lunedì 21 gennaio 2019

Appunti e definizioni SO

File System:

File System: Risiedono su storage secondario. Si rendono necessari per facilitare la mappatura di indirizzi logici in indirizzi fisici. Rendono efficiente e conveniente l'accesso a disco essendone facile il recupero.

File: Insieme di blocchi di dati.

Directory: Collezione di file o altre directory.

Mount Point: È la directory a cui si lega un altro file system di un Device generico.

Link: Puntatore a file nel file System. Se logico non intacca la eliminazione del file. Se fisico e si eliminano tutti i link il file viene eliminato.

Generali: La cartella principale è "root" (/) contiene diverse directories e file e ulteriori fs di altri Device. /proc informazioni sui processi in running. /sys informazioni sul sistema. /dev vista diretta sui Device collegati.

Mount/Unmount Operation: è l'operazione che permette di mappare un file System esterno su un altro fs attraverso un Mount Point (aggancio). Ne risulta che il fs aggiunto sia un sottoalbero della cartella individuata come Mount Point. Potrebbe richiedere privilegi. L'operazione di unmount è SIMPLY l'opposto.

File: Un file è una parte di memoria permanente formata da indirizzi contigui. Il contenuto è definito dall'utente. Il file ha diversi attributi tra cui data di creazione, dimensione su disco, path in cui è salvato, permessi ecc. ecc. ecc. Il file appartiene ad un possessore e ad un gruppo. L'utente può appartenere a uno o più di questi classi: user, Group, others. Ogni classe ha permessi differenti di r/w/x che possono essere modificati tramite chmod <ugo>. Tutte le operazioni su file sono nient'altro che wrapper che nascondono syscall specifiche di sistema.
Quando qualcosa non è realmente un file, viene trattato come se lo fosse tramite una interfaccia file-like.

Directory: è un file di file entries ossia: o hard link di un file (definizione fisico sopra), o soft link (definizione logico sopra) oppure other directories. Può ospitare un altro file System se presente. La struttura del file System Unix, dettata dall'uso di link, risulta un grafo. A volte l'os prevent loops.

File Descriptor: è un intero che caratterizza un file all'interno di un processo ottenuto da dalla creazione o dall'apertura di un file.
Lo stesso file può essere aperto più volte, ogni apertura ritorna un fd differente.
Tre descrittori sono aperti di default, 0 1 e 2, che sono rispettivamente: stdout,

stdin, stderr. Tutte le azioni su un file avvengono tramite il suo descrittore. Visto che il file è una sequenza di bytes, può essere visto come un array che può essere scritto o letto in blocchi. Ogni file descriptor è associato ad un puntatore alla posizione del file all'interno dell'OS. Le operazioni di read e write avvengono alla posizione corrente del puntatore, e fanno side-effect su di esso. Esistono funzioni come lseek che manipolano la posizione del puntatore.

Devices: la directory /dev provvede ganci ai Device fisici nel file System (esempio: /dev/sda è il disco). In unix i Device sono accessibili come file, quest'ultimi sono visti in due tipi: blocchi e caratteri. I Block Device supportano seek e Block read. I device caratteri sono streams. L'interfaccia del file è un modo unificato di unix che permette l'accesso ai device, tutto è visto e trattato come file. La syscall ioctl() permette di accedere a funzioni speciali del device, è un aggancio al driver e funziona in modo simile ad una syscall annidata, prende come parametro una richiesta (un intero) che specifica l'azione da eseguire. Ogni driver installa la sua richiesta e il suo request handler quando viene caricato. I device caratteri possono essere configurati tramite l'interfaccia termios.

Mmap: Possiamo mappare un file in memoria per accederci più velocemente.

Physical Devices: Il sistema operativo deve affrontare le problematiche di ogni singolo device, l'implementazione dovrebbe massimizzare la coerenza dell'interfaccia e forzare i driver a seguire delle specifiche. Il driver è una classe virtuale che dovrebbe sovrascrivere i metodi standard definiti dall'interfaccia.

Disk: I dischi possono essere organizzati in partizioni. La tabella di partizione descrive il layout del disco. Ci si accede tramite controller che traducono istruzioni read/write block. DMAC è spesso usato.

FS Implementations: Dobbiamo mappare il fs nel file generale del disco. L'os dovrebbe essere a conoscenza del fs che il disco monta per gestirlo in modo corretto. Possono coesistere più fs diversi su uno stesso disco.

FS Driver: è il componente dell'os che tratta con l'fs. Fornisce un'interfaccia uniforme (open, read, close…).

Strutture dati di supporto: Per ogni file il kernel crea una struttura OpenFileInfo. Per ogni fd l'os crea una struct OpenFileRef inserita nel PCB del processo possessore dell'fd precedente. La struct OpenFileRef punta alla OpenFileInfo e salva un puntatore indipendente per le operazioni su file.
Ogni file su disco è caratterizzato da una struct chiamata FCB che contiene tutte le informazioni su di esso. Ogni directory ha un header (struct) che estende FCB.

Allocazione contigua: L'idea è assegnare blocchi nel disco contigui per ogni file salvato. Ad ogni file nel proprio FCB verrà indicato l'indice del primo blocco e il

numero di blocchi totali. Problema principale è find space e frammentazione esterna.

Allocazione Linked List: L'idea è che nel FCB si salva solo l'indice iniziale e finale del file. In ogni blocco ci sarà uno spreco di memoria per conservare i puntatori al blocco successivo e a quello precedente. Ottimo è l'accesso sequenziale, sennò piscia.

FAT: File Allocation Table. È inserito un array all'inizio del disco. Serve a codificare le sequenze di blocchi. È copiato in RAM essendo molto piccolo.

Allocazione Indexed: Ci sono due tipi di blocchi: dati e indici. All'interno del FCB abbiamo un blocco indice in cui sono contenuti in ordine tutti gli indici dei blocchi che compongono il file. Comodo per operazioni di modifica di file o di accesso indicizzato. Può essere strutturato a più livelli se il device ha un enorme spazio di memoria.

Directory implementation: Linear List ossia una lista di nomi di file associati a puntatori ai stessi.Semplice da programmare ma la find è lenta. Hash Table linear list con strutture dati hash. Diminuisce il tempo di ricerca ma vanno gestite le problematiche legate all'hashing.

Free Space: BitMap, all'inizio del disco c'è una mappa di bit riferito ad ogni blocco. 0 se è libero, 1 se è allocato. Un po lentina ma è coerente con il layout del disco. Linked List => SLAB Allocator.

IPC:

Scopo: I processi spesso hanno bisogno di dover comunicare tra loro per scambiarsi informazioni o per dividersi un task. Le modalità descritte sono Message Passing e Shared Memory.

Message Passing: A differenza della Shared Memory il Message Passing ha bisogno di copiare la memoria (messaggio) attraverso una struttura centrale chimata Mailbox alla quale tutti i processi interconnessi si interfacciano. È più facile da implementare ma se ne perde di efficienza. Le primitive su cui si basa sono send() e receive() le quali avranno comportamenti differenti in base alle specifiche del Message Passing, ossia (bloccante o non bloccante, diretto o indiretto, limitato o illimitato.

Funzionalità MP: Si basa come detto su una struttura gestita dall'os chiamata Mailboxes con la funzione di message queue in cui i processi sender accoderanno messaggi e quelli receiver li estrarranno dalla testa. I processi hanno bisogno di essere a conoscenza dell'id della mailbox per interagire con essa. Una serie di

primitive: open(), close(), unlink(), get/set attr, put, wait/notify. All'interno della cartella /dev/mqueue sono presenti tutte le queue create dai processi in running.

Shared Memory: È un'area di memoria condivisa tra i processi. Simile al message queues o ai semafori. Hanno funzionalità simili come la open(), la close(), o la destroy(). Una shared memory DEVE essere mappata una volta aperta nella memoria del processo tramite mmap, come se fosse un file (Vedi PIPE o FIFO). In /dev/shm sono visibili tutte le shared memory opened dai processi in running.

AVR:

Digital Ports: Le porte digitali A-L sono mappate in memoria. Ogni bit della porta corrisponde ad un pin. Sono accessibili in C tramite variabili (che corrispondono ai registri a 8 bit presenti nel chip). Indicherò con X la lettera della porta:
DDR<X> direzione della porta => se bit=1 allora è un output, else input. (In scrittura)
PORT<X> output port => se bit=1 e se il pin è settato come output leggo un valore di +5V, se il pin è settato a input è abilitato sarà attivato il resistore di pull up, se bit=0 se pin in output allora leggo 0V. (In scrittura)
PIN<X> input port => se bit=1 leggo un logico 1 senno leggo logico 0. (In lettura)

Timers: Sono periferiche che possono essere usate per misurare intervalli di tempo, leggere il tempo ad un evento esterno, generare onde, generare interrups. Ci sono 5 timer, qualcuno a 16 bit e altri a 8 bit di counter. L'incremento può arrivare ad ogni colpo di Clock o ad una frazione della sua potenza di 2. Un timer ha 3 Output Compare Registers OCR<X>{H/L}, quando il valore matcha con quello del registro si può generare un evento. Il comportamento da eseguire all'occorrenza dell'evento è controllato da un insieme di speciali registri TCCR<X>{L/H}.

PWM: I timers possono essere usati per generare forme d'onda. Possiamo comandare il timer di settare a 1 (o 0) un bit se il valore del timer si trova sotto una certa soglia e di fare il contrario se dovesse trovarsi sopra. In un'onda quadrata il ciclo di lavoro è la frazione del periodo per cui il segnale rimane alto: se il segnale è al 50% alto allora risulta una vera onda quadra.

Interrupts: Possiamo usare i timer per generare un interrupt ogni intervallo di tempo predefinito. Bisogna settare la frequenza del timer dettando il prescaler desiderato, selezionare un OCR register, dire al timer cosa fare quando avviene il match con OCR, settare TIMSK<X> register per avvertire che c'è stato un interrupt e scrivere un interrupt handler.

Serial Port: una porta seriale è controllata da UDR<X> un registro che contiene i dati da inviare/ricevere.UBRR<X> un registro baud rate che indica quanto veloce la transizione deve avvenire, UCSR<X> un semplice status register. Per scrivere un byte sulla seriale bisogna scrivere un dato su UDR<X>, usando le flag di UCSR<X>

possiamo sapere se il dato è stato correttamente inviato. Per leggere invece sempre da UCSR<X> controlliamo le flag per vedere se qualcosa è stato ricevuto, in caso leggessimo 1 allora il dato è già caricato in UDR<X>, la lettura azzera la flag.