***Memoria***

**Indirizzi** Ogni processo deve avere mem sufficiente per il proprio codice e dati, SO astrae / virtualizza la mem dando a ogni processo il proprio VAS e l'impressione di avere mem tutta per sé.

**Traduzione degli indirizzi** per mappare gli VAS a PAS (comporta isolazione tra processi) Semplifica la multiprogrammazione 🡪difficili che due programmi utilizzino indirizzi di mem differenti), Protezione processi e dati 🡪 tramite isolamento dei processi, Nasconde limitazioni fisiche 🡪 mem fisica differente per ogni PC

**Spazio Indirizzo virtuale** 🡪 dipendente dall’architettura es 32bit 🡪 2^32 virtual addr space **(anche TDP)**

**Virtual pages** 🡪 Ogni spazio di indirizzamento viene diviso in parti più piccole (non dipende da memoria fisica disponibile)

**Segmentazione 🡪**Data (var inizializzate), Block Started by Symbol ( var non inizializzate), Heap ( dati dinamici), Stack

**segmentation fault** 🡪Quando un processo accede a mem fuori dal segmento ( dichiarato)

**Principio di località 🡪** utilizzo di nemmeno un segmento di memoria in contemporanea

**Memory Paging** 🡪 suddivisione della mem in parti indipendenti più piccole. Pagine di differenti processi (quindi differenti VAS) caricate su ram, la condivisione tramite DLL o Copy-on-Write(COW)

**Traduzione indirizzi**🡪 meccanismo di traduzione(**HW**) tra Indirizzi virtuali(VAS) e Ram (PAS). Per ogni processo viene utilizzato il memory page table che associa numero di pagina a numero frame.

**Mappatura** 🡪 Quando viene allocata mem, controllo se si adatta alle pagine esistenti, altrimenti si **mappa** una nuova. Mapping significa che la nuova mappa è valida. Quando la pagina non è più necessaria si fa **unmap** (invalida la pagina)

**Malloc/Free** 🡪 estende heap (nuove pagine) mentre free (elimina pagine). (ma non alloca)

**Frammentazione** 🡪 quando vengono creati buchi nelle pagine continue

La **memoria può essere condivisa** esplicitamente tra processi. L'idea è di creare un oggetto memoria condivisa e successivamente mappare questa area nello VAS di ogni processo partecipante (POSIX)

**I semafori POSIX** 🡪per sincronizzare l'accesso alla memoria condivisa

**Memoria condivisa**🡪 utilizzare offset da inizio struttura. Se una lista utilizzare gli indici. Non affidarsi a puntatori poiché l’oggetto può trovarsi in più VAS

**Memory Layout** 🡪 Ci sono aree fatte di pagine accessibili solo dal kernel

**Page Number /Offset** 🡪 VAS dividendo il valore binario dell'indirizzo virtuale in due (+ sign PN – sign Offset)

**Frame/Offset** 🡪 copia offset di VAS e contenuto del PN in frame

**MMU (Memory Management Unit)🡪** esegue algoritmo Page Table **Walker** che analizza la tabella delle pagine per cercare una **valid page translation** 🡪 se c’è indica che può avvenire altrimenti **invalid page fault**

**V bit 🡪** valido non valido **P bit-- >** mappato su un frame **M bit 🡪** 1 se pagina modificata (non da MMU) **R bit 🡪** 1 se pagine referenziata (non da TLB)

**Swap out** 🡪 quando RAM piena si spostano le pagine meno utilizzate in HDD

**Swap in** 🡪 quando richiesta informazione su una pagina specifica viene riportata su RAM

**Windows** 🡪 usa un file su disco (pagefile.sys) **Linux** 🡪usa una partizione dedicata (ma può anche su file)

**Invalid Page Fault** 🡪 non esiste voce valida per tradurre VAS (utente non ha allocato mem🡪segm fault)

**Major Page Fault** 🡪 c'è una voce ma la pagina richiesta, non si trova nella memoria principale (necessario eseguire lo swap in)

**Minor Page Fault** 🡪 Pagina già in **memoria** ma bit P (present) non impostato (contenuto già in RAM)

**TLB (Translation Lookaside Buffer) 🡪** piccola cache per contenere una parte di page table (non traduce VAS)

**TLB hit 🡪** se TLB può tradurlo **TLB miss 🡪** se dobbiamo cercare in tutta la tabella

**TLB flush** 🡪 voci mem nella cache sono legate a ciascun P, il SO deve delete il buffer su un context-switch

**Multi-level page tables** 🡪 richiedono meno spazio, per ogni tabella si hanno tabelle secondari

***Page replacement algorithms***

**First Fit** 🡪 il primo spazio vuoto che soddisfa la richiesta di allocazione **Next Fit** 🡪 come il primo solo che non riparte da zero **Best Fit** 🡪 il più piccolo che soddisfa la richiesta **Worst Fit** 🡪 il più grande che soddisfa la richiesta **Quick Fit** 🡪 mantiene separate le allocazioni, e ritorna il primo vicino

**Page Replacement** 🡪 Algoritmo che sceglie come liberare la memoria se verificatosi un errore

**Global Page Replacement** 🡪 sostituisce tutte le tabelle della pagina **Local Page Replacement** 🡪 solo del processo che ha generato l’errore

**NRU** 🡪 algoritmo che setta a R(referenza)=W(modificato)=0 quando allocato e sceglie la classe più bassa quando deve sostituire una pagina : Classe0🡪 R=W= 0; Classe1🡪R=0; W=1; Classe2🡪R=1;W=0; Classe3🡪R=W=1

**Dirty Pages** 🡪 Se M=0 possiamo rimuoverli senza problemi mentre se modificati (M=1) dobbiamo salvarli prima su disco (operazione molto costosa)

**FIFO** 🡪 La prima pagina caricata è la prima a essere rimossa **Second Change** 🡪 Come fifo ma se bit R = 1 lo mettiamo a 0 e viene posizionato sul retro. Se 0 si elimina **The Clock Algorithm** 🡪 simile a Second change ma richiede pochi puntatori

**Least Recently Used (LRU)** 🡪 sfrattare le pagine che non sono state usate di recente

* Linked list (head + usate, tails – usate) 🡪modificare la lista ad ogni accesso
* Timestamp 🡪 cambiare il timestamp 🡪 molto costoso
* Contatore HW di accessi 🡪 nella tabella delle pagine viene settato, il minore viene cancellato
* Matrice🡪riga viene settata a 1 e colonna a 0, la riga con il valore più basso è il meno utilizzato

**Not Frequently Used (NFU)** 🡪 software ogni pagina settata a 0, periodicamente il valore di R viene modificato di +0 o +1 il valore più basso viene eliminato

**Aggressive swapping**🡪 memoria fisica non sufficiente e molti errori di pagina comporta uso HDD (**trashing)**

**Simulazione segmenti 🡪** è possibile definire diverse aree di memoria che simulano. La dimensione delle aree è arrotondata alle dimensioni della pagina .L'heap e lo stack possono crescere allocando nuove pagine**.**

**External Fragmentation** 🡪 Perché i segmenti hanno lunghezza variabile

**Internal Fragmentation** 🡪Poiche i segmenti hanno lunghezza fissa

**Paging** 🡪 in AMD64 non è più disponibile la segmentazione inoltre viene esteso il No Execute (NX bit)

**NX bit** 🡪 meccanismi assistiti dall’HW per proteggere la memoria (ad esempio stack)

**Address Space Layout Randomization (ASLR)** 🡪per impedire a un utente malintenzionato di passare in modo affidabile a una particolare funzione sfruttata in memoria:

**memoria nel Real Time** 🡪 velocità di allocazione ed evitare la frammentazione

**Frammentazione** Liberazione di una parte di memoria (richiede tempo e non deterministico) non sembre Garbage collector. Per evitarla occorre tenere traccia dei blocchi liberi e allocarli se disponibili

***Virtualizzazione***

**System Level Virtualization** 🡪 Emula un architettura completa (es VirtualBox)

**Application Level Virtualization🡪** implementa una nuova architettura nel software, per consentire il software portatile (JVM)

**Operating System Level Virtualization**🡪 astrazione del sistema operativo che consente l'isolamento e la creazione di "sottosistemi" indipendenti

**Vantaggi** 🡪 **+ sistemi operativi,** **Isolamento** 🡪 sistemi con requisiti differenti, sicurezza **consolidamento**🡪molti servizi sulla stessa macchina, uso più efficiente delle risorse, minori costi, **on-deman** 🡪allocazione dinamica, bilancio del carico

**Proprietà**🡪 **Efficienza**🡪Istruzioni innocue eseguite dall HW, **Resource Control**🡪 Impossibile influenzare le risorse del sistema **Equivalenza**🡪 Poter eseguire tutte le istruzioni se si avesse un vero OS

**Hypervisor(HV)/Virtual Machine Monitor(VMM)🡪** software che gestisce l’accesso HW

**Bare metal (tipo 1)** 🡪 HV direttamente sull HW, non esiste un SO host (VMWare ESx)

**Hosted (tipo 2)** 🡪 HV lavora sull SO, i guest sono gestiti dal HV

**Full Virtualization** 🡪 Non sa che è un VM, alcune operazioni in privilegiata (kernel mode) in x86 non è stata progettata per questo, Alcune istruzioni privilegiate non funzionano quando vengono invocate da un contesto non privilegiato (errore non gestibile)

**Full software emulation (bochs)🡪** ricompila/emula ogni istruzione (lento)

**Paravirtualization(Xen)** 🡪 eseguite in privilegiato, buone prestazioni per **driver**, utilizza SO modificati

**Dynamic translation(VMware)** 🡪 Rileva quando il codice passa in modalità privilegiata, analizza le istruzioni e riscrive le istruzioni privilegiate (**trap and emulate**)

**Trap and emulate** 🡪 può rifiutare di eseguire operazione, accesso alcuni **registri CPU e strutture di memoria** problematiche, accesso ad HW controllato da VMM

VMM può gestire una **shadow copy** di registri "privilegiati" per fornire all'ospite una visione coerente del sistema virtualizzato

**Memory tracing** 🡪 Ogni accesso alla memoria può potenzialmente leggere o modificare strutture di dati privilegiate. memoria privilegiate (sola lettura) e gestiscile eccezioni causate da scrittura

**Exploit page faults**🡪 se la pagina non à mappata, VMM gestisce errore, se manca anche li errore SO

**Shadow page table**🡪 VMM( privilegiata) usa SPT che mappa gli indirizzi virtuali generati all'interno del guest in indirizzi fisici. Quando viene rimossa una pagina il sistema esegue anche flush TLB.

**Supporto HW🡪** Le CPU recenti implementano il supporto hardware per la virtualizzazione e può generare trap per istruzioni privilegiate (a evitare la necessità di frequenti traduzioni dinamiche)

**IOMMU** 🡪MMU per dispositivi I / O che offre una traduzione degli indirizzi simile alla MMU ma per i dispositivi I / O

***Storage Devices***

Un'unità disco è composta da un set di piatti (**platters**) che sono rilegati attorno a un **spindle**, disco ricoperto con uno strato magnetico che può conservare in modo permanente i bit. Velocità costante. Su un disco non possiamo indirizzare singoli bit. **Numero settori non è sempre lo stesso**

**Cilindro** (set di tutte le tracce su ciascun lato di ogni piatto) **Traccia**🡪 circonferenza di un disco **Settore**🡪Circonferenza dentro ad un triangolo (unità di memoria più piccola) **Cluster**🡪 set di settori contigui **Preamble**🡪 Inizio di un settore **2 Head**🡪 Disco

Rotation Delay🡪 nel peggiore dei casi dobbiamo aspettare una rotazione completa

**Seek**🡪 la ricerca impiega un po' di tempo (**seek time**) **Settling time**🡪 tempo prima che possiamo accedere alla nuova traccia

**acceleration time** 🡪 la testa inizia a muoversi **coasting time** 🡪testa si muove **deceleration time** 🡪la testa rallenta sulla traccia target

**Interleaving (track skew)🡪** i settori su diverse tracce sono "non allineati" (di proposito) (non aiuta seek)

**Disk scheduler** 🡪 l’ordine delle operazioni I/O (inutile per SSD)

**Elevator (SCAN)🡪** elabora le richieste durante la scansione del disco

**F-SCAN**: ritardare le richieste in arrivo mentre si scorre sul disco

**C-SCAN** (Circular SCAN): viaggiare solo in una direzione (saltare indietro quando si raggiunge un bordo)

**LOOK**: invertire la direzione se non ci sono richieste in anticipo nella direzione corrente

**SSD** sono costituite da una serie di celle di memoria, Non hanno alcuna parte meccanica (non c’è ritardo)

La tensione applicata al gate d'ingresso controlla la corrente che scorre tra la sorgente e lo scarico (3 pins: source , drain , gate) e lo stato si determina tramite threadshold

**SLC (Single Level Cell)** 🡪 2 stati **MLC(Multi)🡪**4 stati **TLC(Triple)🡪**8 stati

**Erase-before-write🡪** cioè la memoria flash deve essere cancellata prima che possa essere riscritta

**wear out 🡪** ogni cella può essere cancellata solo un numero limitato di volte prima che si guada, si esaurisca (influenzato da filesystem, no TRIM)

**wear leveling🡪**evitare di scrivere sugli stessi blocchi più e più volte

**TRIM** 🡪non posso aumentare lo spazio libero e non obbliga il garbage collector

***Filesystem***

**Physical Block🡪** l'unità di archiviazione di base supportata dal dispositivo

**Logical Block🡪** è la più piccola unità di archiviazione supportata dal filesystem

**Filesystem** implementa e fornisce accesso a un astrazione dei dispositivi a blocchi come una struttura di file e directory (free space (linkedlist/bitmap), allocated space, accedere ai dati, percorsi(ass/rel), chiamate sistema(r/w/x), veloce

**Contiguous allocation**🡪facile, efficiente in lettura, frammentato)

**LinkedList**🡪supporta blocchi non contigui, contiene ptr, lento, non affidabile

**Fat**🡪Facile,veloce, tabella può diventare grande

**Index-Node**🡪veloce, supporta grandi file

**Layout**🡪super block(informazioni filesystem), free space management(bitmap), inodes

**Scenari**

dataBlock scritto (dati inaccessibili, blocco libero)🡪inode aggiornato (dati accessibili, blocco libero)🡪bitmap aggiornato

inode aggiornato (inconsistenza dati)🡪dataBlock scritto (dati accessibili, blocco libero)🡪bitmap aggiornato

bitmap aggiornato(leak,perdita spazio)🡪inode aggiornato (non sappiamo quale file appartiene🡪dataBlock scritto

inode aggiornato (inconsistenza dati) 🡪bitmap aggiornato (dati vecchi/inutili) 🡪dataBlock scritto

**Journaling**🡪registro con tutte le modifiche

**Journaling Fisico** 🡪 ogni aggiornamento sul disco viene registrato

**Journaling Logico** 🡪 registrate modifiche (inode,bitmap)

**Ext3**🡪salvataggio prima delle modifiche

**Writeback**🡪 inode, bitmap, directory sono registrati, dataBlock no→ veloce, non consistente

**Ordered** 🡪 inode, bitmap, directory sono registrati, datablock old cancellato→ lento, non consistente

**Full data journaling**🡪tutto viene registrato🡪lento, consistente