

CONTEXT SWITCHING

Cos'è un contesto?

Sì riferisce ad un processo, e dà informazioni su:

- registri CPU
- memoria d. processo (stack, .text, .data, -bss)

Viene gestito attraverso `ucontext` in C, in assembly è da gestire manualmente.

`ucontext_t` è struct che contiene informazioni sul contesto

- `getcontext (ucontext_t *ucp);`
salva il contesto corrente in ucp, utile per costituire baseline di contesto
- `setcontext (ucontext_t *ucp)`
carica il contesto salvato in cucp
- `makecontext (ucontext_t *ucp, void (*func)(), int argc, ...)`
crea contesto "trampolino", in modo che se caricata venga chiamata la funzione func
- `swapcontext (ucontext_t *oucp, ucontext_t *ucp)`
salva il contesto corrente in oucp, e carica il contesto ucp,

possibile costituire un contesto switcher attraverso

TCB, task control block, che nell'implementazione del prof era usata come double linked list, con salvataggio dello stack context e con relativi registri

↳ utile per context switch AVR

MEMORY Allocators

Qui si studierà al processo dietro ad una funzione come malloc, per dare blocchi di memoria ai processi, gestendo problemi di:

- frammentazione
- tempo di get/release d. blocchi

SLAB Allocator

Comodo quando si hanno molti oggetti d. dimensione fissa, e il numero massimo d. oggetti è noto
↳ es. Blocchi:

C. sarà una divisione della memoria in CHUNK.
ind. riga d. memoria saranno allineate secondo

$$\begin{aligned} \text{ptr_block} &= \text{buf_start} + (\text{idx} * \text{item_size}) \\ \text{idx} &= (\text{ptr_block} - \text{buf_start}) / \text{item_size} \end{aligned}$$

Sì manderà un struttura auxiliaria di blocchi liberi, toglandoli quando allocati ed inserendoli d. nuovo quando liberi. (pila)

Verrà implementata con ArrayList con

- variabile d. start-position
- array d. interi;

Null array d. interi, l'intero in posizione i determina il puntatore al prossimo elemento.

Se c'è -1 allora o non c'è in list, o c'è null

Sarà poi il PoolAllocator a gestire la slab, attraverso una struct contenente:

- N-max + item-size elementi
- array list d. dimensione h-max

Buddy Allocator

Utilizza sistema ad albero per allocare dei blocchi a dimensione variabile.

Si parlerà di buddy come d. quella zona di memoria risultante dal partizionamento del parent allocator.

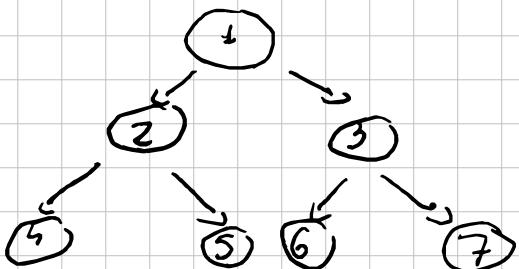
Si avranno potenziali problemi d. frammentazione

Utilizzo d' Alberi !

level (idx) = $\text{floor}(\log_2(\text{idx}))$

parent (idx) = $\text{floor}(\text{idx}/2)$

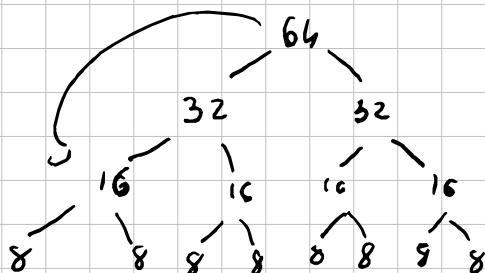
buddy (idx) = ($\text{idx} \geq 2$) ? $\text{idx}-1, \text{idx}+1$



Ogn' buddyList/item ha associato ad esso un blocco di memoria d. dimensioni dettate dal suo livello.

Manteremo nell' allocator una free list per ogni livello, mantenendo uno SLAB allocator per gestire le liste

$$\text{floor}(\log_2 11) = 3, \dots$$



INTERRUPTS

- Metodo in cui si battono gli OS moderni, sono segnalati che interrompono il flow dell'esecuzione per gestire:
- eventi esterni (timer, I/O)
 - **eventi interni**
 - call esplicite (system call)

Migliore alternativa del polling-based, meno overhead.

INTERRUPT VECTOR è array di puntatori a funzioni contenenti indirizzi dell'ISR (interrupt service routine), ognuno d. essi dedicato ad una routine.

syscall-vector ≠

INTERRUPT ID	ISR pointer
0 (es. reset)	0x...
1 (es. I/O)	0x...
5 (trap)	0x ...

Eccezioni:

Interrupt software per segnalare qualcosa. In x86 possono essere

- **Traps** : ISR invocato dopo l'istruzione
- **fault** : es. page fault; invocato prima dell'istruzione
- **abort** : errore irrecuperabile

Gli eventi sono poi gestiti a livello d. OS

INT & CALL

INT salta all'indirizzo di memoria dell'ISR puntato dall' indice XX.

Viene fatta elevazione di privilegi, si passa allo supervisor mode

CALL salta all'indirizzo di memoria indicato YY, che può essere qualsiasi indirizzo valido nello spazio di memoria eseguibile del processo. Esecuzione continua in User mode

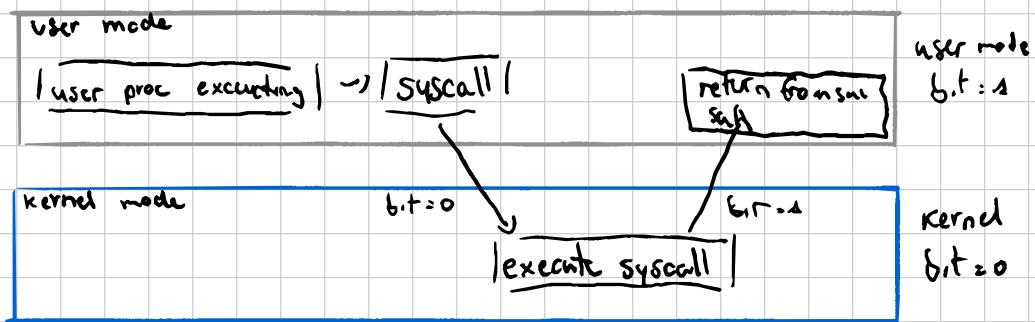
Onde evitare possibili manomissioni, accidentali o volute, dell' ISR, l'esecuzione di esse è effettuata in privileged mode, che è un flag nella memoria di processo modificabile solo in privileged mode, in cui si ha accesso a molte più operazioni.

Alcune entità dell' ISR possono essere eseguite in user-mode senza che esse scatenino un protection exception

SYSTEM CALL

Entry point nel kernel. Ogni system call ha un suo numero associato.

Per chiamare una syscall verrà chiamato ISR che è il syscall handler (in i386 è INT 0x80), inserendo nei registri il numero relativo alla syscall d'interesse



Il cambio di bit di modalità è gestito dall'IRET
(return from interrupt)

In generale le syscall effettuano operazioni low-level,
non sono le stesse operazioni su tutti gli OS,
servirebbe ricorrere ogni app per lo specifico flavour
OS.

Syscall Wrappers - STANDARDS

passano essere a livello di libreria, come
con libc: `fprintf`, `fopen`, `fclose`, **0**

a livello di sistema, nel caso di componenti
più ad alto livello cosa thread, network
e sincronizzazione. Ci sono ad es.

POSIX

WINAPI

Processo

Un processo è un programma, o una sua parte, che è eseguita dall'OS. È caratterizzato da:

- Registri CPU

- Memoria:

- stack

- heap

- code .text

- variables , data , .bss

- mmap region

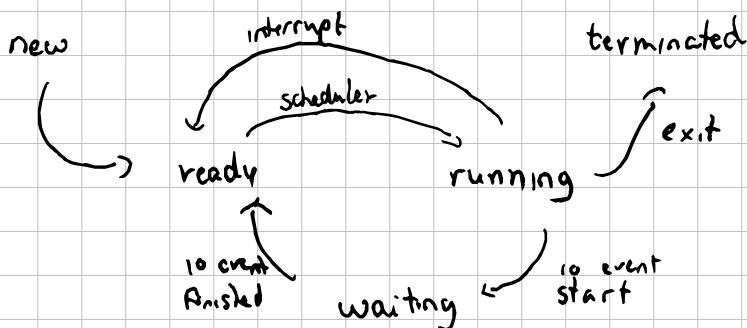
- Risorse:

- file/socket descriptors 0x8040000

- synchronization construct 0x00000000

stack pointer è contenuto in un registro, e dentro lo stack orremo: argomenti di funzioni, return address, registri salvati (push,pop) e variabili locali.

process state



fork()

funzione con cui un processo può creare un altro processo.
Dopo l'invocazione a buon fine di una fork si creeranno
due istanze dello stesso processo, una identica all'altra,
con la differenza del pid assegnato (0 per child, chld-pid per
padre)

vengono copiate sia la memoria che i descriptori!
(a meno dell'invocazione di Vfork, usata per exec)

wait()

mette il processo corrente in waiting fino a che
qualsiasi figlio (wait) o uno specifico (waitpid) termina,
salvando l'output in un intero, visualizzabile con una
macro.

S. va a formare un process tree con relazioni padre
figlio! Se un processo padre termina prima del processo
figlio, allora l'ultimo diverrà figlio del relativo processo padre

L'avviso al padre della terminazione del figlio viene fatto
attraverso **SEGNALI** (SIGCHILD), discusisi dopo.

La gestione dei segnali può essere fatta attraverso
sigaction o signal, col codice del segnale e il relativo
handler da usare

exit()

funzione che fa terminare l'esecuzione di un processo chiamata anche
quando il main termina.

Una volta terminato un processo entra in uno stato zombie,
e tutte le sue risorse sono liberate. Rimarrà zombie
fino a che il processo padre leggerà il return value,
e a quel punto muore.

Init/system chiama periodicamente una wait per far
si che nessun processo zombie orfano ci rimanga
indefinibilmente.

exec & env

exec()

Funzione che rimpiazza la memoria del processo chiamando con quelle caricate dalla funzione exec, rilasciando anche le risorse.

exec inizia la sua esecuzione dalla label _start del processo, che è su tutti gli eseguibili Linux.

Se fanno necessari ulteriori argomenti è possibile usare:

- variabili d'ambiente (vedi corso di sicurezza)
- parametri main (argc, argv)

Execve accetta l'utilizzo di parametri.

Le variabili d'ambiente sono un array di stringhe della forma

"NAME = VALUE"

es

\$PATH

CPU SCHEDULING

Pratica con il quale lo scheduler sceglie il prossimo processo da eseguire fra quelli ready.

In genere lo scheduler CPU è sempre chiamato durante un'attività I/O, che è lento d. natura

Preemption

nel batch scheduler, la norma è che un processo non viene mai terminato se non chiude attività I/O, quindi uno scheduler di tipo preemptive attende un quanto di memoria, per poi fermare indiscriminatamente il processo

(o switching è costoso (cache da buttare))

CPU / I/O BURST

I programmi utente, in genere, sono caratterizzati da

CPU BURST
I/O BURSTS

In generale ci si concentra nei burst di tipo CPU, che sono i più resource consuming.

E' possibile distribuire i burst d. CPU per il loro tempo, mettendo nelle q. il numero d. burst che durano t. millisecondi

Accуни мережі для оцінки scheduler

- | | |
|-----------------|--|
| MAXIMISE | <ul style="list-style-type: none">• utilizzo CPU: tempo utilizzato per far eseguire processi dalla CPU• throughput: numero di processi che si possono eseguire in un'unità di tempo.• turnaround time: tempo per completare un processo |
| MINIMIZE | <ul style="list-style-type: none">• waiting time: tempo che un processo spende nella coda ready• response time: tempo che conta da quando l'utente chiede qualcosa a quando succede effettivamente |

NON PREEMPTIVE

• First Come, First Served (FCFS)

algoritmo semplice di accodamento, prende il processo in testa e lo esegue. Un processo verrà poi accodato.

Problema:

	P ₁	P ₂ P ₃
0		2h 27 30

average waiting time

$$(0 + 2h + 27) / 3 = 17 \text{ h}$$

Processo	Burst
P ₁	2h
P ₂	3
P ₃	3

	P ₂	P ₃ P ₁
0	3	6

average waiting time

$$(0 + 3 + 6) / 3 : 3 = 3 < 17$$

questo accade per l'effetto di convoglio:
i processi più piccoli
primo abbussano il waiting time.

Shortest Job first (SJF)

Sfrutta l'effetto convoglio per avere il waiting time in assoluto minore, ma è necessaria la conoscenza progressiva del burst time! Viene approssimato, sfruttando il fatto che lo stesso processo fa, in media, CPU burst di uguali dimensioni.

Utilizzo di una funzione di stima

$$\hat{b}_{t+1} = \alpha \hat{b}_t + (1 - \alpha) \hat{b}_t$$

↓ ↓ ↓
prediction measurement prediction
t+1 in t in t
↓ ↓ ↓
 decay coefficient

PRIORITY SCHEDULE

i processi hanno assegnato un numero di priorità, e quelli a priorità maggiore vengono eseguiti per primi. Onde evitare una situazione in cui processi a bassa priorità non vengano mai eseguiti, c. sta sistema di "aging"

PREEMPTIVE SCHEDULERS

L'idea di un preemptive scheduler è di allocare ad ogni processo un massimo di un quanto d. tempo, per fare in modo che anche se il burst CPU durasse d. più anche gli altri processi possono avere l'opportunità di eseguirsi.

- Se il processo messo in pausa è già **masso in coda**, si parla di **ROUND ROBIN (RR)**

Il context switch deve essere nettamente minore del tempo d., quanto "q", oppure peserebbe troppo sullo scheduling.

$$\hookrightarrow q \approx 80\% \text{ del tempo d., burst CPU}$$

Multi Level Queue

utile per dividere code in priorità, ad esempio come "background" e "foreground"

L) FCFS R) RR

Lo scheduling in sistemi multi-core è più complicato, necessari meccanismi di sincronizzazione

REAL-TIME SCHEDULER

• **Soft real-time**: nessuna garanzia sul momento in cui un processo real time verrà eseguito.

• **Hard real-time**: c'è un **requisito forte di "deadline"**.

Per i processi cicli (es. aggiornamento posizione del mouse) avremo

- processing time t_i

- deadline d_i

- period T_i

$$U = \sum_i \frac{t_i}{d_i} < 1$$

Per scegliere il modello di scheduling c'è possibile trovarlo deterministicamente oppure lo s. può trovare **simulando** processi e **modellando** lo scheduler

MEMORIA

Assieme ai registri, la memoria è l'unica cosa a cui la CPU ha direttamente accesso, e attraverso anche la cache è abbastanza veloce.

Avvio di programma

Si accede al file header per altre informazioni su come è associata la memoria del processo e le librerie dinamiche da linkare all'esecuzione.

Le informazioni sono contenute nell'**ELF HEADER**

C: Sarà procedura di Address Binding, in modo che i processi non devono sempre trovarsi nella stessa zona fisica di memoria (che altrimenti potrebbe essere la stessa di un altro processo)

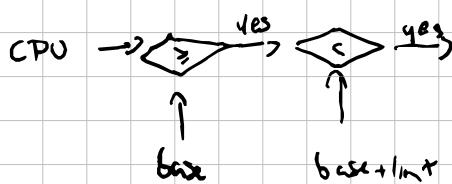
↳ from relocatable to physical address

MEMORY MANAGEMENT UNIT

Unità fisica che è dotata al mapping della memoria virtuale a quella fisica; considerando che la CPU lavora con indirizzi virtuali.

protezioni:

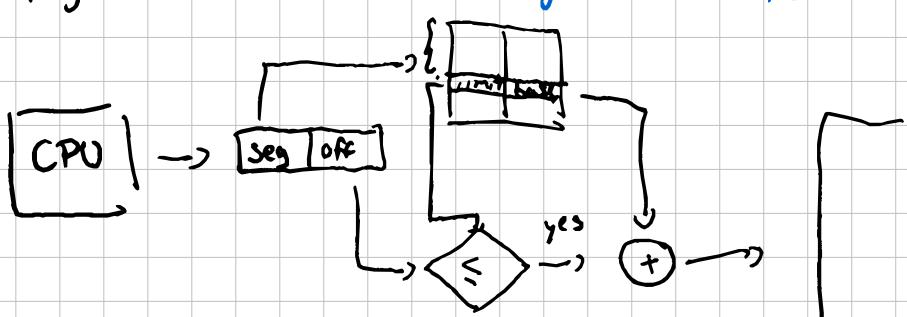
base + limit per evitare accessi ad altri processi



idea migliore: SEGMENTATION

→ utilizzare un registro esterno che mantenga i siti più alti di una zona d. memoria.

mettendolo in hardware, ci possibile costruire una dupla logica che consiste in **<segment-number, offset>**



mappatura segment, limit, base nella TABLE

PAGING

Metodo di virtualizzazione della memoria.

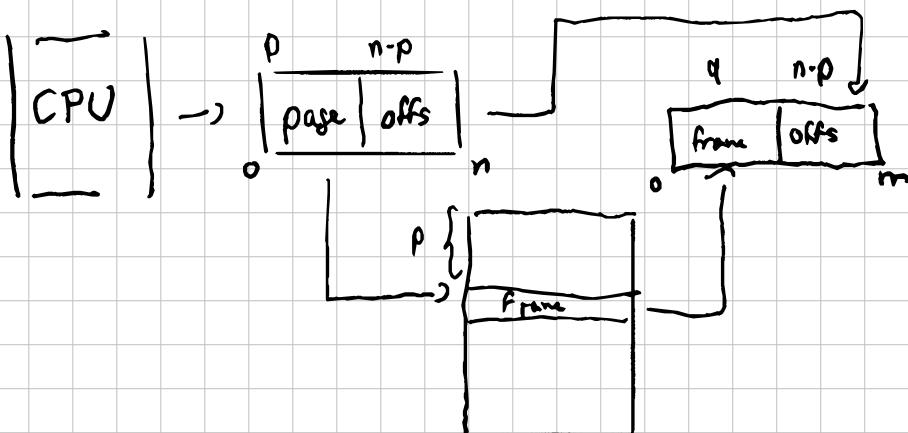
- Divisione della memoria in blocchi di dimensione fissa, chiamati **frame**
di dimensione 2^x
- Divisione della memoria logica in blocchi della stessa dimensione, chiamati **page**
- Creazione della page table per traslare blocchi logici in fisici

Pro

Nessuna frammentazione esterna

Contro

- frammentazione interna, soprattutto con frame grandi
- 2 c.d.: dr. memoria per accedere, meno con TLB
- serie di allocare spazio per la page table



possiamo aggiungere alle tabella un bit per indicare o meno la validità dell'indirizzo (es. quando se c'è swap)

TLB

Translation Lookaside Buffer: è una memoria associativa nell'CPU che memorizza alcune delle entrie delle page table.

↳ usata come "cache" del registro fisico associato.
si basa sul fatto che una zona di memoria acceduta ora sarà probabilmente acceduta presto.

Molto più veloce della RAM, ma poche entrie.

Nel momento in cui la CPU richiede un accesso in memoria si verificherà prima che il numero di pagina non è già presente nella cache TLB, in caso di miss guarda nella page table

Ogni processo ha la sua tabella delle pagine, nel quale ci saranno anche delle page references e zone di memoria condivise (nMAP, SHNEM)

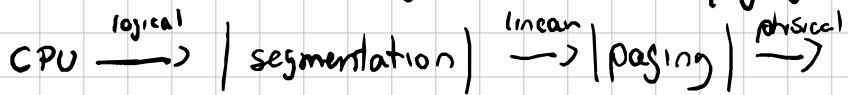
EFFECTIVE ACCESS TIME (EAT)

È il tempo medio di attesa per calcolare l'indirizzo fisico

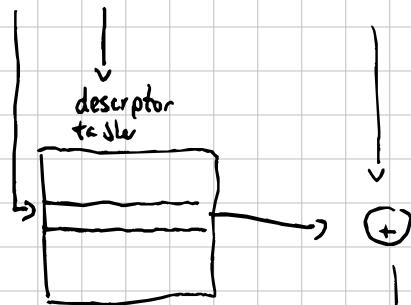
$$EAT(\alpha) = \underbrace{\alpha (T_{\text{DRAM}} + T_{\text{TLB}})}_{\text{TLB hit}} + (1-\alpha) \cdot 2(T_{\text{DRAM}} + T_{\text{RS}})$$

con α il HIT ratio, che spesso è vicino al 99%.

Combinazione di segmentazione e paging.

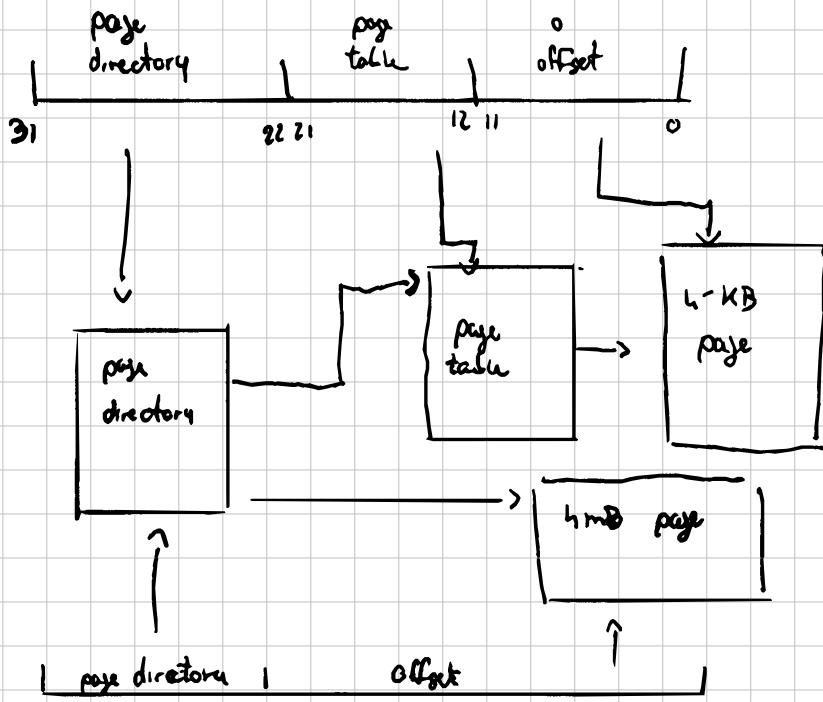


logical address | selector | offset



linear address
page 1, page 2, offset 1
10 10 12

Viene usata in questo modo



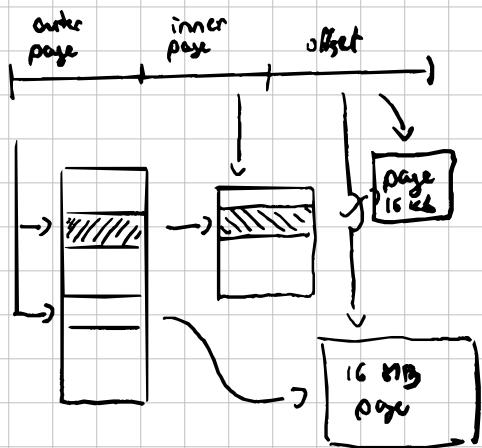
ARCHITETTURA ARM

funzionalmente simile, architetture a 32 bit formata da

- 4 or 16 kb pages
- 1 or 16 mb pages - sections.

2 TLB levels:

- Outer is micro-TLB, one data, one instruction
- inner is single main-TLB



In generale non serve avere l'intero programma in memoria
se si sta eseguendo solo una parte di esso.
↳ Non limitati più dalla memoria fisica, pur
mantenendo lo stesso meccanismo di memoria
logica.

Aggiungendo, è insensato allocare ad ogni programma in esecuzione
lo spazio per l'intera libreria dinamica: funzionalità di
SHARED PAGES. Ci sta mapping in indirizzi virtuali.

Ad esempio, con fork() si condividono le pagine al posto
di copiarle.

SWAPPING

Processo con cui una parte della memoria fisica può essere spostata al di fuori della memoria, in altri dispositivi più capienti, per poi essere ricaricata in memoria all'esecuzione.

memoria utilizzata > memoria fisica

In generale ci sono algoritmi di swapping, ma il tempo maggiore è quello di trasferimento (dischi rigidi più lenti della Ram).

Disattivata di default nei nuovi computer, in generale nel processo di swap-in non c'è sempre necessario riportare il dato nella stessa zona di memoria.

- SWAPPING PROCESSO MOLTO LENTO
- impossibile swap se processo in stato pending I/O

Al posto di portare tutto il processo in memoria a tempo di arrivo, copia in Ram solo quando c'è necessario (**paging on demand**), ci sarà un bit nella tabella delle pagine che sta ad indicare dove si trova la zona di memoria (**page fault**)

$$EAT = (1-p) \cdot T_{RAM} + p (T_{RAM} + T_{swap-in})$$

p : page rate fault, idealmente $\rightarrow 0$

per avere performance degradation minimi $p < 0.000025$!

Alcune ottimizzazioni

RAW SWAP SPACE il filesystem rallenterebbe troppo la procedura di swap, si usa quindi raw storage

COPY ON WRITE (cow) nel momento del forking copia solo la tabella delle pagine del gestore, mettendo un flag.

Se processo padre tenta di accedervi si genera un "trap-on-write", in cui il frame è copiato e il bit è a 0

FREZE FRAMES Mantengono lista di frame liberi (come SLAB) in cui può essere fatta copia di swap

es. Cow

- Processo 1 (P_1) fa fork e genera processo 2 (P_2)
- copiate solo page table, bit trap on write settato (ancora nessuna copia del frame)
- Se processo P_2 prova a scrivere una pagina, trigger del trap e viene copiata la pagina
- Bit per pagine accodato a 0

PAGE REPLACEMENT

- 1) Trovare pagina desiderata in swap
- 2) Trovare un frame libero:
 - se esiste copia lì: la pagina
 - altrimenti trova frame vittima su cui fare swap-in
- 3) Copiare la pagina desiderata nel frame libero trovato
- 4) Continua l'esecuzione da quello che ha scatenato il trap
 - aumenta CTR, due accessi a memoria di swap

Simulazione possibile attraverso simulazioni di accessi alla memoria (con array $S[i]^x$ a tempo i ; sistema usa memoria x)

ALCUNI ALGORITMI

FIRST IN FIRST OUT (FIFO), sceglie come vittima la pagina swapped per ultimo (caratteristiche di Belady se frequenze "sfasate")

LEAST RECENTLY USED (LRU) modello ottimale, che guarda al futuro, rimpiazzando le pagine che non vengono usate per più tempo.

Si possono fare alcune inferenze sul passato attraverso un contatore che si aggiorna ad ogni clock (troppo lento)

SECOND CHANCE: aggiungendo un reference bit nelle tabella delle pagine, (1, se la relativa pagina è stata referenzialata) se, nella ricerca d. pagina vittima, vedremo:

- reference = 0 rimpiazzarla
- reference = 1 setta reference bit a 0, ma lascia in memoria e verifica la pagina dopo con le stesse regole.

Si crea una coda circolare di pagine

Ci sta versione **enhanced second chance**, utilizzando 2 bit <access, modify>

Thrashing è fenomeno con il quale si descrive lo swapping frequente delle pagine: porta ad alto uso del disco e basso uso CPU

FILE SYSTEM

Risiedono sulla memoria secondaria

- mappa memoria virtuale alla memoria fisica
- permette accesso e modifica di file

Alcuni concetti chiave

FILE:

contiene chunk di dati, può avere o non avere estensioni

DIRECTORY:

collezione di file e altre directory

MOUNT POINT:

directory che tiene il file system di un dispositivo

LINK:

puntatore in un file del sistema, che può essere

- logico
- fisico

ES File system Linux

/	- root folder directory
/proc	- filesystem che contiene processi e le sierne
/sys	- contiene informazioni riguardo l'hardware di system
/dev	- filesystem che contiene file fasulli con ls si può verificare: [-c : file a caratteri] [-b : file a blocchi]

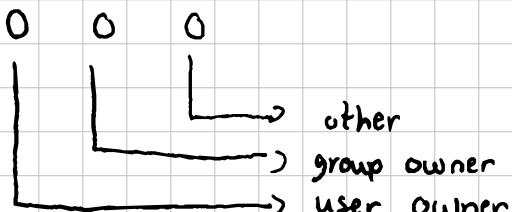
OWNERSHIP

Si usa una base ottale

- 4 read
- 2 write
- 1 execute

gli stessi permessi funzionano anche sulle directory.

- EXECUTE serve a fare il list.



c'è la possibilità di impostare permessi specifici per utenti con setfacl (Advanced Control list)

LINK

(crea link :

- > per link simbolico
- omesso per link hard.

OPERAZIONI SU FILE | FILE DESCRIPTOR

create

|
| intero che caratterizza un file
in un processo.

open

|
| più descrizioni possono portare
allo stesso file.

close

append

seek

:

allocati di default

• 0 stdout

• 1 stdin

• 2 stderr.

DEVICES

dispositivi di I/O. Ci sono gli hooks ai vari d.spositivi fisici nella cartella /dev.

I dispositivi sono v.sti come file, specialmente d. tipo:
-block (supporto d. seek e block read)
-character (sono d. tipo stream - es. mouse -)

È possibile anche leggere/scrivere su di essi!

> cat /dev/ttymACM0
> cat file > /dev/ttymUSB0

Necessità di meccanismi per accordarsi su come scambiarsi i dati!

IODELAY & TERMIOS

Sappendo che i dispositivi sono file, accessibili liberamente, potremmo volerli configurare (es. bandrate, accesso a funzioni specifiche...)

Esiste SYSTEM CALL IODELAY(), che è driver hook che prende in input request come int.

Nel caso dei dispositivi a carattere (es. Serial) è possibile utilizzare TERMIOS nello specifico

IMPLEMENTAZIONE FILESYSTEM

Considero il disco come un grande file binario, che può essere acceduto a blocchi di 4092 bytes.

↳ Mappatura dei file, folder.

Costruzione di struttura dati

Il sistema operativo deve aggiornarsi a hook del filesystem per interfacciarsi attraverso funzioni file/cartelle di apertura, chiusura, cancellazione etc.

comando mount e file fstab gestiscono i mounting dei dischi con varie formattazioni, mostrandole all'utente e al resto del sistema in maniera uniforme.

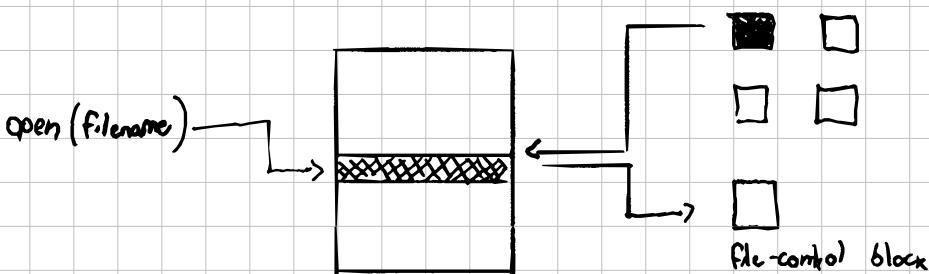
Strutture dati

Kernel

- Per ogni file aperto apri un'istanza di OpenFileInfo.
- Ad ogni reference aperta (apertura di File Descriptor) si apre istanza OpenFileRef, che punta all'istanza univoca OpenFileInfo, mantenendo info su open, write, seek.

Dischi

- Ogni file è caratterizzato da una struttura dati su disco, FileControl Block (FCB)
- ogni directory ha un header aggiuntivo di estensione
- sia i file che le directory possono essere contenuti in più blocchi, attraverso puntatore



FAT (File Allocation Table)

E' un array/list situata sui primi blocchi del disco.

L'array list è molto piccola, ed è INIZIALE.

DIRECTORY @ contiene variabili RECORD di dimensione fisso, che contiene l'indicazione di dove si trova un file e la sua relativa dimensione.

Vedi dopo per più dettaglio

INDEXED NODE FS

I blocchi sono file che contengono:

- INDICI
- DATI

File Control Block contiene metadati, e anche pochi dati.

In caso d. necessità il inode chiedrà d. allocare un set di blocchi, max 10.

Si crea struttura ad albero

DIRECTORY

Array (sciocco) che punta ai blocchi di dati.
Esecuzione costosa.

HashTable potrebbe essere buono, ma può avere collisioni!
buono se file hanno lunghezza fissa.

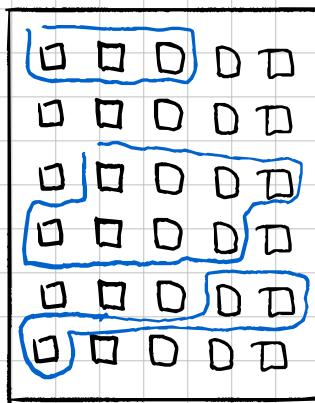
Tipi di allocazione

-contigua

-ogni file ha associato un puntatore d'inizio ed una lunghezza

problemi con la ricerca dello spazio,
potenziale frammentazione esterna.

Sarà comunque
(de)commentazione)



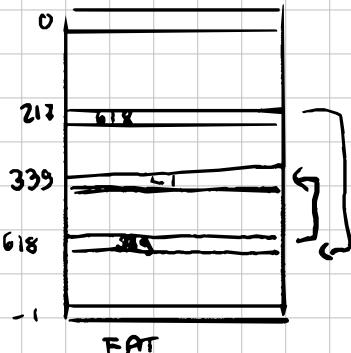
-linked

file ha un blocco d'inizio e di fine,
e ogn. blocco ha informazioni sufficienti
per prossimo blocco.

→ se blocchi non sono contigui c'è sarà frequente
spostamento della testina di lettura; perdita di
performance

directory entry

test 1	...	217
name	start	



INDEXED

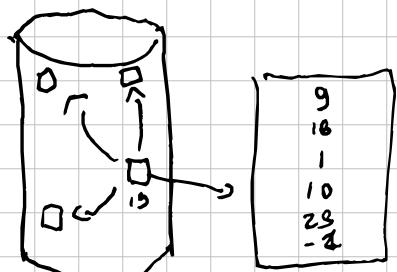
I blocchi nel disco possono essere

di 2 TIPI

- **INDIRECT:** contengono array di indici d. i blocchi dati contingui
- **DIRECT:** contengono dati.

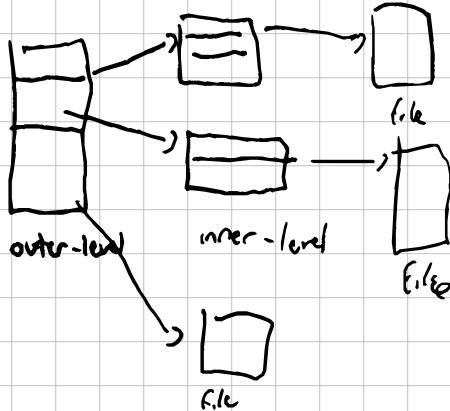
directory

file	index
keep	19



Level 0

- puntatore a blocchi:
di indici d. livello 1
- livello 1
- puntatori a blocchi dati



FREE SPACE MANAGEMENT

Utilizzo bitmap all'inizio del disco, con ogni bit che rappresenta un blocco e se è libero.

Oppure uso linked list di blocchi liberi (come Slab Allocator)

INTERPROCESS COMMUNICATION

Nato dalla necessità d. più process, d. comunicare tra loro. Due modi:

- memoria condivisa
- message passing

Message Passing

Più facile della shared memory: 3 operazioni:

- SEND (to mailbox / to user)
- RECEIVE (...)

Comunicazione può essere

- SYNCHRONIZED
- ASYNCHRONIZED
- DI RETTA
- PER MAIL BOX

POSIX MESSAGE QUEUES

`mq-[operazione]()`

l'utilizzo è molto simile a lettura scrittura di file descriptor!

es

`mq_receive()`

`mq_send()`

UTILIZZO DELLE CODE

Struttura dati usata dai message queue
Tutte le code sono organizzate nel files

ls /dev/mqueue

con cat <nome-coda> vedo lo stato delle code.

Le code sono inizializzate in kernel mode

SHARED MEMORY

mmaped, la shared memory condivide un area di memoria
condivisa visibile da processi

-ftruncate per scegliere dimensioni

Sarà poi compito di mmap di mappare il descrittore in memoria,
guardando le shmem aperte in /dev/shm

Oltre a posix si possono usare quelle systemV

DEVICES

Computer possono interagire con più dispositivi:

- storage device
- transmission devices
- generic io devices

Come ci si interagisce? FILE, riglore allineamento tra API

Direct Memory Access, per evitare di rompere alla CPU. CPU garantisce area di memoria, e DMA invia interrupt a trasferimento terminato

```
int fd = open("/dev/input/js0", O_RDONLY);
```

ad esempio, il joystick è event-based!

es. bottone premuto

levetta azione

:

```
struct js_event event;
```

dentro vi c'è la dimensione del joystick
& dei suoi eventi

```
read(fd, &event, sizeof(struct js_event))
```

SERIALE

Sempre un file, ma dev'essere configurato!

- ioctl manipola parametri d. un device considerando il suo descrittore.

Ogni tipo di device avrà tipo d. ioctl sua.

Vener implementato un livello di astrazione sopra l'ioctl più ad alto livello. **TERMIOS**

Termios è la stessa interfaccia della shell!

Una volta impostati i parametri ci si può interfacciare con read e write.

TERMIOS è una struct

AVR

Classi di CPU ad 8 bit fatti da ATMEL usato da microcontrollori, come Arduino.

Con registri "a shift", ovvero de un bit d'un registro serve ad attivare o disattivare funzioni, e configuralo

AVR:

- 8-bit
- architettura RISC
- viene programmata con C/C++

Esiste flavour di C++ più adatto per programmazione first step.
AVRDUE è flasher

Alcune periferiche AVR

- serial (USART, I2C)
 - AD converter
 - timers
 - power
- } controllati con shift registers

DIGITAL PIN

- memory mapped pin da A-
- pin specifico definito da lettera + numero

DDRX: data direction port (write)

- 1 = pin setupato come output
- 0 = pin setupato come input

PORTX: output port (write)

- 1: se pin è output, fa uscire +5V. Se pin è input attiva il pull-up resistor
- 0: porta output porta 0v

PINX: input port (read)

- 1 se input pin riceve 5v
- 0 se input pin riceve 0v

TIMERS

Utile strumento che dà la possibilità di mantenere un timer (F.n. a 5). Usato per:

- interrupt
- pwm generators

Un timer incrementa un counter $TCNT<X>\{L|H\}$ per ogni clock della CPU, oppure ogni potenza di 2 attraverso il prescaling.

Il counter verrà poi confrontato con $OCR<X>\{L|H\}$ (output compare register) e se gli stessi trigger dr. azione.

Quello che succede al raggiungimento dell'OCR dipende da come è settato $TCCR<X>\{L|H\}$, ad esempio per settare il prescaler.

Interrupt timers

attraverso il registro TIMSK è possibile attivare un interrupt del relativo timer utilizzato.

La routine che verrà compiuta sarà quella descritta in ISR (corrente). È importante mantenere leggera questa routine (es. cambio variabile).

I flag interrupt saranno poi settabili con

- cli() clear interrupts
- sei() set interrupts

Interrupt possono essere anche regolati da eventi esterni:

- INT<>X>: triggered at specific pin, ISR can handle single pins and can set its accuracy
- PCINT<>X>: triggered on pin change. ISR gestisce più pin.