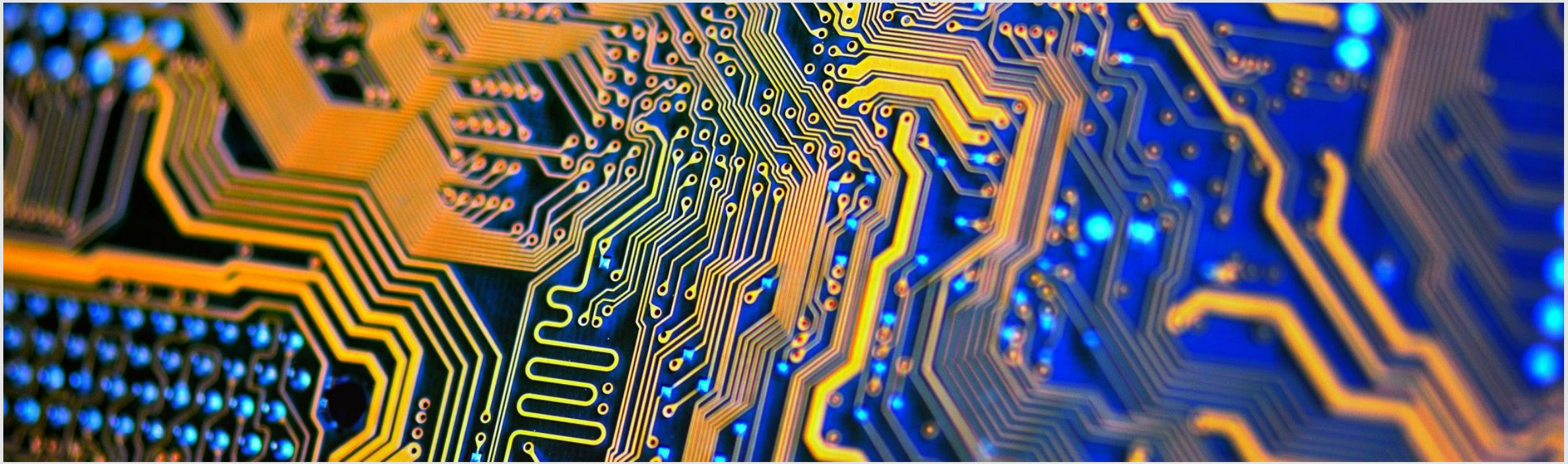
PROGRAMMAZIONE DI SISTEMA
A.A. 2022/2023

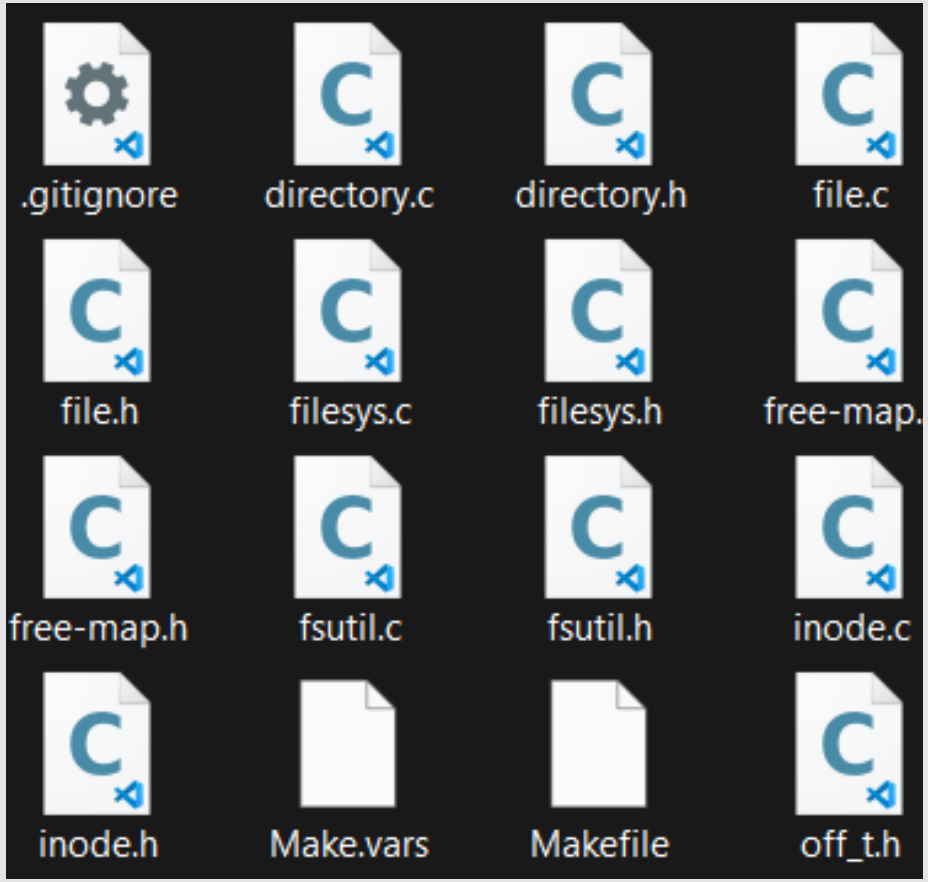
S318904 MARCELLO VITAGGIO
S317264 FABRIZIO VITALE

PINTOS - FILESYSTEM



INTRODUZIONE

Il filesystem di un sistema operativo svolge un ruolo cruciale nella gestione dei file su disco. Nel contesto di Pintos, esploriamo la sua implementazione, focalizzandoci sugli inode, le directory, i bitmap e le operazioni chiave del filesystem.



pintos/src/filesys

INODE IN PINTOS

Gli inode nel filesystem di Pintos rappresentano i file su disco e contengono informazioni cruciali come dimensioni, posizioni dei blocchi, permessi e timestamp. Questi inode differiscono sia nella rappresentazione in memoria che su disco, ognuno con attributi specifici e funzioni di gestione. La struttura degli inode in Pintos è definita nel file "inode.h" e "inode.c", dove vengono gestiti e utilizzati per supportare le operazioni di gestione dei file.

```
struct inode
  {
    struct list_elem elem;
    block_sector_t sector;
    int open_cnt;
    bool removed;
    int deny_write_cnt;
    struct inode_disk data;
  };
```

DIFFERENZE TRA INODE SU DISCO E INODE IN MEMORIA

Inode su Disco

- Rappresenta l'immagine persistente di un file su disco.
- Contiene attributi essenziali come la posizione dei blocchi e le informazioni di dimensione.
- Dimensione fissa di 512 byte nel filesystem di Pintos, anche se non completamente utilizzati.
- Definito dalla «**struct inode_disk**» nel codice sorgente.

Inode su Memoria

- Encompassa l'inode su disco e include informazioni aggiuntive necessarie durante l'esecuzione.
- Contiene dati come il numero di volte che il file è aperto, informazioni sul blocco associato al filesystem, e altro.
- Struttura definita dalla «**struct inode**» nel codice sorgente.

RAPPRESENTAZIONE DELLE DIRECTORY IN PINTOS

- Le directory sono rappresentate come file con i propri inodes in Pintos.
- Ogni entry di directory contiene il nome del file e il numero inode associato.
- La struttura dell'entry della directory è definita da «**struct dir_entry»** nel codice sorgente.

```
struct dir_entry
  {
    block_sector_t inode_sector;
*/
    char name[NAME_MAX + 1];
*/
    bool in_use;
  };
```

OPERAZIONI SULLE DIRECTORY

Creazione di una directory

bool dir_create (block_sector_t sector, size_t entry_cnt)

Apertura di una directory

struct dir *dir_open (struct inode *inode)

Chiusura di una directory

void dir_close (struct dir *dir)

Aggiunta di un file a una directory

bool dir_add (struct dir *dir, const char *name, block_sector_t inode_sector)

Ricerca di un file in una directory

bool dir_lookup (const struct dir *dir, const char *name, struct inode **inode)

UTILIZZO DEI BITMAP NEL FILESYSTEM DI PINTOS

```
free_map_allocate (size_t cnt, block_sector_t *sectorp)
{
  block_sector_t sector = bitmap_scan_and_flip (free_map, 0, cnt,
false);
  if (sector != BITMAP_ERROR
      && free_map_file != NULL
      && !bitmap_write (free_map, free_map_file))
    {
      bitmap_set_multiple (free_map, sector, cnt, false);
      sector = BITMAP_ERROR;
    }
  if (sector != BITMAP_ERROR)
    *sectorp = sector;
  return sector != BITMAP_ERROR;
}
```

- I bitmap nel filesystem di Pintos svolgono un ruolo cruciale nella gestione dell'allocazione dei blocchi
- Rappresentano la disponibilità o l'occupazione di settori (blocchi) nel filesystem.
- I bitmap sono implementati come array di bit, con ciascun bit che rappresenta lo stato di occupazione di un settore (blocco).
- Un bit impostato a 0 indica che il blocco è libero, mentre un bit impostato a 1 indica che il blocco è occupato.
- Quando un file viene creato o esteso, i bitmap vengono consultati per trovare blocchi liberi.
- Durante l'allocazione, i bitmap vengono aggiornati per riflettere l'occupazione dei nuovi blocchi.

PINTOS VS. OS/161

ARCHITETTURA

- Pintos:
  - Progettato per l'architettura 80x86 (Intel x86).
  - Supporta thread del kernel, esecuzione di programmi utente, file system e memoria virtuale.
- OS/161:
  - Progettato per l'architettura MIPS.
  - Supporta fino a 32 processori, con funzionalità multiprocessore.

SIMULATORI

- Pintos:
  - Eseguito su simulatori di sistema come Bochs e QEMU.
  - Simulano una CPU 80x86 e i dispositivi periferici associati.
- OS/161:
  - Utilizza il simulatore di macchina System/161.
  - Supporta funzionalità avanzate come il debug remoto e il profiling del kernel.

ANALISI COMPARATIVA FILE SYSTEM: PINTOS VS OS/161

PINTOS

Limitazioni Conosciute:

- Mancanza di sincronizzazione interna.
- Dimensione del file fissa alla creazione.
- Allocazione dati come un'unica estensione.
- Assenza di sottodirectory.
- Nomi file limitati a 14 caratteri.
- Possibili corruzioni del disco in caso di arresto anomalo.

Operazioni Speciali:

- Semantica simile a Unix per filesys_remove(): blocchi non deallocati se il file è aperto.
- Creazione di un disco simulato con pintos-mkdisk.
- Copia dei file dentro e fuori dal sistema file simulato con pintos -p e -g.

OS/161

Configurazione dei Dischi	• I dischi sono numerati in base all'ordine di rilevamento sul bus di sistema. • Configurazione nel file sys161.conf con immagini dei dischi (es. LHD0.img).
Creazione di Immagini Disco	• Uso di disk161 per creare o ridimensionare immagini disco. • Creazione delle immagini al momento dell'avvio o tramite disk161.
Formattazione del Disco	• Utilizzo di mksfs per preparare le strutture on-disk prima dell'uso. • Esecuzione da OS/161 o da fuori.
Montaggio e Accesso	• Montaggio del filesystem con il comando "mount" nella console OS/161. • Accesso ai file e alle directory con comandi come cd, cp, ls.
Bootfs e Root Directory	• Introduzione di bootfs come volume di avvio. • Possibilità di cambiare il bootfs da menu.
Controllo e Ripristino	• Utilizzo di sfsck per controllare e ripristinare la consistenza del filesystem. • Eseguibile da dentro o fuori OS/161, principalmente a scopo diagnostico.

HTTP://OS161.ORG/RESOURCES/SFS.HTML
HTTPS://WEB.STANFORD.EDU/CLASS/CS140/PROJECTS/PINTOS/PINTOS_3.HTML#SEC35