiede l'implementazione di un numero inferiore di funzionalita'.
- **NOTA BENE**: il progetto deve costituire ***una creazione originale***, quindi non è possibile condividere parti di codice o della eventuale relazione, se richiesta, con altri studenti/gruppi, o copiare contenuti derivanti da altre fonti.

Il progetto richiede lo sviluppo di un filesystem minimale, denominato `ufs` (micro filesystem)

Il progetto – Specifiche generali

Il micro filesystem ufs:

- dovra' emulare il comportamento del filesystem Linux, fornendo all'utente un insieme completo, seppur limitato, di funzionalita'
- dovra' essere implementato come un'applicazione nello user space, cioe' NON integrata nel kernel, ma basata su eseguibili e librerie
- dovra' organizzare e memorizzare le informazioni non volatili (quelle residenti su disco, cioe' i dati dei file con gli opportuni metadati e quelle relative alla struttura del filesystem stesso) all'interno di un unico file regolare Linux (dev_ufs)
- dovra' fornire un'API (libufs.a) simile alle system call Linux per file I/O
- dovra' fornire una shell interattiva (ufsh) per il test dell'applicazione
- dovra' fornire il servizio mediante istanze dedicate di un demone (ufsd.[0-N]), che fungono da processi responsabili della gestione diretta dello spazio dati del filesystem

Caratteristiche base del micro filesystem ufs

- la struttura dati su disco del filesystem da realizzare deve essere basata sul FAT32 semplificato (si veda allegato per una descrizione dettagliata)
- gestisce file (regolari) e directory
- supporta l'accesso ai file/directory soltanto mediante pathname assoluto
- gestisce **opzionalmente** (per il progetto) l'accesso in read/write ai file in relazione al proprietario (owner) e agli altri (others)
- lo spazio disponibile e la dimensione dell'unita' minima allocabile per i dati (*file block o cluster*) e' definito staticamente al momento della creazione del filesystem mediante ufs_mke2fs (si veda *Comandi base per sysadmin*)

Considerazioni e precisazioni

- per le applicazioni utente l'accesso al filesystem dev_ufs deve avvenire mediante un sistema di server dedicati
- ad ogni applicazione deve essere associata un'istanza (identificata come ufsd.i) dedicata del server con relativa coppia di FIFO R/W per la comunicazione full-duplex
- la creazione del server ufsd.i deve avvenire durante l'esecuzione della system call ufs_connect()
- i server ufsd.i devono gestire in modo opportuno (eventualmente in mutua esclusione mediante lock di parti del file dev_ufs) le system call che, modificando i dati all'interno di dev_ufs, possono generare problemi di incoerenza delle informazioni, nel caso in cui vengano chiamate concorrentemente da piu' applicazioni utente o siano interrotte in modo asincrono dalla ricezione di segnali
- la terminazione del server ufsd.i deve avvenire a seguito dell'invocazione della system call ufs_disconnect()

Differenze tra le specifiche applicabili all'Esonero n.2 e quelle del progetto (appello normale)

- **Esonero n.2:**
  tutte le system call base (pagg. 6-8) e i comandi base (pag. 9)

- **Progetto:**
  in aggiunta alle system call base e ai comandi base, implementazione di UNA funzionalità opzionale tra quelle proposte (pag. 12)

- La scadenza per la consegna dell'Esonero n.2 è di circa 10 giorni in anticipo rispetto a quella del progetto per il primo appello dell'anno (verificare le scadenze sul sito del corso)

System call base da implementare - I

Nota:

- tutte i tipi NON definiti dall'ANSI C (typedef tipo size_t, off_t, ...) devono essere identici a quelli definiti per le corrispondenti system call Linux
- tutti i pathname devono iniziare per / (pathname assoluto).

Operazioni base sui file

- int ufs_creat(const char *pathname, mode_t mode);
  – crea un file. mode può essere ignorato per le funzionalità base, ma va gestito se si implementa il controllo dei permessi di accesso (r/w).
- int ufs_remove(const char *pathname);
  – cancella un file.
- int ufs_rename(const char *oldpath, const char *newpath);
  – cambia il nome di un file/directory.
- int ufs_open(const char *pathname, int flags);
  – apre un file. flags può essere O_RDONLY, O_WRONLY, O_RDWR. Il supporto di O_APPEND e O_TRUNC è un'estensione opzionale.
- int ufs_close(int fd);
  – chiude un file

System call base da implementare - II

Operazioni base sui file

- ssize_t ufs_read(int fd, void *buf, size_t count);
  – legge un determinato numero di byte da un file
- ssize_t ufs_write(int fd, const void *buf, size_t count);
  – scrive un determinato numero di byte in un file
- off_t ufs_lseek(int fildes, off_t offset, int whence);
  – sposta il puntatore di lettura/scrittura del file. whence ha gli stessi valori della system call Linux corrispondente.
- int ufs_stat(const char *pathname, struct ufs_stat *buf);
  – legge gli attributi del file. Il tipo struct ufs_stat deve essere definito come:

```
struct ufs_stat {
        short mode; /* attributi del file/directory */
        short uid;  /* uid del file owner */
        int   ctime;/* data e ora di creazione */
        int   mtime;/* data e ora dell'ultima modifica */
        int   size; /* dimensione del file in byte */
};
```

System call base da implementare - III

Operazioni base sulle directory

- int ufs_mkdir(const char *pathname, mode_t mode);
  – crea una directory. mode può essere ignorato per le funzionalità base, ma va gestito se si implementa il controllo dei permessi di accesso (r/w).
- int ufs_rmdir(const char *pathname);
  – cancella una directory, ma solo se vuota.
- ufs_DIR *ufs_opendir(const char *pathname);
  – apre una directory.
  Il tipo ufs_DIR deve essere definito come:
  typedef int ufs_DIR; /* opaque numeric index of an internal data structure */
  Il valore effettivo di quanto ritornato da opendir() e' il puntatore ad un indice numerico, simile al valore ritornato dalla syscall ufs_open(), che rappresenta una chiave univoca di accesso alla entry appropriata della tabella process-wide (cioe' nel server dedicato ufsd.i) delle directory aperte
- int ufs_closedir(ufs_DIR *dir);
  – chiude una directory.

System call base da implementare - IV

Operazioni base sulle directory

- struct ufs_dirent *ufs_readdir(ufs_DIR *dir);
  – legge una directory, ritornando la prossima directory entry. Ls struttura ufs_dirent deve essere definita come:

```
/* entry retrieved by ufs_readdir(), exactly mapped onto
   the directory entry defined in dev_ufs */
struct ufs_dirent {
        char  name[12]; /* file/subdir name *;
        short type;     /* file/subdir attributes */
        short uid;      /* user ID of file/subdir owner */
        int   ctime;    /* creation date */
        int   mtime;    /* modification date */
        int   clini;    /* first assigned cluster index */
        int   size;     /* file/subdir size in bytes */
};
```

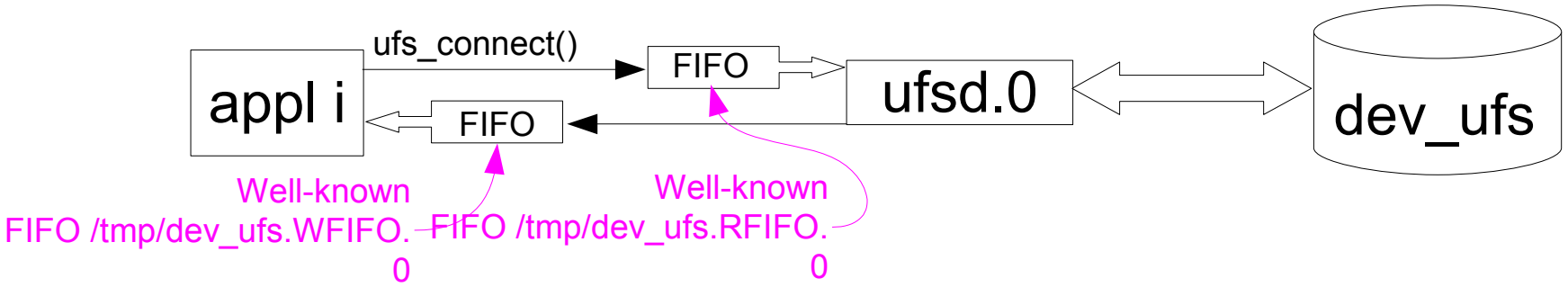
System call base da implementare - IV

Operazioni per l'accesso al filesystem

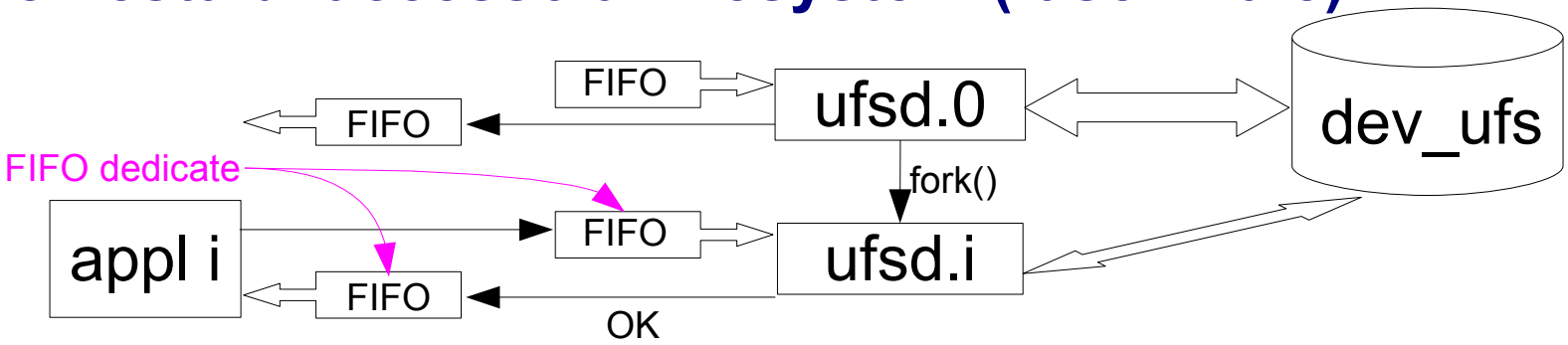
- int ufs_connect(void);
  – predispone l'applicazione utente per l'accesso al filesystem dev_ufs, inviando la richiesta al server ufsd.0 sulla well-known FIFO /tmp/dev_ufs.RFIFO.0 e ottenendo 2 canali di comunicazione dedicati /tmp/dev_ufs.RFIFO.i, tmp/dev_ufs.WFIFO.i con i>0 (oppure /tmp/dev_ufs.RFIFO.PID, tmp/dev_ufs.WFIFO.PID con PID=PID dell'applicazione) rispettivamente per l'invio delle richieste e ricezione delle risposte al/dal server dedicato ufsd.i (generato da ufsd.0). Questa chiamata deve precedere qualsiasi altra chiamata alle altre system call.
- int ufs_disconnect(void);
  – rilascia l'accesso al filesystem dev_ufs, terminando la comunicazione col server dedicato ufsd.i (ufsd.i deve essere terminato in modo opportuno e le FIFO /tmp/dev_ufs.RFIFO.i e tmp/dev_ufs.WFIFO.i devono essere rimosse). Dopo questa chiamata non possono piu' essere utilizzate le altre system call.

server ufsd: procedura di accesso (schema indicativo)

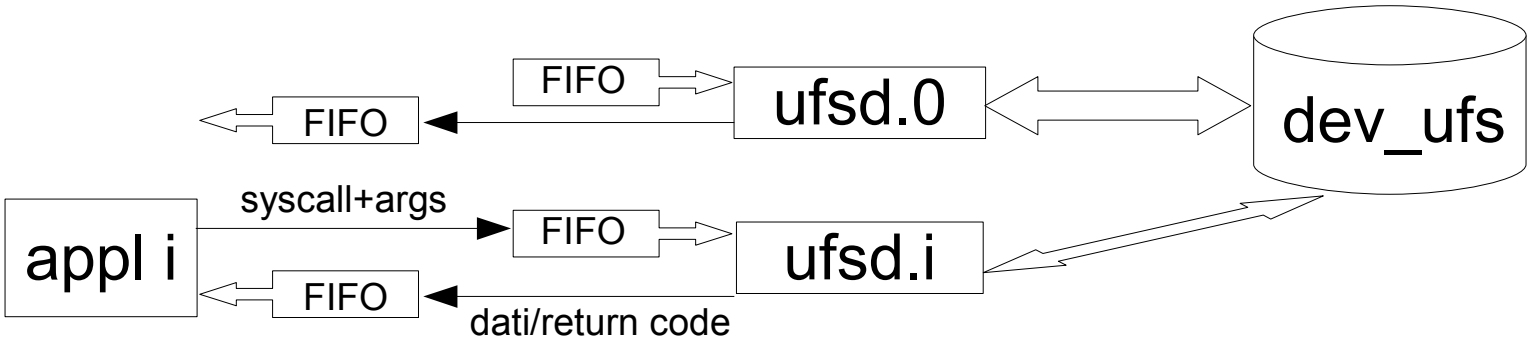
A) Richiesta di accesso al filesystem (fase iniziale)



B) Richiesta di accesso al filesystem (fase finale)

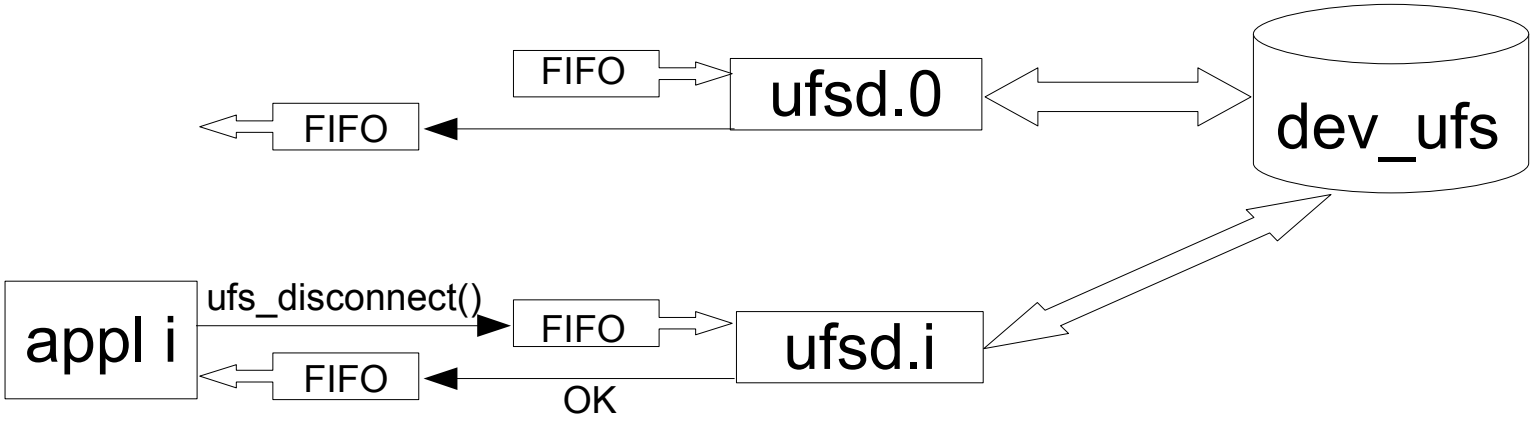


C) Situazione a regime: interazione applicazione-server

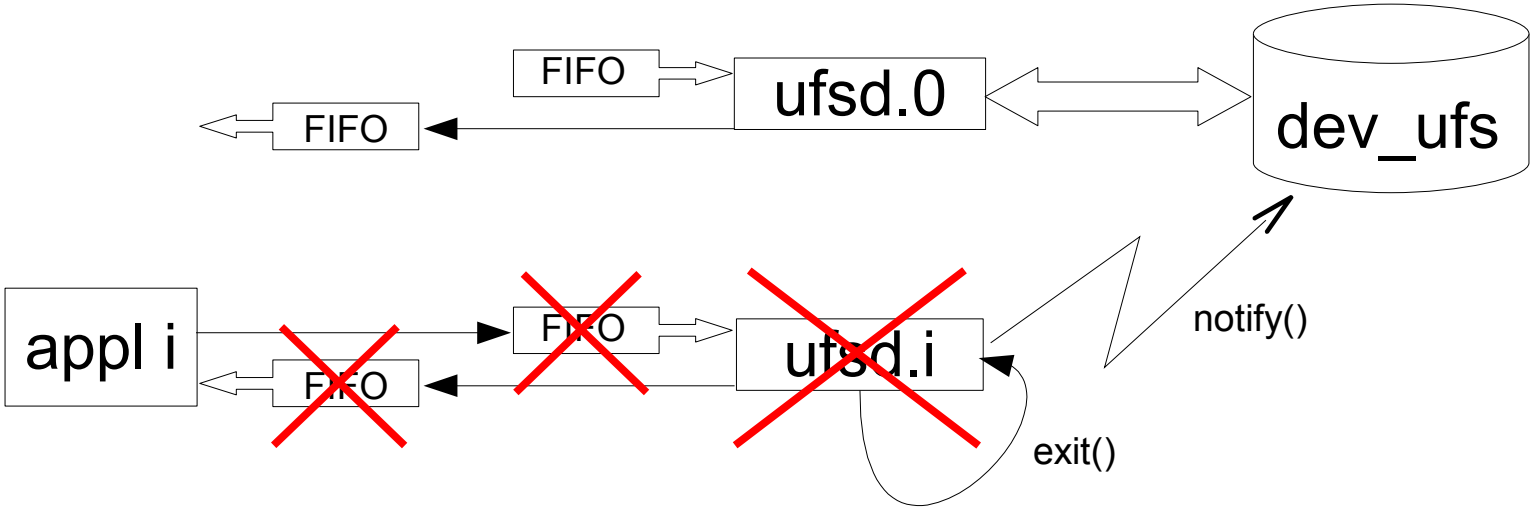


server ufsd: procedura di rilascio (schema indicativo)

A) Richiesta di rilascio



B) Terminazione del server dedicato e rimozione delle FIFO



Comandi base da implementare - I

Comandi base per il sysadmin (dal prompt della shell Linux)

- ufs_mke2fs – crea il file che conterra' il filesystem e ne organizza la struttura interna (si veda allegato su FAT32 per sintassi ed esempi)
- ufs_mount – rende il filesystem accessibile ad applicazioni utente mediante lancio ed inizializzazione del demone ufsd.0
- ufs_umount – rende il filesystem inaccessibile ad applicazioni utente mediante terminazione dei vari demoni ufsd.[0-N] attivi.

Comandi base da implementare - II

Comandi base per il test (dal prompt della shell `ufsh`)

- ls [filepath | dirpath] - stampa i dati della directory entry relativa al file filepath specificato o ai file/subdirectory contenuti nella directory dirpath specificata. Senza argomenti stampa i dati delle directory entry relative ai file/subdirectory contenuti nella directory root.
- cp [-s | -d] source dest - copia il file source sul file dest. Con opzione -s, source deve essere interpretato come nome di file nel filesystem Linux. Con opzione -d, dest deve essere interpretato come nome di file nel filesystem Linux.
- mv source dest - rinomina il file source in dest.
- rm filepath - rimuove il file filepath
- cat filepath – stampa il contenuto di filepath a video.
- mkdir dirpath – crea la directory dirpath.
- rmdir dirpath – rimuove la directory dirpath, se vuota.
- tree dirpath – stampa la gerarchia di subdirectory a partire da dirpath.
- altri possibili comandi di test possono essere definiti a piacere ed implementati come estensione opzionale

Il progetto: funzionalità opzionali

- Per il progetto, si deve implementare **UNA funzionalità** tra quelle sotto proposte
- L'implementazione CORRETTA di piu' di una funzionalità verrà considerato come un **bonus** da aggiungere al punteggio
- Si raccomanda di sviluppare tali funzionalità solo dopo che tutte le funzionalita' base richieste siano state sviluppate correttamente

File locking

- int ufs_fcntl(int fd, int cmd, struct flock *lock); cmd può valere F_GETLK, F_SETLK e F_SETLKW

File mappati in memoria

- void *ufs_mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);
- int ufs_munmap(void *start, size_t length);

Attributi di owner e permessi di accesso read/write

- UID
- rw per owner, rw per others

Package da consegnare - I

- La compilazione degli eseguibili e librerie del package **`ufs`** deve essere gestita mediante **`Makefile`**
- L'installazione degli eseguibili e librerie deve avvenire nella directory $HOME/ufs e **NON** deve richiedere il privilegio di root
- Il progetto deve essere documentato mediante una relazione che descrive:
  - l'organizzazione del package (albero delle directory, suddivisione delle funzioni nei vari file sorgenti, ...)
  - la struttura dei programmi ufsh, ufsd, ufs_mount, ufs_umount
  - le principali scelte implementative
  - il dettaglio delle funzionalita' supportate (con precisazioni sulle eventuali limitazioni e assunzioni sulla correttezza dei possibili dati di input da parte dell'utente)
  - le informazioni necessarie alla corretta compilazione, installazione e test (variabili d'ambiente, file di configurazione, ...)
  - eventuali programmi per il test delle varie funzionalita'

Package da consegnare - II

- La consegna deve consistere in un package, da spedire via mail, in formato tar compresso (**`tar czf`**) col nome ufs.tgz, contenente i sorgenti C, gli include file, il Makefile, la documentazione ed eventuali file per il test
- il file ufs.tgz deve contenere una sola directory (con pathname relativo) di nome **`ufs`**
- la directory **`ufs`** deve contenere:
  - file AUTHORS (Cognome Nome Matricola, uno studente per linea)
  - file README (info basilari sul package consegnato)
  - relazione in formato PDF o OpenOffice nella directory doc
  - directory src (sorgenti .c), inc (include .h), test (file/programmi di test), bin (eseguibili generati), lib (librerie generate), obj (file compilati .o), ed eventuali altre directory