Q: Quando il prepaging è vantaggioso?

A: Quando α (percentuale di pagine utilizzate tra quelle precaricate) è sufficientemente alta, così che il costo dei page fault risparmiati superi il costo delle pagine inutilmente caricate.

Q: Quali sono i pro e i contro di usare pagine grandi o piccole in memoria virtuale?

A: Pagine piccole: meno frammentazione interna, ma tabelle più grandi. Pagine grandi: meno accessi I/O e meno page fault, ma più frammentazione interna.

Q: Cos'è la TLB Reach e perché è importante?

A: È la quantità di memoria coperta dalla TLB: TLB Reach = dimensione TLB × dimensione pagina. Se non copre l'insieme di lavoro del processo, aumentano i page fault.

Q: Come influisce la struttura del programma sulla gestione della memoria virtuale?

A: Esempio: azzerare una matrice per righe genera molti più page fault rispetto a farlo per colonne, se ogni riga occupa una pagina.

Q: Cos'è il pinning di una pagina in memoria?

A: È il blocco di una pagina durante l'I/O per evitare che venga rimossa dal sistema prima della fine dell’operazione.

Q: Cos'è la gestione della memoria in un sistema operativo?

A: Allocare, monitorare e liberare la memoria in modo sicuro ed efficiente tra i processi, garantendo isolamento e massimizzazione dell'utilizzo delle risorse.

Q: Cos'è la MMU e quale ruolo svolge?

A: La MMU (Memory Management Unit) è un dispositivo hardware che converte gli indirizzi logici in indirizzi fisici durante l'esecuzione, usando ad esempio il registro di rilocazione.

Q: In cosa consiste il caricamento dinamico?

A: È la tecnica che carica in RAM solo le parti del programma al momento in cui vengono utilizzate, risparmiando spazio e migliorando l'efficienza della memoria.

Q: Cos'è il collegamento dinamico?

A: È un meccanismo in cui il collegamento alle librerie avviene a tempo di esecuzione, usando uno stub che carica e collega la routine solo quando serve.

Q: Cos'è l'allocazione contigua della memoria?

A: È una tecnica in cui ogni processo occupa una singola partizione continua della memoria fisica. È semplice ma soggetta a frammentazione esterna.

Q: Cosa si intende per allocazione a partizioni variabili?

A: È un metodo che adatta dinamicamente la dimensione delle partizioni alle esigenze dei processi, riducendo la frammentazione interna ma non quella esterna.

Q: Quali sono gli algoritmi di allocazione più comuni?

A: Best-fit, First-fit e Worst-fit. I primi due sono generalmente migliori in termini di utilizzo dello spazio e velocità.

Q: Come si può ridurre la frammentazione esterna?

A: Attraverso la compattazione, che consiste nel rimescolare la memoria per ottenere un unico blocco libero contiguo.

Q: Cos'è la paginazione nella gestione della memoria?

A: È una tecnica che divide la memoria logica e fisica in blocchi di dimensione fissa (pagine e frame) per eliminare la frammentazione esterna.

Q: Come si ottiene l'indirizzo fisico a partire da f e d nella paginazione?

A: I bit meno significativi dell'indirizzo fisico coincidono con l'offset d, mentre i più significativi con l'indice del frame f trovato nella page table.

Q: Cos'è la frammentazione interna nella paginazione?

A: È lo spazio inutilizzato nell'ultima pagina allocata a un processo. In media, ammonta a mezzo frame.

Q: Cosa contiene una riga della page table oltre al numero del frame?

A: Bit di validità, protezione, modifica, riferimento, e cache disabled.

Q: Quali registri gestiscono la page table nella CPU?

A: Il Page Table Base Register (PTBR) e il Page Table Length Register (PTLR).

Q: A cosa serve la TLB?

A: La TLB è una cache veloce nella CPU che memorizza traduzioni recenti da indirizzi logici a fisici, riducendo i tempi di accesso.

Q: Cos'è un TLB hit e un TLB miss?

A: Un TLB hit avviene quando la pagina cercata è nella TLB, un miss quando non lo è e si deve accedere alla page table.

Q: Cos'è l'EAT (Effective Access Time)?

A: È il tempo medio di accesso alla memoria, considerando il tasso di hit nel TLB e il tempo di accesso in caso di miss.

Q: Come viene implementata la protezione della memoria nella paginazione?

A: Attraverso bit di protezione associati a ciascun frame e bit di validità nelle voci della page table.

Q: Cosa sono le pagine condivise?

A: Pagine mappate sullo stesso frame per più processi, usate per codice rientrante e condivisibile (es. librerie).

Q: Come si struttura una page table gerarchica?

A: È una struttura ad albero: un indirizzo logico è diviso in p1 (outer), p2 (inner) e offset. Si passa dalla outer table alla inner table per trovare il frame.

Q: Perché si usano più livelli di paginazione?

A: Per gestire grandi spazi di indirizzi logici evitando tabelle delle pagine enormi, suddividendole in più livelli (2, 3 o più).

Q: Cos'è una tabella delle pagine hash?

A: Una struttura in cui il numero di pagina virtuale è usato come chiave di hash per accedere rapidamente alla corrispondente entry con numero di frame.

Q: A cosa servono le page table clustered?

A: A mappare più pagine (es. 16) per ogni voce, migliorando l'efficienza con spazi di indirizzi sparsi.

Q: Cos'è una tabella delle pagine invertita?

A: Una tabella con una voce per ogni frame fisico, che contiene il numero di pagina virtuale e il processo proprietario.

Q: Qual è il vantaggio della tabella delle pagine invertita?

A: Riduce l'uso di memoria per le tabelle delle pagine, ma richiede una ricerca più lenta (spesso hash).

Q: Cosa contiene la tabella delle pagine?

A: Contiene le corrispondenze tra pagine virtuali e frame fisici. Serve a tradurre gli indirizzi logici in indirizzi fisici durante l'accesso alla memoria.

Q: Come si struttura un indirizzo virtuale nella paginazione?

A: È diviso in due parti: il numero di pagina (page number) e l'offset all'interno della pagina.

Q: Cos'è il resident set?

A: È l'insieme delle pagine di memoria virtuale di un processo attualmente caricate in RAM.

Q: Perché la RAM è necessaria per l'esecuzione di un processo?

A: Perché la CPU può accedere solo alla RAM e ai registri, quindi il programma deve essere caricato lì per essere eseguito.

Q: Qual è il ruolo della cache nella gestione della memoria?

A: La cache si trova tra CPU e RAM e serve a ridurre la latenza di accesso, velocizzando l'esecuzione.

Q: Qual è la funzione dei registri base e limite?

A: Proteggere la memoria: il registro base indica l'inizio dell'area di memoria di un processo, il limite indica la dimensione dell'area.

Q: Cosa accade se due processi sono caricati all'indirizzo 0?

A: Non possono convivere senza rilocazione: uno dei due deve essere spostato, e i suoi riferimenti aggiornati.

Q: A cosa serve la rilocazione automatica?

A: A permettere che i programmi credano di essere all'indirizzo 0, ma vengano eseguiti da un'altra parte della memoria.

Q: Quali sono le fasi di binding degli indirizzi?

A: A tempo di compilazione, a tempo di caricamento e a tempo di esecuzione.

Q: Cos'è l'indirizzo logico?

A: È l'indirizzo generato dalla CPU, visibile al programma. È anche detto indirizzo virtuale.

Q: Cos'è l'indirizzo fisico?

A: È l'indirizzo effettivo usato dalla RAM, calcolato dalla MMU in fase di esecuzione.

Q: In quali casi gli indirizzi logici e fisici coincidono?

A: Quando il binding avviene a tempo di compilazione o caricamento.

Q: Cos'è la MMU (Memory Management Unit)?

A: È l'unità hardware che traduce gli indirizzi logici in indirizzi fisici durante l'esecuzione del programma.

Q: Cos'è la memoria virtuale e perché è importante?

A: Permette l'esecuzione di processi più grandi della memoria fisica, isolamento tra processi e allocazione non contigua, migliorando flessibilità e sicurezza.

Q: Cos'è la paginazione (paging)?

A: Divide la memoria virtuale in pagine e la memoria fisica in frame di dimensione fissa. Mappa le pagine ai frame tramite una tabella delle pagine, evitando frammentazione esterna.

Q: Qual è la differenza tra indirizzo logico e indirizzo fisico?

A: L'indirizzo logico è generato dal processo (CPU), mentre l'indirizzo fisico è quello reale in RAM. La MMU esegue la traduzione.

Q: A cosa serve la tabella delle pagine?

A: Tiene traccia della corrispondenza tra pagine logiche e frame fisici, includendo anche bit di controllo (presenza, modifica, protezione, ecc.).

Q: Cosa significa "demand paging"?

A: Le pagine sono caricate in RAM solo quando richieste, riducendo l'uso iniziale della memoria e accelerando l'avvio dei processi.

Q: Quando si verifica un page fault?

A: Quando un processo accede a una pagina non presente in RAM. Il sistema operativo interrompe il processo, carica la pagina da disco, aggiorna la tabella e riprende l'esecuzione.

Q: Quali sono i principali algoritmi di sostituzione delle pagine?

A: FIFO: sostituisce la pagina più vecchia. LRU: sostituisce la meno recentemente usata. Clock: approssima LRU in modo efficiente.

Q: Cosa fa lo swapping in un sistema operativo?

A: Sposta interi processi tra memoria RAM e disco per liberare spazio, utile nei sistemi senza memoria virtuale o con poca RAM.

Q: Cos'è lo swapping?

A: Tecnica che sposta temporaneamente un processo dalla RAM al backing store (disco) per liberare memoria fisica.

Q: Qual è il tempo tipico di uno swap in/out?

A: Per 100MB su disco a 50MB/s: 2s per lo swap out, 2s per lo swap in, totale 4s per lo switch di contesto.

Q: Cos'è il double buffering nello swapping?

A: Tecnica che copia prima i dati in memoria kernel, poi permette lo swap, infine trasferisce all'I/O. Utile quando c'è I/O sospeso.

Q: Quando non è possibile fare swap-out di un processo?

A: Quando è in stato di wait per un'I/O in corso, poiché l'I/O verrebbe eseguito su un processo sbagliato.

Q: Cos'è lo swapping with paging?

A: Tecnica che swap-out solo alcune pagine del processo, generalmente quelle non attive, riducendo il tempo di swap.

Q: Come si gestiscono i blocchi liberi nella memoria fisica?

A: Bitmap: array di bit, 1 = occupato, 0 = libero. Elenchi: liste di blocchi liberi con indirizzo e dimensione.

Q: Quali tecniche usa il kernel per gestire dinamicamente la memoria?

A: Buddy System: divide la memoria in potenze di 2. Slab Allocator: prealloca oggetti dello stesso tipo per evitare frammentazione.

Q: Cos'è la frammentazione e che tipi esistono?

A: Interna: spazio inutilizzato dentro i blocchi allocati. Esterna: spazio libero tra blocchi non contigui, non riutilizzabile efficacemente.

Q: Come si può ottimizzare la gestione della memoria?

A: Usando TLB, clustering di pagine, allocazione basata su priorità, e compressione delle pagine inattive.

Q: Cos'è lo slab allocator e a cosa serve?

A: È un sistema di allocazione della memoria usato nel kernel per gestire oggetti piccoli e frequenti in modo efficiente. Prealloca blocchi (slab) contenenti oggetti dello stesso tipo, pronti all'uso e riutilizzabili, riducendo la frammentazione e migliorando le prestazioni.

Q: In che modo la memoria virtuale aumenta efficienza e flessibilità?

A: Permette isolamento, protezione, multitasking efficiente, carico parziale dei programmi, e gestione ottimizzata delle risorse fisiche.

Q: Cos'è la memoria virtuale?

A: È la separazione tra indirizzi logici e fisici: permette che lo spazio di indirizzi di un processo sia più grande della RAM fisica disponibile.

Q: Quali vantaggi offre la memoria virtuale?

A: Spazio di indirizzamento più ampio, creazione processi più efficiente, più processi concorrenti, meno I/O per caricare/swappare.

Q: Cos'è il virtual address space?

A: È l'insieme degli indirizzi logici che un processo può usare, solitamente contiguo e separato dallo spazio fisico.

Q: Cos'è la paginazione su richiesta (demand paging)?

A: È una tecnica che carica le pagine in memoria solo quando sono effettivamente richieste da un processo.

Q: Cos'è un page fault?

A: Un errore che si verifica quando si tenta di accedere a una pagina non residente in memoria (bit valido = i nella page table).

Q: Quali sono i vantaggi della paginazione su richiesta?

A: Meno I/O, meno memoria necessaria, risposta più rapida e più processi supportati contemporaneamente.

Q: Cos'è la pura paginazione a richiesta?

A: È il caso estremo dove ogni accesso a pagina causa un page fault, poiché nessuna pagina è inizialmente residente in RAM.

Q: Cos'è la località di riferimento?

A: Concetto secondo cui un'istruzione può accedere a più pagine contigue, riducendo il numero complessivo di page fault.

Q: A cosa serve la lista dei frame liberi?

A: Permette di trovare rapidamente un frame libero da usare in caso di page fault, ottimizzando la gestione della memoria.

Q: Quali sono le fasi di gestione di un page fault?

A: 1. Trap, 2. Salvataggio stato, 3. Verifica errore, 4. Controllo legalità, 5. Lettura disco, 6. Aggiornamento tabelle, 7. Riavvio istruzione.

Q: Quali sono i principali schemi di allocazione dei frame?

A: Allocazione fissa e allocazione prioritaria.

Q: Cos'è l'allocazione proporzionale?

A: Frame assegnati in base alla dimensione dell'address space del processo.

Q: Qual è la differenza tra rimpiazzamento globale e locale?

A: Il rimpiazzamento globale sceglie frame da qualsiasi processo, quello locale solo dal processo corrente.

Q: Cos'è la strategia Reclaiming Pages?

A: Attiva il page replacement quando i frame liberi scendono sotto una soglia prestabilita.

Q: Cos'è l'accesso alla memoria non uniforme (NUMA)?

A: In architetture multicore la RAM è divisa in partizioni locali per ogni CPU, con tempi di accesso variabili.

Q: Cos'è il thrashing?

A: Condizione in cui il sistema passa più tempo a swappare pagine che a eseguire il codice.

Q: Cos'è il modello Working Set?

A: Modello che identifica l'insieme di pagine su cui un processo lavora attivamente entro una finestra temporale Δ.

Q: Come funziona il metodo Page Fault Frequency?

A: Aggiunge o rimuove frame a un processo in base alla distanza temporale tra due page fault (τ).

Q: Come viene gestita la memoria del kernel?

A: Con metodi specifici come allocazione contigua, Buddy System o Slab Allocator per evitare frammentazione.

Q: Come funziona il Buddy System?

A: Alloca memoria in potenze di 2; i blocchi possono essere divisi o uniti in "buddy" per ridurre frammentazione.

Q: Cos'è lo Slab Allocator?

A: Sistema che gestisce oggetti di dimensione fissa in slab per ridurre la frammentazione e velocizzare l'allocazione.

Q: Cos'è il prepaging?

A: Tecnica che pre-carica alcune pagine prima che vengano referenziate, per ridurre i page fault all'avvio.

Q: Cosa succede dopo che si riceve un interrupt di I/O completato durante un page fault?

A: Il sistema corregge le tabelle, attende che la CPU sia riassegnata, ripristina lo stato del processo e riprende l'esecuzione.

Q: Cos'è l'EAT (Effective Access Time) in presenza di page fault?

A: EAT = (1 – p) x tempo accesso memoria + p x (overhead page fault + swap out + swap in)

Q: Cos'è la Copy-on-Write (COW)?

A: Una tecnica in cui padre e figlio condividono inizialmente pagine; quando una viene modificata, viene copiata solo allora.

Q: Quando si usa la tecnica 'zero-fill-on-demand'?

A: Quando si vogliono allocare frame liberi già azzerati, pronti all'uso, provenienti da un pool dedicato.

Q: Quando viene attivata la sostituzione delle pagine?

A: Quando non ci sono frame liberi, bisogna scegliere una pagina da rimpiazzare per fare spazio.

Q: Qual è il compito dell'algoritmo di sostituzione delle pagine?

A: Minimizzare i page fault scegliendo una pagina che difficilmente verrà riutilizzata a breve.

Q: Come funziona l'algoritmo FIFO per il page replacement?

A: Sostituisce la pagina da più tempo in memoria, con struttura a coda circolare.

Q: Cos'è l'anomalia di Belady?

A: Fenomeno per cui aggiungendo più frame con l'algoritmo FIFO si possono avere più page fault.

Q: Come funziona l'algoritmo ottimo di page replacement?

A: Sostituisce la pagina che non verrà usata per più tempo in futuro; è teorico e usato come benchmark.

Q: Come funziona l'algoritmo LRU?

A: Sostituisce la pagina meno recentemente usata, usando contatori o stack per tenere traccia.

Q: Cos'è l'algoritmo second-chance?

A: Versione approssimata di LRU che dà una seconda possibilità alle pagine con bit di riferimento a 1.

Q: Cos'è l'algoritmo Enhanced Second-Chance?

A: Usa sia il bit di riferimento sia il bit di modifica per rimpiazzare le pagine più 'leggere'.

Q: Differenza tra LFU e MFU?

A: LFU sostituisce la pagina usata meno frequentemente, MFU quella usata più frequentemente.

Q: Cos'è il page buffering?

A: Mantiene le pagine vittima in un pool per poterle riutilizzare rapidamente senza rilettura da disco.

Q: Cos'è il doppio buffering nelle applicazioni?

A: È quando sia il sistema operativo che l'applicazione mantengono una copia della pagina in memoria.

Q: Quali sono le componenti principali e i tempi caratteristici degli HDD?

A: Piatti rotanti, testine mobili. Tempo di posizionamento = seek + latenza rotazionale. Transfer rate lento (~1 Gbit/s). Access latency = tempo medio di accesso. Rischio di head crash.

Q: Come funziona lo scheduler di deadline in Linux per evitare la starvation?

A: Mantiene code separate per lettura e scrittura, assegna priorità alla lettura e gestisce richieste in batch. Usa 4 code: 2 LBA (tipo C-SCAN) e 2 FCFS, selezionando la più vecchia se supera i 500ms.

Q: Quali algoritmi di scheduling disco sono disponibili in RHEL 7?

A: CFQ (equità tra processi), NOOP (coda FIFO, nessun riordinamento, utile per SSD con controller autonomo).

Q: Perché gli SSD preferiscono accessi casuali e NOOP?

A: Non hanno testine meccaniche, quindi il seek non incide. Accessi casuali sono più efficienti e NOOP evita riordinamenti inutili.

Q: Cos'è il fenomeno della write amplification negli SSD?

A: È l'effetto per cui una singola scrittura può causare molte letture/scritture a causa della garbage collection, rallentando le prestazioni.

Q: Quali tecniche si usano per rilevare e correggere errori nei dischi?

A: Bit di parità, checksum, CRC (Cyclic Redundancy Check), ECC (Error Correction Code).

Q: Qual è la differenza tra errori "soft" e "hard" nei dischi?

A: Soft: temporanei e correggibili. Hard: fisici e permanenti, possono essere rilevati ma non sempre corretti.

Q: In cosa consiste la formattazione di un disco?

A: Formattazione fisica (creazione settori con ECC) e logica (creazione file system, strutture dati, cluster).

Q: Cosa succede durante il montaggio (mount) di una partizione?

A: Il sistema verifica il file system e, se corretto, lo aggancia al sistema operativo. La root viene montata all'avvio.

Q: Cos'è il boot block e qual è il suo ruolo?

A: Contiene il bootstrap loader, il codice che carica il kernel in RAM. Nei sistemi multi-boot può essere un boot manager.

Q: Cos'è l'accesso raw e perché è utile?

A: È l'accesso diretto al disco senza passare dal file system, usato ad esempio dai database per maggiore efficienza.

Q: Qual è il ruolo dello spazio di swap?

A: Estende la RAM sul disco, usato per swapping e paging. Può risiedere su partizioni raw o file. Linux usa bitmap o tabelle per gestirlo.

Q: Quali sono le tre architetture principali di collegamento dello storage?

A: Host-attached (es. SATA, USB), NAS (file system remoto via rete), SAN (accesso a blocchi via rete, es. Fibre Channel, iSCSI).

Q: Cos'è un RAID e quali sono i suoi obiettivi?

A: È un insieme di dischi per aumentare affidabilità, prestazioni e disponibilità continua attraverso ridondanza e accesso parallelo.

Q: Quali sono le principali metriche di affidabilità nei sistemi RAID?

A: MTTF (tempo medio al guasto), MTTR (tempo di riparazione), MTTDL (tempo medio alla perdita dati irreversibile).

Q: Perché la NVRAM viene usata nei sistemi RAID?

A: Agisce da buffer veloce e sicuro per scritture. I dati non si perdono in caso di interruzione di corrente.

Q: Cosa caratterizza RAID 0 (striping)?

A: Distribuisce i dati in blocchi su più dischi. Migliora le prestazioni ma non offre ridondanza. Il guasto di un solo disco causa la perdita totale dei dati.

Q: Cosa caratterizza RAID 1 (mirroring)?

A: Ogni disco ha un duplicato esatto. Fornisce alta affidabilità, ma dimezza la capacità effettiva.

Q: Cosa caratterizza RAID 4?

A: Usa striping con un disco di parità dedicato. Buono per letture sequenziali, ma inefficiente per scritture casuali.

Q: Cosa caratterizza RAID 5?

A: Usa striping con parità distribuita. Buon compromesso tra spazio, prestazioni e affidabilità. Tollera il guasto di un disco.

Q: Cosa caratterizza RAID 6?

A: Come RAID 5, ma con doppia parità. Tollera il guasto di due dischi contemporaneamente.

Q: Cosa caratterizza RAID 0+1 e RAID 1+0 (RAID 10)?

A: Combinano mirroring e striping. Offrono alte prestazioni e ridondanza, richiedono almeno quattro dischi.

Q: Cosa sono i dischi hot spare nei RAID?

A: Dischi inattivi pronti a sostituire automaticamente un disco guasto, riducendo il tempo di riparazione.

Q: Cos'è uno snapshot nel contesto RAID?

A: Un'immagine istantanea del file system, utile per backup o rollback.

Q: Cos'è la replication nei RAID?

A: Copia automatica dei dati su un altro array o sito. Può essere sincrona o asincrona.

Q: Cos'è il thin provisioning?

A: Allocazione dinamica dello spazio su disco in base all’uso reale.

Q: Cos’è l’indirizzamento LBA?

A: L’LBA (Logical Block Addressing) mappa ogni blocco logico a una posizione fisica. Il settore 0 è nella traccia più esterna del cilindro esterno.

Q: Quali sono i vantaggi e svantaggi delle SSD rispetto agli HDD?

A: Pro: accesso rapido, affidabilità, silenziosità. Contro: capacità inferiore, usura limitata delle celle (max ~100.000 cicli), no sovrascrittura diretta.

Q: Quali algoritmi migliorano l’efficienza e la durata delle SSD?

A: FTL: mappa logica dei blocchi. Garbage collection: recupera spazio. Wear leveling: distribuisce l’usura.

Q: Cos'è un RAM Disk e quando si usa?

A: È una porzione di DRAM usata come disco ad alta velocità, utile per elaborazioni temporanee ad alte prestazioni.

Q: Perché si usano ancora i nastri magnetici?

A: Per backup e archiviazione a lungo termine: sono economici e capienti, ma lenti.

Q: Qual è lo scopo degli algoritmi di disk scheduling?

A: Minimizzare il tempo di seek e massimizzare la larghezza di banda del disco.

Q: Pro e contro dell'algoritmo FCFS?

A: Pro: semplice, nessuna starvation. Contro: inefficiente in termini di seek.

Q: Qual è il rischio principale con SSTF?

A: Starvation per le richieste lontane, anche se minimizza i singoli tempi di seek.

Q: Come funziona l'algoritmo SCAN?

A: La testina serve richieste in una direzione fino a un'estremità, poi inverte. Evita starvation.

Q: Come differisce C-SCAN da SCAN?

A: Dopo aver raggiunto un'estremità, la testina torna all'inizio senza servire richieste. Offre tempi di attesa più uniformi.

Q: Cos'è un HDD e come funziona?

A: Un HDD è composto da dischi magnetici rotanti. I dati vengono registrati e letti tramite un pattern magnetico sulla superficie dei dischi.

Q: Cos'è un cilindro in un HDD?

A: È un insieme di tracce allineate su tutte le piastre.

Q: Quali sono le componenti del tempo di accesso di un HDD?

A: Seek time (tempo per spostare la testa alla traccia giusta) e rotational latency (tempo per allineare il settore giusto).

Q: Come si calcola il Tempo Medio di Accesso?

A: Average Access Time = Average Seek Time + Average Rotational Latency

Q: Cosa include il Tempo Medio di I/O?

A: Average Access Time + (Amount to Transfer / Transfer Rate) + Controller Overhead.

Q: Quali sono i vantaggi degli SSD rispetto agli HDD?

A: Sono più veloci, affidabili, consumano meno energia e non hanno parti mobili.

Q: Cosa fa il Flash Translation Layer (FTL)?

A: Mappa i blocchi logici ai fisici e gestisce la garbage collection.

Q: Cos'è un RAM drive?

A: Una porzione di RAM usata come memoria secondaria, estremamente veloce, per archiviazione temporanea.

Q: Quali sono le principali interfacce di connessione dei dischi?

A: SATA, NVMe, USB, SCSI.

Q: Quali algoritmi di disk scheduling conosci?

A: FCFS, SCAN, C-SCAN.

Q: Cos'è l'algoritmo SCAN?

A: Il braccio si muove in una direzione servendo richieste, poi torna indietro.

Q: Quali sono i livelli RAID principali?

A: RAID 0, 1, 4, 5, 6, 0+1, 1+0.

Q: Quali algoritmi di disk scheduling conosci?

A: FCFS, SCAN, C-SCAN.

Q: Cos'è l'algoritmo SCAN?

A: Il braccio si muove in una direzione servendo richieste, poi torna indietro.

Q: Quali sono i livelli RAID principali?

A: RAID 0, 1, 4, 5, 6, 0+1, 1+0.

Q: Come si calcola l'MTTDL in RAID 1?

A: $MTTDL \approx MTTF^2 / (2 \times MTTR)$.

Q: Cos'è lo striping?

A: Distribuzione dei dati su più dischi per migliorare le prestazioni.

Q: Differenze tra RAID 0 e RAID 1?

A: RAID 0: striping, prestazioni. RAID 1: mirroring, affidabilità.

Q: Come si calcola l'MTTDL in RAID 1?

A: $MTTDL \approx MTTF^2 / (2 \times MTTR)$.

Q: Cos'è lo striping?

A: Distribuzione dei dati su più dischi per migliorare le prestazioni.

Q: Differenze tra RAID 0 e RAID 1?

A: RAID 0: striping, prestazioni. RAID 1: mirroring, affidabilità.

Q: Cos'è un mutex lock in OS161?

A: Un meccanismo per proteggere sezioni critiche. Usa acquire() e release() per garantire mutua esclusione. Può essere implementato con spinlock.

Q: Cos'è uno spinlock?

A: Un lock che attende attivamente (busy waiting) finché non diventa disponibile. Efficiente solo per sezioni critiche brevi.

Q: Cos'è un semaforo in OS161?

A: Variabile intera con operazioni P (wait) e V (signal), usata per sincronizzare thread. Implementa blocco e risveglio dei thread.

Q: Come funziona la funzione P() di un semaforo?

A: Eleva gli interrupt con splhigh(), controlla se valore semaforo è 0, se sì blocca il thread con thread_sleep(), altrimenti decrementa e continua.

Q: Come funziona la funzione V() di un semaforo?

A: Eleva gli interrupt, incrementa il valore del semaforo e chiama thread_wakeup() per sbloccare i thread in attesa.

Q: Cos'è un lock in OS161?

A: Meccanismo per mutua esclusione tra thread. Solo chi ha acquisito un lock può rilasciarlo. Basato su semaforo binario.

Q: Perché si preferiscono i lock ai semafori per sezioni critiche?

A: Per ownership garantita, maggiore sicurezza, flessibilità e assenza di busy waiting con thread_sleep.

Q: Cos'è una condition variable (CV)?

A: Struttura di sincronizzazione usata con i lock. Offre cv_wait(), cv_signal() e cv_broadcast().

Q: Come funziona cv_wait() in OS161?

A: Il thread si blocca e rilascia il lock. Al risveglio, riacquisisce automaticamente il lock associato.

Q: Cosa fanno cv_signal() e cv_broadcast()?

A: cv_signal() sveglia un solo thread in attesa. cv_broadcast() sveglia tutti i thread in attesa sulla stessa condition variable.

Q: Cos'è un monitor?

A: Un'astrazione per sincronizzazione che include mutua esclusione e variabili condition. Solo un processo alla volta può entrare.

Q: Quali sono le due politiche di scheduling nei monitor?

A: signal-and-wait (il chiamante si blocca) e signal-and-continue (il chiamante continua, il risvegliato aspetta).

Q: Cos'è un wait channel (wchan) in OS161?

A: Strumento del kernel per bloccare e risvegliare thread. Basato su spinlock. Funzioni: wchan_sleep() e wchan_wakeone().

Q: Cosa fa wchan_sleep()?

A: Sospende il thread con thread_switch(), rilascia lo spinlock, e lo riacquisisce al risveglio.

Q: Cosa fa wchan_wakeone()?

A: Risveglia un singolo thread bloccato su un wait channel e lo rende runnable.

Q: Qual è il compito della funzione mips_trap in OS161?

A: Gestisce eccezioni, interrupt e chiamate di sistema. Legge il codice del motivo dal trapframe e chiama il gestore appropriato.

Q: Cosa fa syscall() in OS161?

A: Gestisce le chiamate di sistema: legge il numero di syscall da v0, esegue la funzione appropriata, gestisce retval/err, aggiorna epc per evitare ripetizione.

Q: Cosa succede se la TLB MIPS non trova una corrispondenza?

A: Solleva una page fault exception. OS161 gestisce l’errore con vm_fault, che carica una voce TLB per la pagina richiesta.

Q: Qual è il compito della funzione vm_fault() in dumbvm?

A: Gestisce l’eccezione da page fault. Crea una voce TLB con i dati corretti dallo spazio di indirizzamento e la carica.

Q: Cosa fa execv() in OS161?

A: Rimpiazza lo spazio di indirizzamento di un processo con uno nuovo e carica un file ELF da eseguire, configurando anche gli argomenti.

Q: Cos'è un file ELF?

A: Contiene segmenti con codice e dati. Specifica dove caricare il codice nello spazio degli indirizzi virtuali e l'indirizzo iniziale d'esecuzione.

Q: Cosa rappresenta il problema della sezione critica?

A: È il problema di garantire che solo un processo per volta acceda a una sezione di codice che usa risorse condivise.

Q: Quali sono le tre condizioni fondamentali per risolvere la sezione critica?

A: Mutua esclusione, progresso e attesa limitata (bounded waiting).

Q: Cosa fa l'istruzione test_and_set?

A: Imposta un lock a true e restituisce il valore precedente. È atomica, usata per acquisire lock evitando race condition.

Q: Cosa fa compare_and_swap (CAS)?

A: Confronta il valore corrente con uno atteso e, se coincide, lo sostituisce con uno nuovo. Altrimenti non fa nulla. Ritorna il valore letto.

Q: Cosa fa getppages in OS161?

A: Alloca pagine fisiche. Cerca prima tra le pagine libere nella bitmap e, se non ne trova, chiama ram_stealmem per ottenere memoria contigua.

Q: Cosa fa getfreeppages?

A: Cerca un blocco contiguo di pagine libere nella bitmap per l'allocazione di memoria fisica.

Q: Cosa fa freeppages?

A: Marca le pagine come libere nella bitmap, permettendone il riutilizzo futuro.

Q: Cosa consente il paging nello user space in OS161?

A: Permette l'allocazione non contigua della memoria utente, migliorando la gestione della memoria e supportando spazi di indirizzamento virtuali più grandi.

Q: Qual è il ruolo dei registri MIPS in OS161?

A: Vengono utilizzati per memorizzare dati e indirizzi durante l'esecuzione. Alcuni sono temporanei, altri devono essere salvati/ripristinati durante il context switch.

Q: Dove avviene il salvataggio/ripristino dei registri MIPS in OS161?

A: Nel file switch.S, durante il dispatching tra thread.

Q: Cos'è la struttura 'proc' in OS161?

A: Rappresenta un processo. Contiene nome, numero di thread, spazio di indirizzamento e directory di lavoro.

Q: Ogni thread ha uno stack utente e uno kernel?

A: Sì. Ogni thread ha uno stack per l'esecuzione utente e uno per l'esecuzione kernel, per protezione e gestione separata.

Q: Qual è il ruolo di runprogram()?

A: Avvia un processo utente: apre il file, crea lo spazio d'indirizzamento, carica l'eseguibile, prepara lo stack, entra in user mode.

Q: Cosa fa enter_new_process()?

A: È l'ultima funzione eseguita dal kernel prima di passare il controllo all’eseguibile utente. Imposta il contesto iniziale e invoca mips_usermode().

Q: Cos'è un processo in OS161?

A: Un processo è un programma in esecuzione. Include codice (text), stato (program counter, registri), stack (parametri e variabili locali), sezione dati (variabili globali) e heap (memoria dinamica).

Q: Qual è la differenza tra programma e processo?

A: Il programma è un'entità passiva (file su disco), il processo è attivo (programma in esecuzione in memoria).

Q: Cos'è un thread in OS161?

A: Un thread è un'unità di esecuzione di un processo, con il proprio contesto (PC, SP, registri), stack privato, ma condivide codice e dati con altri thread del processo.

Q: Qual è la struttura 'thread' in OS161?

A: È una struttura dati che rappresenta un thread kernel. Contiene nome, stato, stack, contesto dei registri, CPU assegnata, e riferimento al processo.

Q: A cosa serve la funzione thread_fork in OS161?

A: Crea un nuovo thread kernel, specificando nome, processo, funzione di entrypoint e dati. Lo rende schedulabile inserendolo nella runqueue.

Q: Cosa fa thread_exit in OS161?

A: Termina il thread corrente, rimuovendolo dal sistema. Libera risorse, disabilita interruzioni e invoca lo scheduler per un context switch.

Q: Qual è lo scopo di thread_yield?

A: Permette al thread corrente di cedere volontariamente la CPU pur rimanendo eseguibile. Viene reinserito nella runqueue.

Q: Cosa fa thread_switch?

A: Esegue il context switch: salva il contesto del thread corrente e passa al successivo dalla runqueue della CPU.

Q: Che ruolo ha getppages in OS161?

A: Alloca pagine fisiche (multipli di 4 KB) chiamando ram_stealmem, con protezione tramite lock per ambienti multithreaded.

Q: Cosa fa ram_stealmem?

A: Alloca memoria fisica contigua a partire da firstpaddr. Usata nella fase iniziale del kernel prima che la VM sia attiva.

Q: Cos'è OS161?

A: Un sistema operativo didattico che gira su architettura MIPS simulata. È progettato per essere semplice e comprensibile.

Q: Qual è il comportamento di un processo quando termina in OS161 e cosa succede se nessuno chiama waitpid() su di esso?

A: Il processo entra in uno stato "zombie" e la sua struttura dati struct proc resta in memoria finché un altro processo non recupera il suo stato di uscita tramite waitpid().

Q: Quale funzione viene utilizzata per creare un nuovo processo tramite proc_create_runprogram()?

A: La funzione crea il processo e lo avvia, salvando il suo pid, ad esempio in proc→p_pid, e possibilmente utilizzando sys_getpid(proc) per recuperarne il pid.

Q: Quando un processo termina e un altro processo chiama sys_waitpid(), cosa succede?

A: La funzione cerca il processo corrispondente nella tabella, chiama proc_wait() passando il processo, attende la sua terminazione, libera la struct proc e ritorna il codice di uscita.

Q: Qual è l'obiettivo di proc_wait() in relazione alla funzione sys_waitpid()?

A: proc_wait() viene chiamata internamente da sys_waitpid() ed è responsabile di attendere la terminazione del processo, di liberare la sua struttura dati e di fornire il codice di uscita.

Q: Come si garantisce la corretta gestione dei processi zombie in OS161?

A: Attraverso l'uso di waitpid(), che permette a un processo di aspettare la terminazione del processo figlio, recuperarne lo stato e liberare le risorse assegnate, evitando che rimangano in memoria in stato zombie.

Q: Perché è importante implementare correttamente waitpid() nel sistema operativo?

A: Per permettere la sincronizzazione tra processi nel loro ciclo di vita e per liberare correttamente le risorse, prevenendo perdite di memoria o processi zombie inutilmente in memoria.

Q: Che ruolo gioca sys_getpid() nel contesto dei processi?

A: sys_getpid() permette di ottenere il pid del processo corrente, utile per identificare e gestire i processi in tutte le operazioni di sistema.

Q: Quali sono i passi principali coinvolti nella terminazione di un processo e nell'attesa di un processo figlio?

A: Creazione del processo con proc_create_runprogram(), terminazione del processo, gestione di processi zombie, chiamata a waitpid() per recuperare lo stato di uscita e liberare le risorse.

Q: Qual è il ruolo di proc_end_waitpid() nel ciclo di gestione dei processi in OS161?

A: Rimuove un processo dalla tabella dei processi, dealloca le strutture di sincronizzazione associate a waitpid() come semafori, condition variable, e lock, e libera le risorse di memoria usate dal processo.

Q: Come viene assegnato un PID univoco in OS161?

A: Durante l'inizializzazione, proc_init_waitpid() cerca un indice libero nella tabella dei processi, lo assegna a p_pid e lo inserisce nella tabella, garantendo unicità finché ci sono slot disponibili.

Q: Che funzione svolge proc_init_waitpid()?

A: Assegna un PID univoco a un nuovo processo, inserisce il processo nella tabella globale, e inizializza le strutture di sincronizzazione come semafori o condition variable per waitpid().

Q: Perché viene preferibile delegare la distruzione di un processo a proc_wait() invece che a sys__exit()?

A: Per evitare che il processo rimanga zombie; proc_wait() attende la terminazione, recupera lo stato di uscita e distrugge correttamente la struttura, assicurando che il padre possa recuperare lo stato senza problemi.

Q: Quali campi sono fondamentali in struct proc per la gestione di waitpid() e perché?

A: p_status salva lo stato di uscita, p_pid identifica il processo univocamente, e le strutture di sincronizzazione (p_sem o p_cv e p_lock) sono usate per sincronizzare il padre con il termine del figlio.

Q: Che cosa succede quando un processo chiama exit() in OS161?

A: Cambia lo stato a terminato, imposta p_status, risveglia il padre tramite V(p_sem) o cv_signal(), ma la struttura proc resta fino a quando waitpid() non la libera, per evitare processi zombie.

Q: In cosa consiste l'implementazione di proc_end_waitpid()?

A: Rimuove il processo dalla tabella, dealloca i semafori o condition variable e lock associati, e libera la risorsa di memoria, garantendo la corretta gestione delle risorse dei processi terminati.

Q: Qual è il ruolo di proc_wait() nel gestire la terminazione di un processo in OS161?

A: proc_wait() blocca il processo padre finché il processo figlio non termina, recupera lo stato di uscita (p_status), e libera le risorse associate alla struttura proc, garantendo una gestione corretta del ciclo di vita del processo.

Q: Come viene sincronizzata la terminazione del processo tra exit() e waitpid()?

A: exit() aggiorna p_status e risveglia il padre con V(p_sem) o cv_signal(), mentre waitpid() si blocca attendendo questa notifica, poi recupera p_status e libera la struttura proc.

Q: Che funzione ha la variabile return_status in proc_wait()?

A: Memorizza il valore di p_status del processo terminato, che sarà restituito come risultato della chiamata proc_wait().

Q: Qual è il comportamento di proc_destroy()?

A: Libera tutte le risorse associate al processo, incluse le strutture di sincronizzazione (semaphori, CV, lock) e la memoria allocata, e rimuove il processo dalla tabella globale prontamente usata dal sistema.

Q: Perché la rimozione di un processo termina a proc_destroy() e non immediatamente in exit()?

A: Per garantire che il processo padre abbia il tempo di recuperare lo stato di uscita prima che le risorse vengano deallocate, evitando così processi zombie.

Q: Come avviene il recupero della risorsa p_sem o p_cv in proc_end_waitpid()?

A: Viene deallocata tramite sem_destroy() o cv_destroy() e lock_destroy(), rispettivamente, per liberare le risorse di sincronizzazione associate al processo terminato.

Q: Qual è l'importanza di mantenere p_status aggiornato?

A: Permette al processo padre di conoscere lo stato di uscita del processo figlio, utile in waitpid() per decidere il comportamento successivo e per gestire l'esito della terminazione.

Q: Cosa sono paddr_t e vaddr_t?

A: paddr_t è un indirizzo fisico in RAM; vaddr_t è un indirizzo virtuale usato da processi o kernel.

Q: Come avviene il debugging del kernel OS161?

A: Con sys161 -w kernel in ascolto e mips-harvard-os161-gdb su un secondo terminale.

Q: Cosa fa thread_fork in OS161?

A: Crea un nuovo thread specificando nome, processo padre e funzione da eseguire.

Q: Che struttura rappresenta un processo in OS161?

A: La struct proc, contenente nome, lock, numero thread, addrspace, cwd, e info waitpid.

Q: Cos'è la struct addrspace?

A: Rappresenta lo spazio di indirizzamento virtuale di un processo.

Q: Come funziona DumbVM?

A: Traduce vaddr in paddr usando segmenti fissi per codice, dati e stack, con allocazione contigua.

Q: A cosa serve ram_stealmem?

A: Alloca pagine fisiche in modo contiguo direttamente dalla RAM.

Q: Che ruolo ha la funzione getppages?

A: Interfaccia di allocazione memoria che usa getfreeppages o ram_stealmem.

Q: Come viene gestita la deallocazione della memoria?

A: Con freeppages, che segna le pagine come libere in freeRamFrames.

Q: Come funziona la syscall write?

A: Scrive dati su stdout/stderr leggendo byte da un buffer utente.

Q: Come funziona la syscall read?

A: Legge dati da stdin, byte per byte, memorizzandoli in un buffer.

Q: Cos'è una syscall e come viene gestita in OS161?

A: È un'interfaccia per servizi del kernel, gestita da syscall() che usa un trapframe.

Q: Come funziona waitpid?

A: Permette a un processo padre di attendere la terminazione del figlio e recuperarne lo stato.

Q: Che struttura tiene traccia dei processi in OS161?

A: La tabella processTable, che associa PID a struct proc.

Q: Cosa sono race condition, deadlock e starvation?

A: Problemi della concorrenza dovuti all'ordine di esecuzione e gestione delle risorse.

Q: Cos'è la soluzione di Peterson?

A: Un algoritmo software per mutua esclusione tra due processi.

Q: Cosa fa compare_and_swap?

A: Aggiorna una variabile solo se ha il valore atteso, garantendo atomicità.

Q: Che differenza c'è tra lock e semaforo?

A: Il lock ha ownership, può essere rilasciato solo dal proprietario; il semaforo no.

Q: Cos'è una condition variable?

A: Meccanismo di sincronizzazione che consente a un thread di attendere una condizione.

Q: Come funziona cv_wait?

A: Il thread rilascia il lock, va in attesa e lo riacquisisce al risveglio.

Q: Cos'è una wait channel?

A: Struttura kernel per mettere in attesa i thread in modo efficiente.

Q: Cosa fa la funzione proc_wait?

A: Attende la terminazione del processo figlio e ne recupera lo stato.

Q: Come funziona la system call exit?

A: Libera l'addrspace, notifica il padre e termina il thread con thread_exit.

Q: Qual è la differenza tra proc_wait e sys_waitpid?

A: proc_wait lavora con puntatori a proc, sys_waitpid lavora con PID.

Q: Come si implementano lock in OS161?

A: Con semaforo binario o con wait channel e spinlock, a seconda dell'opzione.

Q: Come funziona il test tt1?

A: Crea 8 thread che stampano caratteri; verifica la gestione dei thread.

Q: Qual è il comportamento di un processo quando termina in OS161 e cosa succede se nessuno chiama waitpid() su di esso?

A: Il processo entra in uno stato "zombie" e la sua struttura dati struct proc resta in memoria finché un altro processo non recupera il suo stato di uscita tramite waitpid().

Q: Quale funzione viene utilizzata per creare un nuovo processo tramite proc_create_runprogram()?

A: La funzione crea il processo e lo avvia, salvando il suo pid, ad esempio in proc→p_pid, e possibilmente utilizzando sys_getpid(proc) per recuperarne il pid.

Q: Quando un processo termina e un altro processo chiama sys_waitpid(), cosa succede?

A: La funzione cerca il processo corrispondente nella tabella, chiama proc_wait() passando il processo, attende la sua terminazione, libera la struct proc e ritorna il codice di uscita.

Q: Qual è l'obiettivo di proc_wait() in relazione alla funzione sys_waitpid()?

A: proc_wait() viene chiamata internamente da sys_waitpid() ed è responsabile di attendere la terminazione del processo, di liberare la sua struttura dati e di fornire il codice di uscita.

Q: Come si garantisce la corretta gestione dei processi zombie in OS161?

A: Attraverso l'uso di waitpid(), che permette a un processo di aspettare la terminazione del processo figlio, recuperarne lo stato e liberare le risorse assegnate, evitando che rimangano in memoria in stato zombie.

Q: Perché è importante implementare correttamente waitpid() nel sistema operativo?

A: Per permettere la sincronizzazione tra processi nel loro ciclo di vita e per liberare correttamente le risorse, prevenendo perdite di memoria o processi zombie inutilmente in memoria.

Q: Che ruolo gioca sys_getpid() nel contesto dei processi?

A: sys_getpid() permette di ottenere il pid del processo corrente, utile per identificare e gestire i processi in tutte le operazioni di sistema.

Q: Quali sono i passi principali coinvolti nella terminazione di un processo e nell'attesa di un processo figlio?

A: Creazione del processo con proc_create_runprogram(), terminazione del processo, gestione di processi zombie, chiamata a waitpid() per recuperare lo stato di uscita e liberare le risorse.

Q: Qual è il ruolo di proc_end_waitpid() nel ciclo di gestione dei processi in OS161?

A: Rimuove un processo dalla tabella dei processi, dealloca le strutture di sincronizzazione associate a waitpid() come semafori, condition variable, e lock, e libera le risorse di memoria usate dal processo.

Q: Come viene assegnato un PID univoco in OS161?

A: Durante l'inizializzazione, proc_init_waitpid() cerca un indice libero nella tabella dei processi, lo assegna a p_pid e lo inserisce nella tabella, garantendo unicità finché ci sono slot disponibili.

Q: Che funzione svolge proc_init_waitpid()?

A: Assegna un PID univoco a un nuovo processo, inserisce il processo nella tabella globale, e inizializza le strutture di sincronizzazione come semafori o condition variable per waitpid().

Q: Perché viene preferibile delegare la distruzione di un processo a proc_wait() invece che a sys__exit()?

A: Per evitare che il processo rimanga zombie; proc_wait() attende la terminazione, recupera lo stato di uscita e distrugge correttamente la struttura, assicurando che il padre possa recuperare lo stato senza problemi.

Q: Quali campi sono fondamentali in struct proc per la gestione di waitpid() e perché?

A: p_status salva lo stato di uscita, p_pid identifica il processo univocamente, e le strutture di sincronizzazione (p_sem o p_cv e p_lock) sono usate per sincronizzare il padre con il termine del figlio.

Q: Che cosa succede quando un processo chiama exit() in OS161?

A: Cambia lo stato a terminato, imposta p_status, risveglia il padre tramite V(p_sem) o cv_signal(), ma la struttura proc resta fino a quando waitpid() non la libera, per evitare processi zombie.

Q: In cosa consiste l'implementazione di proc_end_waitpid()?

A: Rimuove il processo dalla tabella, dealloca i semafori o condition variable e lock associati, e libera la risorsa di memoria, garantendo la corretta gestione delle risorse dei processi terminati.

Q: Quali sono i campi principali della struttura proc per la gestione del ciclo di vita dei processi in OS161?

A: I campi principali sono p_status, p_pid, p_name, p_lock, p_cv o p_sem, p_addrspace, p_cwd, oltre a contatori di thread e altre strutture di sistema.

Q: A cosa serve il campo p_status nella struttura proc?

A: p_status salva lo stato di uscita del processo ed è utilizzato dalla funzione waitpid() per determinare se un processo è terminato.

Q: Come viene identificato univocamente un processo in OS161?

A: Attraverso il campo p_pid, che assegna un identificatore univoco a ogni processo.

Q: Qual è lo scopo di un meccanismo di sincronizzazione come semafori o variabili di condizione?

A: Permette al processo padre di attendere la terminazione del processo figlio, garantendo una gestione corretta delle operazioni di exit() e waitpid().

Q: Quando p_status viene inizializzato e perché?

A: p_status viene inizializzato a 0 per indicare che il processo non ha ancora terminato o segnalato il suo stato di uscita.

Q: Quali strutture di sincronizzazione vengono usate nelle implementazioni di waitpid()?

A: Si possono usare semafori (p_sem) o variabili di condizione (p_cv) e lock (p_lock), a seconda della configurazione.

Q: Come viene assegnato un PID ai processi durante l'inizializzazione in OS161?

A: Utilizzando una strategia circolare in una tabella di processi, tramite la funzione proc_init_waitpid(), che assegna un PID libero e lo registra nella tabella.

Q: In cosa consiste la funzione common_prog() presente in questa versione di sys_waitpid()?

A: Crea un processo, avvia un thread per eseguire il programma, quindi ottiene il PID del processo figlio e attende la sua terminazione tramite sys_waitpid(pid).

Q: Qual è il ruolo della tabella globale dei processi in sys_waitpid()?

A: Tiene traccia di tutti i processi e permette di effettuare l'attesa e il recupero dello stato di uscita attraverso i PID, poiché i kernel o processi utente non accedono direttamente a proc.

Q: Perché nel kernel OS161 bisogna usare KASSERT()?

A: Per verificare che le condizioni critiche siano vere durante l'esecuzione; se false, il sistema effettua un panico per prevenire danni seri come sovrascrizioni di memoria o comportamento imprevedibile.

Q: Qual è l’obiettivo principale di proc_init_waitpid() in OS161?

A: Inizializza i campi di un processo, in particolare p_pid, p_status, e le strutture di sincronizzazione (p_sem o p_cv e p_lock), per consentire la gestione della terminazione e dell’attesa tramite waitpid().

Q: Come funziona proc_init_waitpid() per assegnare un PID a un nuovo processo?

A: Cerca un indice libero nella tabella dei processi usando una strategia circolare, assegna il processo a quell'indice, e imposta p_pid di conseguenza. Se la tabella è piena, genera un panico.

Q: Qual è il valore di p_status all'inizio durante l'inizializzazione con proc_init_waitpid()?

A: p_status viene impostato a 0, indicando uno stato iniziale di uscita ancora non definito.

Q: Come viene creato il semaforo o le variabili di condizione associate a un processo in proc_init_waitpid()?

A: Se si usa USE_SEMAPHORE_FOR_WAITPID, si crea un semaforo con sem_create(). Altrimenti, si creano una variabile di condizione (cv_create()) e un lock (lock_create()).

Q: Qual è lo scopo di p_sem, p_cv, e p_lock nei processi?

A: Sono usati per sincronizzare il processo padre con il processo figlio durante la terminazione, permettendo al padre di aspettare (waitpid()) e di leggere lo stato di uscita.

Q: Cosa succede se la tabella dei processi è piena durante proc_init_waitpid()?

A: Viene chiamato panic() con un messaggio di errore, indicando che il sistema ha troppi processi attivi e la tabella è piena.

Q: Qual è il ruolo di p_pid nel sistema di gestione dei processi in OS161?

A: Identifica univocamente un processo nel sistema, permettendo di riferirsi ad esso nelle chiamate di sistema come waitpid() e nelle tabelle globali.

Q: Qual è il ruolo della funzione proc_search_pid() in OS161?

A: Restituisce il puntatore alla struttura proc associata a un dato PID, utile per implementare waitpid() o per cercare figli da parte del processo padre.

Q: Come funziona proc_search_pid()?

A: Controlla che il PID sia valido (tra 1 e MAX_PROC), quindi ritorna il puntatore nella tabella processTable.proc[pid], verificando che il PID corrisponda.

Q: Che cosa rappresenta la struttura _processTable in OS161?

A: È una tabella globale che tiene traccia di tutti i processi attivi, inclusa la lista di puntatori a proc, l'indice dell'ultimo PID assegnato (last_i), e un lock (lk) per l'accesso concorrente.

Q: Qual è lo scopo di proc_destroy() in OS161?

A: Dealloca le risorse associate a un processo, come p_sem o p_cv, rimuove il processo dalla tabella, e libera la memoria strutturale, garantendo che non rimangano processi zombie.

Q: Come funziona proc_end_waitpid()?

A: Rimuove il processo dalla tabella dei processi, dealloca le strutture di sincronizzazione (p_sem, p_cv, p_lock), e libera le risorse usate dal processo, come parte della gestione di terminazione.

Q: Perché proc_wait() non distrugge immediatamente il processo?

A: Per garantire che il padre possa recuperare lo stato di uscita del figlio, proc_wait() aspetta che il processo termini, recupera lo stato e poi distrugge la struttura proc.

Q: Come funziona proc_wait() in OS161?

A: Usa un meccanismo di sincronizzazione (semaphore o CV+lock) per bloccare il processo padre fino a quando il figlio termina. Quindi legge p_status, chiama proc_destroy(), e restituisce lo stato di uscita.

Q: Che cosa succede quando un processo termina in sys__exit()?

A: Notifica il termine con thread_exit() ma non distrugge subito la struttura proc, lasciando questa operazione a proc_wait() per evitare processi zombie.

Q: Come vengono rimosse risorse di sincronizzazione in proc_end_waitpid()?

A: Se si usa un semaforo, si chiama sem_destroy(). Se si usano CV e lock, si chiamano cv_destroy() e lock_destroy(). Poi si rimuove il processo dalla tabella.

Q: Che cosa rappresenta p_status in una proc?

A: Memorizza lo stato di uscita del processo, impostato da exit() e letto da waitpid(); inizialmente a 0, poi aggiornato al termine.

Q: Perché si utilizza una strategia circolare con last_i in proc_init_waitpid()?

A: Per cercare in modo efficiente una posizione libera nella tabella dei processi, ripartendo dall'ultimo assegnato, ottimizzando l'allocazione dei PID.

Q: Qual è il ruolo di KASSERT() nel sistema?

A: È una macro di controllo che verifica condizioni critiche durante l'esecuzione. Se una condizione fallisce, blocca il sistema e stampa un messaggio di errore, facilitando il debugging.

Q: Come viene garantita la sicurezza delle operazioni sulla tabella dei processi?

A: Utilizzando un spinlock (processTable.lk) per controllare l'accesso concorrente e mantenere l'integrità dei dati.

Q: Che cosa si intende con "processi zombie" in OS161?

A: Processi terminati che non sono ancora stati distrutti, ancora presenti nella tabella dei processi, perché il padre non ha chiamato waitpid() per recuperare lo stato.

Q: Qual è il comportamento di proc_destroy() in relazione a waitpid()?

A: proc_destroy() viene chiamato da proc_wait() dopo aver recuperato lo stato di uscita, garantendo che il processo sia correttamente eliminato dalla tabella e tutte le risorse siano deallocate.

Q: Qual è lo scopo principale di proc_init_waitpid()?

A: Assegnare un PID unico, registrare il processo nella tabella globale, e inizializzare le primitive di sincronizzazione (p_sem o p_cv e p_lock) per supportare waitpid().

Q: Come vengono allocate le risorse di sincronizzazione in proc_init_waitpid()?

A: Se si utilizza un semaforo, si crea con sem_create(). Se si utilizzano CV e lock, si creano con cv_create() e lock_create() rispettivamente.

Q: Qual è il ruolo di p_pid all'interno della struttura proc?

A: Identifica in modo univoco il processo nel sistema, assegnato da proc_init_waitpid() e usato per cercare processi specifici.

Q: Cosa fa la funzione proc_end_waitpid()?

A: Rimuove un processo dalla tabella, dealloca le primitive di sincronizzazione, e libera le risorse associate a waitpid().

Q: Come viene gestita la rimozione di un processo dalla tabella in proc_end_waitpid()?

A: Acquisisce il lock sulla tabella, azzera l'entry corrispondente a p_pid, poi dealloca p_sem o p_cv e p_lock secondo l'opzione.

Q: Che cosa rappresenta p_status in struct proc?

A: Lo stato di uscita del processo, vengono impostati da exit() e letti da waitpid(), inizialmente a 0.

Q: Qual è il ruolo di proc_destroy() nel ciclo di vita di un processo?

A: Dealloca le risorse della struttura proc, rimuove il processo dalla tabella e libera la memoria, mantenendo il sistema pulito da processi zombie.

Q: Come si assegna un nuovo p_pid in proc_init_waitpid()?

A: Si cerca un indice libero nella tabella usando un'operazione circolare partendo da last_i+1 e, una volta trovato, si assegna p_pid e si aggiorna last_i.

Q: Cosa succede se la tabella dei processi è piena in proc_init_waitpid()?

A: La funzione panica con il messaggio "troppi processi. proc table is full" quando non è possibile assegnare un PID.

Q: Qual è la funzione di proc_search_pid()?

A: Cerca e ritorna il puntatore alla struttura proc associata a uno specifico PID, facilitando operazioni come waitpid().

Q: Come viene garantita la sicurezza durante l'assegnazione dei PID?

A: Acquisendo il lock processTable.lk prima di modificare la tabella, per garantire accesso esclusivo e integrità dei dati.

Q: Che cosa succede quando un processo termina e exit() viene chiamato?

A: Aggiorna p_status con il valore di uscita, avvisa eventualmente il padre tramite la primitive di sincronizzazione, ma la distruzione del processo avviene tramite waitpid().

Q: Perché in proc_end_waitpid() si rimuove il processo dalla tabella?

A: Per evitare riferimenti pendenti e per mantenere aggiornata la tabella, consentendo di riutilizzare gli slot liberi per nuovi processi.

Q: Come si deallocano le primitive di sincronizzazione in proc_end_waitpid()?

A: Se si usa un semaforo, si chiama sem_destroy(). Se si usa CV e lock, si chiamano cv_destroy() e lock_destroy().

Q: Qual è la funzione principale del ciclo di vita di proc in relazione a waitpid()?

A: Creare, assegnare PID, sincronizzare con padre, attendere terminazione, leggere stato di uscita, e infine deallocare le risorse.

Q: Quali sono le principali categorie di dispositivi di I/O?

A: Dispositivi di memorizzazione (dischi, SSD), dispositivi di trasmissione (schede di rete), interfacce utente (tastiere, mouse, schermi).

Q: Cos'è il memory-mapped I/O?

A: È una tecnica in cui i registri dei dispositivi di I/O sono mappati nello spazio di indirizzamento della memoria, rendendoli accessibili tramite normali istruzioni di load/store.

Q: Come funziona il polling per l'I/O?

A: La CPU interroga ripetutamente il registro di stato del dispositivo. È semplice ma inefficiente per dispositivi lenti perché causa busy waiting.

Q: Cos'è un interrupt e come migliora il polling?

A: Un interrupt è un segnale inviato al processore da un dispositivo pronto. La CPU risponde eseguendo una routine di servizio, evitando l’attesa attiva del polling.

Q: Cosa fa il controller DMA?

A: Permette ai dispositivi di trasferire dati direttamente in memoria senza coinvolgere la CPU. Migliora le prestazioni usando il bus mastering e riduce il carico sulla CPU.

Q: Qual è il ruolo del device driver?

A: È un modulo software che fornisce un'interfaccia standard al kernel, nascondendo le peculiarità hardware dei dispositivi e implementando le system call di I/O.

Q: Qual è la differenza tra dispositivi a blocchi e a caratteri?

A: A blocchi (dischi): accesso a blocchi di dati con DMA. A caratteri (tastiere): accesso byte per byte con funzioni come get() e put().

Q: Che differenza c’è tra I/O bloccante, non bloccante e asincrono?

A: Bloccante: il processo aspetta. Non bloccante: continua l’esecuzione e verifica lo stato. Asincrono: una callback o evento segnala la fine dell’I/O.

Q: Cos'è l'I/O vettorizzato?

A: Permette di effettuare più operazioni di I/O con una sola system call. Riduce il numero di chiamate, migliora performance e può offrire atomicità.

Q: Cosa fa il buffering nell'I/O?

A: Utilizza aree di memoria tampone per gestire differenze di velocità e dimensioni tra dispositivi. Permette doppio buffering per efficienza e coerenza.

Q: Cos'è l'allocazione a lista concatenata?

A: Ogni file è memorizzato in blocchi non contigui collegati tra loro. Ogni blocco contiene un puntatore al successivo. Vantaggi: elimina frammentazione esterna. Svantaggi: accesso diretto lento e fragilità in caso di perdita di link.

Q: Come funziona la FAT (File Allocation Table)?

A: È una tabella che tiene traccia della sequenza dei blocchi di un file. Migliora fault tolerance e accesso diretto, memorizzata in RAM per velocità. Ogni blocco ha un puntatore nella FAT.

Q: Cos'è l'allocazione a indice?

A: Ogni file ha un blocco indice che contiene gli indirizzi dei blocchi dati. Vantaggi: accesso diretto ed eliminazione frammentazione esterna. Svantaggi: limite dimensione file e necessità di livelli multipli d’indice per file grandi.

Q: Come gestisce Unix l'allocazione a indice?

A: Usa I-node con indici diretti, indiretti, doppi e tripli. Permette scalabilità dei file: pochi livelli per file piccoli, più livelli per file grandi.

Q: Quali tecniche esistono per la gestione dello spazio libero?

A: Bitmap, liste concatenate, grouping, counting. Bitmap è efficiente per trovare blocchi; grouping e counting migliorano efficienza della lista concatenata.

Q: Come si ottimizzano le prestazioni del file system?

A: Tecniche: preallocazione dei metadati, collocazione dati/metadati, buffer cache, scritture asincrone, free-behind e read-ahead.

Q: Cos'è la Page Cache e la Unified Buffer Cache?

A: Page Cache mappa i blocchi su pagine di memoria (memory-mapped I/O). Unified Buffer Cache evita doppio caching unificando page cache e buffer cache.

Q: Come avviene il recovery del file system?

A: Attraverso log structured (journal) e backup. I log salvano lo stato e le operazioni in corso, permettendo il recupero dopo un crash.

Q: Cosa caratterizza la Directory a due livelli (Two-Level Directory)?

A: Ogni utente ha una propria directory. La root contiene una directory per ogni utente. Risolve i problemi di naming e grouping, ma non consente sottogruppi logici all'interno della directory utente.

Q: Cosa caratterizza la Directory ad albero (Tree-Structured Directory)?

A: Struttura gerarchica con directory e sottodirectory. Supporta path assoluti e relativi. Utile per organizzazione logica e comandi come cd, mkdir, rm.

Q: Cosa caratterizza la Directory a grafo aciclico (Acyclic-Graph Directory)?

A: Permette la condivisione di file o directory tramite link. Gestisce aliasing. Richiede soluzioni per evitare dangling pointers come backpointers o reference counting.

Q: Cosa caratterizza la Directory a grafo generale (General Graph Directory)?

A: Estensione del grafo aciclico che consente cicli. Il sistema deve rilevare i cicli e usare garbage collection o restrizioni per evitarli.

Q: Cosa significa montare un file system (Mounting)?

A: Collega un file system fisico a una directory esistente nel file system corrente. Può essere fatto all'avvio o dinamicamente. Un file system non montato non è accessibile.

Q: Come funziona la condivisione dei file nei sistemi multiutente?

A: Tramite ID utente e di gruppo, permessi RWX, ACL, e protocolli di rete. File e directory hanno proprietario e gruppo. La protezione è fondamentale per coerenza e sicurezza.

Q: Quali sono i permessi di accesso ai file nei sistemi Unix-like?

A: Read (R), Write (W), Execute (X). Ogni file ha permessi assegnati a Owner, Group, Others. Esempio: rw-rw-r-- $\rightarrow$ Proprietario e gruppo possono leggere e scrivere, altri solo leggere.

Q: Cos'è un ACL (Access Control List)?

A: Struttura che specifica permessi granulari per utenti o gruppi. Permette di definire eccezioni o regole dettagliate rispetto ai permessi standard RWX.

Q: Cos'è la gestione I/O nei sistemi operativi?

A: È la componente che controlla l'interazione tra il sistema e i dispositivi I/O, astratta tramite driver.

Q: Quali sono i componenti hardware principali per l'I/O?

A: Porta, bus (es. PCIe), e controller (es. HBA).

Q: Come comunica la CPU con un controller I/O?

A: Tramite registri mappati in memoria o porte I/O.

Q: Cos'è il polling e quando è inefficiente?

A: Tecnica in cui la CPU controlla ripetutamente lo stato del dispositivo; inefficiente con dispositivi lenti.

Q: Cosa sono gli interrupt e come migliorano l'efficienza?

A: Segnali hardware che avvisano la CPU quando un dispositivo è pronto, evitando il polling continuo.

Q: Differenza tra interrupt, eccezioni e trappole?

A: Interrupt: evento esterno. Eccezione: errore interno. Trappola: istruzione software.

Q: Cos'è il DMA e quale vantaggio offre?

A: Permette trasferimento diretto tra memoria e dispositivo senza CPU, utile per grandi volumi di dati.

Q: Cosa fa il kernel per ottimizzare l'I/O?

A: Gestisce buffering, caching, spooling, errori e pianificazione I/O.

Q: Cos'è un file system?

A: Sistema che organizza e gestisce file e directory sui dispositivi di memorizzazione.

Q: Quali sono le operazioni fondamentali sui file?

A: Creazione, lettura, scrittura, seek, eliminazione, truncamento.

Q: Cos'è un file lock e quali tipi esistono?

A: Meccanismo di blocco per garantire l'accesso esclusivo; può essere condiviso o esclusivo.

Q: Differenza tra accesso sequenziale e diretto?

A: Sequenziale: lettura/scrittura ordinata. Diretto: accesso a blocchi specifici.

Q: Come funziona una directory tree-structured?

A: Ogni file ha un percorso unico; gli utenti possono creare sottodirectory.

Q: Quali metodi di accesso ai file esistono?

A: Sequenziale, diretto, indicizzato (es. ISAM).

Q: Cos'è un FCB?

A: File Control Block: contiene metadati del file, come permessi e posizione sul disco.

Q: Quali sono i principali metodi di allocazione dei file?

A: Contigua, legata, indicizzata.

Q: Cos'è la strategia di reclaiming pages?

A: Una tecnica del kernel che libera memoria quando la free list scende sotto una soglia minima, tramite una routine detta "reaper", per prevenire il thrashing e mantenere il sistema reattivo.

Q: Cos'è NUMA (Non-Uniform Memory Access)?

A: È un'architettura in cui l'accesso alla memoria varia in base alla CPU: ogni CPU accede più velocemente alla sua memoria locale rispetto a quella delle altre CPU.

Q: Cosa causa il thrashing?

A: Un numero eccessivo di page fault causati da processi con troppo poche pagine disponibili, portando il sistema a passare più tempo a gestire pagine che a eseguire codice.

Q: Come si può prevenire il thrashing?

A: Limitando la multiprogrammazione o usando il Locality Model e Working Set Model per allocare frame sufficienti ai processi.

Q: Cosa rappresenta il Working Set di un processo?

A: L'insieme delle pagine usate nelle ultime Δ operazioni di memoria, rappresentando una stima della località attiva del processo.

Q: In cosa consiste l'algoritmo Page-Fault Frequency (PFF)?

A: Monitora la frequenza dei page fault: se è alta, aumenta i frame al processo; se è bassa, rimuove le pagine non usate.

Q: Perché la memoria del kernel viene allocata separatamente?

A: Perché il kernel ha bisogno di allocazioni contigue e oggetti piccoli e dinamici, per cui usa sistemi come buddy system e slab allocator.

Q: Come funziona il buddy system?

A: Divide la memoria in blocchi di potenze di due e li suddivide ricorsivamente finché trova un blocco adeguato; al rilascio, i buddy adiacenti possono essere uniti (coalescing).

Q: Cos'è la slab allocation?

A: È un metodo per allocare oggetti kernel in cache preinizializzate (slab), evitando frammentazione e migliorando le performance.

Q: Qual è lo scopo del dirty bit?

A: Indica se una pagina è stata modificata: se non lo è, può essere rimossa senza salvarla su disco durante un page replacement, riducendo overhead.

Q: Cosa sono gli stack utente e kernel in OS161?

A: Ogni thread ha due stack: uno per l'esecuzione utente, uno per l'esecuzione kernel (privilegiata).

Q: Cosa contiene il PCB (Process Control Block)?

A: Nome, numero thread attivi, spinlock, puntatore a addrspace del processo.

Q: Quali sono i passaggi per eseguire un programma in OS161?

A: proc_create_runprogram → thread_fork → runprogram (creazione spazio indirizzi, ELF, passaggio al codice utente).

Q: Come avviene una system call?

A: L'utente genera un'interruzione, il kernel esegue la syscall e ritorna in user mode.

Q: Cos'è il demand paging?

A: Tecnica che carica in memoria solo le pagine necessarie all'esecuzione corrente del processo.

Q: A cosa serve la free-frame list?

A: Tiene traccia dei frame liberi da allocare in caso di page fault.

Q: Cosa accade in caso di page fault con frame liberi?

A: Il frame libero viene assegnato e la pagina viene caricata dal disco.

Q: Cosa accade in caso di page fault senza frame liberi?

A: Si sceglie un victim frame da liberare (page replacement), si aggiorna la page table e si carica la pagina richiesta.

Q: Cos'è il copy-on-write?

A: Tecnica in cui processi padre e figlio condividono pagine finché non vengono modificate.

Q: Qual è la formula dell'EAT?

A: EAT = (1 - p) * ma + p * page fault time, con p = probabilità di page fault.

Q: Cos'è l'algoritmo FIFO di page replacement?

A: Sostituisce la pagina più vecchia in memoria (prima entrata).

Q: Cos'è l'anomalia di Belady?

A: Con FIFO, più frame possono causare più page fault: comportamento controintuitivo.

Q: Cos'è l'algoritmo Ottimale di page replacement?

A: Sostituisce la pagina che non sarà usata per il periodo di tempo più lungo.

Q: Quanti bit servono per identificare la pagina con un address space a 64 bit e pagine da 4KB?

A: Servono 52 bit per identificare la pagina (64 - 12).

Q: Quante voci avrebbe una page table a due livelli per 64-bit?

A: Primo livello: 2^42 voci; Secondo livello: 2^10 voci. Troppo grande da gestire.

Q: Come si calcolano le voci in una page table convenzionale?

A: Numero voci = Virtual address space / Page size.

Q: E come si calcolano le voci in una Inverted Page Table?

A: Numero voci = Physical memory / Page size.

Q: Qual è la formula dell'EAT con TLB?

A: EAT = hit_ratio × TLB_access + miss_ratio × (TLB_access + memory_access × livelli_page_table)

Q: Esempio: 21-bit VA, 16-bit PA, 2KB pagine. Quante pagine logiche?

A: 2^21 / 2^11 = 2^10 = 1024 pagine logiche.

Q: Esempio: 21-bit VA, 16-bit PA, 2KB pagine. Quante pagine fisiche?

A: 2^16 / 2^11 = 2^5 = 32 pagine fisiche.

Q: Esempio: 32-bit VA, 512MB RAM, pagine da 4KB. Quante voci nella page table?

A: Page table convenzionale: 2^20 voci; IPT: 2^17 voci.

Q: Qual è il vantaggio dell'IPT?

A: Una sola tabella per tutti i processi; riduce l'uso di memoria.

Q: E lo svantaggio dell'IPT?

A: Richiede ricerche lente; migliorabile con hashing o TLB.

Q: Come funziona una hashed page table?

A: Usa una funzione di hash sull'indirizzo virtuale per accedere alla tabella; collisioni gestite con liste concatenate.

Q: Cosa accade in uno swap standard?

A: L'intero processo viene spostato nel disco (backing store).

Q: Qual è il vantaggio dello swapping con paging?

A: Solo alcune pagine vengono spostate, riducendo il tempo di swap.

Q: Qual è lo scopo del file system?

A: Gestire l'archiviazione, l'accesso e la protezione dei file su dispositivi secondari.

Q: Quali sono alcune strutture del file system?

A: Directory, attributi, spazio libero, journaling, controlli di accesso.

Q: Cosa sono le system call?

A: Un'interfaccia per richiedere servizi al kernel.

Q: Differenza tra mutex e semaforo?

A: Il mutex consente accesso esclusivo, il semaforo può controllare accesso multiplo con contatore.

Q: Come viene garantita la protezione della memoria?

A: Tramite i registri base e limit: se l'indirizzo non rientra, si genera un trap.

Q: Cosa fa la MMU?

A: Traduce dinamicamente gli indirizzi logici in fisici aggiungendo l'indirizzo di rilocazione.

Q: Differenza tra indirizzo logico e fisico?

A: Logico è generato dalla CPU, fisico è usato dalla RAM.

Q: Quali sono i principali algoritmi di allocazione dinamica?

A: First-fit, best-fit, worst-fit.

Q: Cos'è la frammentazione interna?

A: Parte di memoria allocata ma non utilizzata (es. fine pagina).

Q: Cos'è il paging?

A: Tecnica che divide memoria logica in pagine e memoria fisica in frame, per evitare frammentazione esterna.

Q: Vantaggi del paging?

A: Elimina frammentazione esterna, supporta esecuzione non contigua.

Q: Cos'è l'offset in un indirizzo logico?

A: È la posizione interna alla pagina da cui si accede.

Q: Come lavora la MMU con il paging?

A: Estrae la pagina, consulta la page table per il frame, costruisce l'indirizzo fisico combinando frame + offset.

Q: Con paging, perché si ha frammentazione interna?

A: Perché l'ultima pagina può essere solo parzialmente utilizzata.

Q: Qual è il vantaggio principale del paging?

A: Permette gestione della memoria non contigua, eliminando la frammentazione esterna.

Q: Quali strutture compongono la memoria nel paging?

A: Pagine nella memoria logica e frame nella memoria fisica.

Q: A cosa serve la page table?

A: Mappa le pagine logiche ai frame fisici per ciascun processo.

Q: Come viene diviso un indirizzo logico nel paging?

A: In Page Number (indice nella page table) e Page Offset (posizione nella pagina).

Q: Qual è il ruolo del PTBR?

A: Contiene l'indirizzo base della page table per il processo corrente.

Q: Cos'è la TLB?

A: Una cache hardware per le traduzioni indirizzi, accelera l'accesso alla page table.

Q: Cosa accade in caso di TLB miss?

A: Si accede alla page table in memoria, con penalità di tempo.

Q: Come si proteggono le pagine di memoria?

A: Tramite bit di protezione nella page table (es. read-only) e valid-invalid bit.

Q: Cosa sono le pagine condivise?

A: Pagine di codice condivise da più processi, mentre i dati restano separati.

Q: Qual è il problema della page table lineare?

A: Richiede troppa memoria contigua (es. 4 MB per processo).

Q: Qual è la soluzione alla page table lineare?

A: Usare una page table gerarchica (multi-livello).

Q: Come funziona una page table a due livelli?

A: L'indirizzo logico è diviso in tre parti: indice esterno, indice interno e offset.

Q: Qual è la causa della frammentazione interna nel paging?

A: L'ultima pagina può essere parzialmente inutilizzata.

Q: Quanto spazio si spreca mediamente per frammentazione interna?

A: In media, mezza pagina per processo.

Q: Quali sono gli effetti di avere pagine piccole?

A: Meno frammentazione interna, ma più overhead (più entry nella page table).

Q: Quali sono gli effetti di avere pagine grandi?

A: Più frammentazione interna, ma meno overhead di gestione.

Q: Quali sono gli stati principali di un thread?

A: Running, Ready, Blocked/Waiting, Terminated.

Q: Cosa contiene il TCB di un thread?

A: Stack, PC, registri CPU, stato, e altre info per ripristino.

Q: Cos'è un context switch?

A: È il salvataggio dello stato del thread attivo e il caricamento di quello del thread successivo.

Q: Quali sono i campi chiave di struct thread in OS/161?

A: t_name, t_state, t_stack, t_context, t_cpu, t_proc, t_wchan_name.

Q: Qual è la relazione tra thread, processo e CPU?

A: Ogni thread appartiene a un processo; t_cpu indica il core dove è attivo.

Q: Cosa fa thread_fork()?

A: Crea un nuovo thread, assegnandogli una funzione iniziale, nome e processo.

Q: Cosa fa thread_yield()?

A: Il thread attivo cede volontariamente la CPU.

Q: Cosa fa thread_exit()?

A: Termina il thread e libera le sue risorse.

Q: Cosa fa thread_consider_migration()?

A: Valuta la migrazione del thread su un altro core.

Q: Cosa fa thread_make_runnable()?

A: Inserisce un thread nella coda dei pronti all'esecuzione.

Q: Cosa fa thread_switch()?

A: Gestisce il salvataggio e il caricamento del contesto tra due thread.

Q: Cos'è switchframe_init()?

A: Prepara la struttura dati con lo stato dei registri CPU.

Q: Cos'è switchframe_switch()?

A: Funzione assembly che salva e carica i registri durante il cambio di contesto.

Q: Perché OS/161 usa assembly per il context switch?

A: Perché solo in assembly si possono manipolare direttamente i registri della CPU MIPS.