***DOMANDE TEORIA SISTEMI OPERATIVI***

***(DOMANDA 17) Descrivi quali sono le differenze tra un semaforo spinlock e uno che non lo è. Descrivi un uso concreto dei spinlock da parte del kernel di un sistema operativo.***

Un semaforo spinlock è un semaforo che si basa su busy waiting, cioè blocca il processo in un loop infinito fino al verificarsi di una determinata condizione che lo libererà e gli permetterà di eseguire le istruzioni successive. Questo tipo di semaforo è "dannoso" in quanto consuma molta CPU (a causa delle risorse continuamente utilizzate all'interno del loop infinito).

Un semaforo normale invece non blocca il processo in un loop infinito ma mentre un processo è in attesa, le risorse vengono restituite al sistema in modo da non causare un elevato utilizzo di CPU

La seconda parte dei semafori l'ho imparata quasi a memoria da degli appunti che ho trovato (dato che sugli appunti del prof non spiega la differenza o comunque non accenna alle risorse liberate)

***Illustrare brevemente gli algoritmi SSTF e SCAN per lo scheduling degli accessi a un disco rigido.***

SSTF: Con l’algoritmo SSTF la testina del disco rigido viene spostata alla traccia più vicina a quella corrente fra quelle finora richieste. Il problema principale di questo algoritmo è che si va incontro al rischio di starvation, ossia si rischia che le tracce più esterne (0 e MAX) vengano servite con un ritardo molto alto.

SCAN: Per ovviare al problema della starvation, si può usare questo algoritmo che può essere svolto in due direzioni, la prima spostando la testina del disco verso le tracce superiori a quella corrente, oppure, al contrario, spostando la testina verso le tracce inferiori a quella corrente. Con questo algoritmo, la testina viene spostata prima in una direzione finché ci sono richieste, poi cambia direzione e così via. La direzione è controllata da un bit di stato.

***Mostrare una sequenza di richieste per la quale l’algoritmo SSTF porta a starvation di una delle richieste.***

La sequenza delle seguenti richieste porta a starvation:

(100, 120, 80, 0, 70, 110, 90, 115, 130, …)

Infatti la richiesta di servire la traccia 0 rimane per tanto tempo in attesa

***(DOMANDA 18) Fornire una ragione (in termini intuitivi) per la quale SCAN non può portare a starvation di una richiesta. Quale potrebbe essere il caso peggiore (tempo di attesa massimo per una richiesta)?***

Il caso peggiore, ossia quello di attesa massimo per una richiesta, avviene quando è appena stata cambiata la direzione della testina e il dato che arriva è precedente a quello che è appena stato processato. In questo caso, tale dato dovrà aspettare tutta la lettura delle tracce in salita più la lettura delle tracce in discesa (o viceversa) prima di essere servito. Un esempio è quando si arriva alla traccia 0, si cambia direzione e mentre si sta processando la traccia 1 arriva la richiesta da parte di 0. In questo caso la testina deve prima arrivare alla traccia più grande (MAX) e poi deve tornare indietro. Così facendo la traccia 0 deve aspettare tutta una lettura in salita più tutta una lettura in discesa.

Considerando l’esempio precedente, partiamo da una situazione di SCAN di tracce in ordine decrescente a partire dalla traccia 50, considerando che più richieste alla stessa traccia, se servite consecutivamente, non richiedano tempi aggiuntivi. Supponendo un *tseek = 1ms/traccia*, abbiamo che nel caso peggiore possiamo avere una richiesta per ogni pagina *i* con *0 <= i <= 50*.

Arrivati alla pagina 0, si sarà costretti a invertire la rotta; quindi potrà essere visitata anche la traccia 210. Questo dimostra che l’algoritmo di SCAN non ha rischi di starvation (è facilmente dimostrabile anche per il caso dello SCAN crescente).

In questo caso particolare al massimo dopo si avrà l’inversione, dopodichè verranno soddisfatte le richieste in ordine fino a quella della traccia 210, quindi nel caso peggiore a partire dalla situazione iniziale (dalla traccia 50), il tempo massimo di attesa per la richiesta di visita alla traccia 210 sarà di *260 ms*.

***Discutere brevemente i vantaggi e gli svantaggi dei tre metodi di allocazione dello spazio disco considerati a lezione (contigua, a lista concatenata, indicizzata) nei seguenti scenari:***

***1. Accesso sequenziale in sola lettura a un solo grande file (esempio: compilazione di un grosso file sorgente)***

L'allocazione di spazio contigua permette un accesso sequenziale più veloce. Questo dovuto al fatto che allocando porzioni di memoria contigue è sufficiente avere come riferimento il primo blocco della serie e la lunghezza di quest'ultima. Lo svantaggio è che per file di grandi dimensioni è necessario trovare sufficienti

blocchi di memoria contigui che riescano ad allocarlo.

L'allocazione di spazio a lista concatenata permette di allocare il file in maniera più semplice dato che non è più necessario fare riferimento a blocchi di memoria

contigui. In questa situazione, l'unico svantaggio è che se un blocco dovesse corrompersi, i successivi blocchi risulterebbero irraggiungibili. L'unica soluzione è realizzare una lista doppiamente concatenata.

L'allocazione indicizzata in questa situazione può comportare l'occupazione di grandi aree di memoria da parte della tabella degli indici, andando così a limitare la possibile dimensione del file. Nonostante ciò, per effettuare un accesso sequenziale è sufficiente scorrere la tabella in quanto già ordinata.

***2. Accesso casuale in sola lettura a un solo grande file (esempio: tabella di database)***

L'allocazione contigua risulta essere molto efficiente per gli accessi casuali dato che è sufficiente avere il riferimento del primo blocco del file e la lunghezza (vale a dire il numero di blocchi in cui è memorizzato il file). L'unico svantaggio è dato dalle dimensioni del file: il sistema impiegherà più tempo per allocare file di grandi dimensioni dato che sarà necessario trovare un sufficiente numero di blocchi contigui che lo riescano a contenere.

L'allocazione di spazio a lista non permette l'accesso casuale a causa della distribuzione casuale dei blocchi in cui è memorizzato il file. Per questo motivo, è possibile effettuare solo accessi sequenziali, cioè è necessario scorrere tutta la lista (o parte di essa) per raggiungere il blocco desiderato.

L'allocazione indicizzata ha il vantaggio di essere molto efficiente in accessi casuali dato che l'i-esimo blocco avrà indice i-esima. L'unico svantaggio è la dimensione del file: file molto grandi richiederanno molto spazio per la creazione della tabella degli indici .

***3. Operazioni di scrittura in append su un solo un grande file (esempio: log di un server web molto trafficato)***

Per poter effettuare un append in un file allocato in maniera contigua, è sufficiente prendere in considerazione il blocco iniziale, sommargli la lunghezza e aggiungere il blocco in tale posizione. Lo svantaggio è che il blocco da allocare che viene subito dopo all'ultimo blocco allocato del file potrebbe essere già occupato e questo comporterebbe lo spostamento dell'intero file in un'area di memoria adeguata.

L'append per un file allocato a lista risulta essere particolarmente semplice: basta infatti trovare un blocco libero e aggiungerlo a fine lista. Per farlo è necessario che l'ultimo blocco punti al nuovo blocco e che la directory contenente il file punti al blocco appena aggiunto (oltre che al primo blocco del file). Rimane comunque la possibilità che un blocco risulti corrotto causando l'inaccessibilità dei successivi blocchi.

L'append per un file allocato tramite indici è particolarmente semplice: basta aggiungere un nuovo indice in fondo alla tabella degli indici dato che è ordinata. Rimane comunque il problema che per file molto grandi, la tabella degli indici occuperà molto spazio

***4. Creazione e cancellazione ripetuta di molti piccoli file (esempio: utilities del sistema operativo).***

La creazione/cancellazione ripetuta di piccoli file allocati in porzioni di memoria contigue non presenta particolari problemi date le piccole dimensioni dei files.

Infatti, trovare blocchi di memoria vicini in grado di contenere questi files risulta essere abbastanza semplice. Nonostante ciò, questo tipo di allocazione contribuirà

ad aumentare la frammentazione esterna e nel caso in cui i files dovessero crescere in dimensione, sarà necessario riallocarli (abbastanza difficile se i file sono molto grandi

dato che aumenta la difficoltà di poter trovare blocchi di memoria contigui sufficicentemente grandi da poterli contenere).

La cancellazione/creazione ripetuta di piccoli file allocati tramite lista non presenta particolari problemi dato che è necessario individuare in tutta la memoria un numero sufficiente di blocchi che possa contenerli (questi infatti non devono essere necessariamente contigui). Rimane comunque la possibilità che un blocco risulti corrotto causando l'inaccessibilità dei successivi blocchi.

La cancellazione/creazione ripetuta di piccoli file allocati tramite lista indicizzata non presenta particolari problemi dato che si tratta di individuare in tutta la memoria un numero sufficiente di blocchi che possa contenerli (questi infatti non devono essere necessariamente contigui) e aggiungerli in ordine alla tabella degli indici. Trattandosi di files di piccole dimensioni, non ci sarà neanche il rischio che la tabella degli indici occupi un grande spazio in memoria. Risulta essere svantaggiosa se dovessero essere creati molti file di piccole dimensioni, in quanto ognuno richiederebbe una propria index table e dunque molti blocchi verrebbero sprecati per far spazio alle tali tabelle.

***(DOMANDA 19) Consideriamo un sistema a paginazione su richiesta (on-demand paging), in cui il grado di multiprogrammazione (numero di processi che si alternano nell’uso della CPU) è attualmente 4. Per ciascuno dei seguenti esiti della misurazione dell’utilizzo della CPU e del disco di swap, illustrare cosa sta verosimilmente accadendo, discutere se la paginazione sia d’aiuto o meno e cosa si può fare per aumentare l’uso della CPU.***

***1. Utilizzo della CPU: 15%; occupazione del disco di swap: 95%.***

Situazione abbastanza negativa perché nel disco di swap abbiamo tanti indirizzi ed è quasi pieno, mentre la CPU sta svolgendo pochi processi. Il computer potrebbe essere in uno stato di crashing, tutti i processi potrebbero essere in attesa di un frame lasciando la cpu inutilizzata. Lo scheduler a lungo termine vedendo la cpu poco utilizzata aggiungerebbe altri processi mandando il sistema in blocco.

***2. Utilizzo della CPU: 90%; occupazione del disco di swap: 5%.***

Situazione ottimale (è più o meno la situazione opposta a quella di prima). In questo caso, i processi sono soddisfatti della loro porzione di memoria e questo causa una bassa percentuale di swap (perchè non sono in attesa di frame) e un'alta percentuale di utilizzo di cpu dato che, appunto, essendo soddisfatti della memoria a disposizione sono liberi di essere processati dalla cpu.

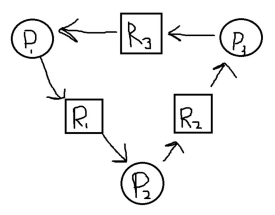
***3. Utilizzo della CPU: 15%; occupazione del disco di swap: 5%.***

Situazione tranquilla dove i processi non stanno “stressando” la CPU, infatti si hanno pochi processi in esecuzione e un disco di swap quasi inutilizzato.

In generale io confronto le due percentuali, se la % di cpu è maggiore rispetto a quella di swap, allora lo considero come una situazione positiva: i processi sono soddisfatti della memoria che hanno a disposizione e la cpu li sta processando correttamente. Al contrario, se la % di swap supera quella della cpu, i processi sono bloccati in attesa di frame causando così un sottoutilizzo della cpu

***(DOMANDA 20)***

***1. Presentare una condizione di deadlock che coinvolga almeno tre processi in attesa di risorse ad accesso mutuamente esclusivo. Rappresentare graficamente la situazione utilizzando un grafo di allocazione delle risorse***



***2. Descrivere una sequenza di richieste che, a partire da una situazione iniziale a risorse libere, porta alla situazione di deadlock descritta al punto 20.1.***

* P1 richiede R3 e la ottiene
* P2 richiede R1 e la ottiene
* P3 richiede R2 e la ottiene
* P1 richiede R1
* P2 richiede R2
* P3 richiede R3

Essendo R1,R2 e R3 già assegnate siamo in una situazione di deadlock.

***3. Descrivere sommariamente (a parole o pseudocodice) un algoritmo per evitare queste situazioni di deadlock. Chiarire su quali assunzioni si basa, e applicarlo alla sequenza descritta al punto 20.2 mostrando come il deadlock viene evitato.***

*Algoritmo del banchiere*

***(ESAME 12/07/2021)***

***Si spieghi in dettaglio cos'è e come viene utilizzata la memoria TLB***

La TLB è una tabella mantenuta dalla MMU e di dimensioni inferiori rispetto ad una normale page table. Il suo compito è quello di memorizzare una serie di chiavi/valori, dove una chiave è un numero di pagina e il valore è l’indirizzo del frame corrispondente. Viene utilizzata per evitare che la CPU debba compiere due accessi alla memoria prima di risalire all’effettivo indirizzo di un frame. Si parla di “page hit” quando la CPU trova il frame nella TLB, mentre si parla di “page miss” quando il frame non è memorizzato nella tabella ed è quindi necessario accedere all’intera page table per trovarlo.

***Spiega in dettaglio cosa è l’i-node e che informazioni contiene***

L’i-node è una particolare tipologia di blocco utilizzato nei sistemi UNIX per puntare ai file salvati in memoria. Un i-node è organizzato a lista in cui le prime *n* righe puntano ai primi blocchi fisici in cui è memorizzato il file per velocizzarne l’accesso, mentre le restanti righe sono puntatori a tabelle di livelli successivi. Un i-node contiene tutte le informazioni di un file tranne che il nome

***(ESAME 21/07/2021)***

***Si descriva in dettaglio il concetto di paginazione***

La paginazione è una tecnica di allocazione di memoria molto diffusa che permette di allocare spazio che non sia necessariamente disposto in porzioni di memoria contigue. La memoria viene suddivisa in frame di stessa dimensione e ad ogni processo verrà assegnato un numero di frame pari a s/f dove s è lo spazio totale richiesto dal processo e f la dimensione di un singolo frame. Per poter rendere queste porzioni di memoria non contigue accessibili, la MMU crea per ogni processo una page table, ovvero una tabella di indirizzi. Il suo compito è quello di associare ogni indirizzo logico generato dalla CPU a corrispondente indirizzo fisico di uno dei frame associati al processo. Per farlo, i bit più significativi dell’indirizzo logico identificano una pagina della tabella (riga), mentre quelli meno significativi un offset da dover applicare all’indirizzo fisico (offset logico coincide con offset fisico). Interrogando la page table è dunque possibile risalire all’indirizzo fisico del frame cercato a cui dovrà poi essere applicato l’offset trovato precedentemente.

***Spiegare in dettaglio quali sono i casi di frammentazione interna ed esterna che possono avvenire usando la paginazione***

Usando la tecnica di paginazione, si riduce la paginazione esterna in quanto è possibile occupare porzioni di memoria non contigue e tutte aventi la stessa dimensione. Per quanto riguarda la frammentazione interna invece, potrebbe capitare che il numero di frame concessi al processo venga arrotondato per eccesso: se s/f non restituisce un numero intero, non è possibile assegnare ad un processo una porzione più piccola di un frame e dunque gli verrà assegnato un numero di frame pari all’intero successivo. In questi casi si può verificare frammentazione interna dato che il processo non finirebbe per occupare tutta la memoria a lui dedicata

***Che cos’è il deadlock? Quali sono le condizioni necessarie e sufficienti perché si possa verificare il deadlock?***

Un deadlock è una situazione di blocco che si viene a verificare quando due processi si tengono bloccati a vicenda attendendo una o più risorse che detiene l’altro processo. Le condizioni necessarie e sufficienti affinché si verifichi un deadlock sono: le risorse con cui stanno interagendo i processi richiedono accesso esclusivo, le risorse possono essere rilasciate solo dal processo (il sistema operativo non può forzarne il rilascio), condizione di hold and wait in cui detengono una risorsa e ne stanno aspettando un’altra e infine attesa circolare.

***(ESAME 28/06/2021)***

***Si descrivano le fasi necessarie per trasformare un programma in processo. Si spieghi quindi il concetto di binding degli indirizzi e infine si presentino le differenti tipologie di collegamento (linking) e caricamento (loading).***

Le fasi necessarie per trasformare un programma in un processo sono:

* Un compilatore prende in carico il nostro programma, lo compila e restituisce un programma in linguaggio assembly;
* Un assembler prende il programma in assembly e ci restituisce un file oggetto in linguaggio macchina;
* Attraverso il linker, il nostro file oggetto viene unito alle librerie (anch’esse in linguaggio macchina) e viene restituito un unico file eseguibile;
* Questo file eseguibile, successivamente, viene preso dal loader che lo carica in memoria e da qui diventa un processo.

Con il Binding Dinamico, la CPU ha una visione della memoria basata su indirizzi logici che vengono tradotti dalla MMU durante la fase di esecuzione ed essa produce un indirizzo fisico che punta alla porzione di memoria nella quale si trova il dato che ci serve.

Con il Binding Statico, invece, gli indirizzi sono fissati nella fase di compilazione/linking o in sede di caricamento. La CPU emette gli indirizzi fisici di memoria

***Si descrivano le differenze principali tra una thread e un processo, indicando un esempio pratico in cui l’uso di thread è conveniente rispetto all’uso dei processi.***

I processi sono le unità di possesso delle risorse, in particolare, un processo è associato ad uno spazio di indirizzamento e ad un elenco di risorse.

I thread sono le unità di utilizzo della CPU, uno stesso processo può essere associato a più thread, in particolare, un thread è associato ad uno stato di esecuzione, ad un insieme di registri CPU e ad uno stack. Essi non sono indipendenti tra loro e condividono il codice, i dati e le risorse dal sistema assegnate al processo di appartenenza.

Possiamo dire che un thread è un flusso di esecuzione indipendente all’interno di un processo che condivide lo spazio di indirizzamento con gli altri thread del processo. Esso è rappresentato da un TCB (thread control block) che punta al PCB (process control block) del processo contenitore.

***(ESAME 25/08/2020)***

***Descrivere in dettaglio come funzionano e le differenze tra paginazione e segmentazione***

Paginazione: la memoria viene suddivisa in segmenti di uguale dimensione… (vedi sopra)

Segmentazione: cerca di mantenere la corrispondenza fra l’allocazione della memoria virtuale e la visuale che l’applicazione ha della stessa. Un processo viene visto come una collezione di segmenti in memoria, ciascuno dei quali ha uno scopo specifico. I segmenti possono variare di dimensione e sono determinati in base allo spazio degli indirizzi richiesto dal processo. L’indirizzo logico generato dalla CPU è una coppia:

(numero di segmento, offset)

Come per la paginazione, la MMU mantiene un STRB (segment table base register) e un STLR (segment table length register) che sono contenuti nel PCB e caricati ad ogni cambio di contesto.

I vantaggi della paginazione sono che non c’è frammentazione esterna, mentre nella segmentazione c’è molta frammentazione esterna; al contrario, nella paginazione c’è frammentazione interna (anche se minima), mentre nella segmentazione questo problema non si presenta. Inoltre, nella paginazione, l’allocazione dei frame è libera, mentre nella segmentazione servono degli algoritmi ben precisi di allocazione. Entrambi i metodi però hanno il vantaggio della protezione e della condivisione.

***Si spieghi in dettaglio come è implementata la tabella delle pagine invertita***

L’inverted page table (o tabella della pagine invertita), è una delle soluzioni alla proliferazione della page table a più livelli. La page table invertita mappa ogni frame sui processi che li possiedono e alle porte logiche a cui sono associati. Essa è una tabella di sistema, con una riga per ogni frame della memoria fisica, contenente la coppia *<pid, page>* alla quale il frame è associato. L’indirizzo logico è del tipo:

(pid, page, offset)

è una tabella con una entry per ogni frame, non per ogni page. Ogni entry consiste nel numero della pagina (virtuale) memorizzata in quel frame, con informazioni riguardo il processo che possiede la pagina. La IPT diminuisce la memoria necessaria per memorizzare le page table, ma aumenta il tempo di accesso alla tabella.

***(ESAME 15/07/2019)***

***Spiegare in dettaglio che cos'è il trashing, perché si verifica e le possibili misure per mitigarlo/eliminarlo***

Il trashing è un fenomeno che si verifica quando la domanda di frame da parte dei processi supera la disponibilità dei frame stessi. Questo comporta ad un sottoutilizzo della CPU (processi in attesa a causa dell’insufficienza dei frame senza i quali non possono proseguire nella loro esecuzione) che porta lo scheduler a lungo termine a caricare sempre più processi in memoria a causa del sottoutilizzo della CPU, contribuendo alla situazione di crash del sistema. Per ovviare a questo problema, vengono fissati un limite superiore di page fault ratio (ritenuto accettabile) e uno inferiore (cioè la migliore condizione concessa in cui si possa trovare un processo in termini di possedimento frame), in modo da consentire ad ogni processo di non avere né troppi frame né troppi pochi.

***Spiegare la differenza tra scheduler a lungo termine e scheduling a breve termine***

Lo scheduler a breve termine sceglie tra i processi in stato di Waiting, il prossimo da portare in stato Running. Questo tipo di scheduler viene invocato molto di frequente e deve essere veloce e non consumare molta CPU.

Lo scheduler a lungo termine, invece, sceglie quali processi avviare e portare in memoria principale. Esso viene invocato meno di frequente e determina il grado di multiprogrammazione del sistema, cioè il numero di processi in memoria.

***Si descrivano in dettaglio i passi necessari per attivare la comunicazione tra due processi mediante memoria condivisa in UNIX. Come avviene la rimozione dell'area di memoria condivisa?***

La condivisione della memoria è utile per mettere in comunicazione grandi quantità di dati e strutture dati complesse. Una regione di memoria condivisa risiede nello spazio di indirizzamento del processo che ne richiede la creazione, mentre gli altri processi che ne vogliono fare uso devono disporre di un meccanismo per inserirla anche nel proprio spazio, aggirando i meccanismi di protezione. Fatto ciò i processi possono comunicare e scambiare informazioni scrivendo e leggendo la memoria.

***(ESAME 19/06/2019)***

***Cosa si intende per “race condition”? Rispondere servendosi di un esempio***

Si parla in particolare di race condition laddove il contenuto di un’area di memoria condivisa o il valore letto da questa dipende dall’ordine di esecuzione dei processi. Ciò è chiaramente un problema in quanto fa sì che il codice non sia più univoco nelle operazioni svolte e nei risultati generati a parità di input (matematicamente non sarebbe più una funzione). La sezione di codice contenente le istruzioni di accesso alle risorse che pongono il sistema in race condition è chiamata sezione critica. Un esempio è dato da due processi che devono incrementare una stessa variabile. Supponiamo quindi che i due processi effettuino questa operazione contemporaneamente e che il valore iniziale della variabile sia 0. Il primo processo prenderebbe il valore iniziale della variabile, cioè 0, lo incrementerebbe di 1 e riscriverebbe il nuovo risultato nella variabile. La stessa cosa fa però il secondo processo che, anzichè prendere come valore della variabile quello modificato dal precedente processo, prenderà nuovamente 0. Dunque, il valore finale della variabile a seguito di un doppio incremento simultaneo e non gestito da parte di processi distinti sarà 1 e non 2.

***Spiega in dettaglio il concetto di paginazione in un contesto di memoria virtuale***

Con la memoria virtuale è possibile considerare parte del disco come memoria primaria e caricare in memoria solo le parti del programma necessarie per l’esecuzione attuale. Sebbene generalmente complessa da implementare, la memoria virtuale permette di caricare più processi in memoria, aumentando quindi il throughput, ed inoltre permette facilmente la condivisione di risorse fra processi. Una pagina viene caricata in memoria solo all’occorrenza. In particolare, se un processo dovesse richiedere una pagina non presente in memoria, il processo viene messo in uno stato blocked è aggiunto ad una coda specifica, viene identificato un frame libero per la pagina da memorizzare (il suo caricamento può richiedere del tempo, dunque altri processi verranno processati nel mentre), viene aggiornata la page table e infine viene sbloccato il processo e ripetuta l#istruzione che aveva causato page fault.

***Descrivere la tecnica del buddy system per l’allocazione di memoria ai processi***

È una tecnica utilizzata per allocare memoria ai processi. In particolare, viene definito un limite superiore pari a 2^u che corrisponde alla memoria massima e un limite inferiore 2^l, cioè la minima memoria che può essere allocata. Se un processo richiede memoria pari a s, viene effettuata una ricerca su una lista di blocchi di dimensioni pari a 2^k, dove s<= 2^k. Se la lista è vuota allora la ricerca viene effettuata per blocchi leggermente più grandi 2^k+1 e così via fino a quando non si riesce ad allocare memoria al processo. Se un processo libera memoria e il buddy liberato ha vicino un altro buddy della stessa dimensione, allora vengono uniti e aggiunti alla lista di blocchi di dimensione 2^k+1 (supponendo che prima fossero buddies di dimensione pari a 2^k).

***(ESAME 27/08/2018)***

***Spiega in dettaglio come funzionano i tre diversi metodi di accesso ai dispositivi di I/O indicando vantaggi e svantaggi per ciascuno di essi***

L’accesso ai dispositivi di I/O può avvenire o in maniera sincrona, cioè la CPU legge i dati accedendo a specifiche porte di I/O (ad esempio tramite polling), in maniera asincrona, cioè la CPU riceve un interrupt che interrompe quello che sta facendo per poter processare i dati ricevuti dai dispositivi di I/O. Infine, l’accesso ai dati di questi dispositivi può avvenire tramite DMA (Direct Memory Access) ovvero un dispositivo che processa questo tipo di dati nei momenti in cui il bus non viene utilizzato dalla CPU

***(ESAME 23/01/2018)***

***Spiegare in dettaglio cosa è l’attesa attiva (busy waiting). Perché' è un problema e come è possibile risolverlo***

L’attesa attiva (busy waiting) è un tipo di attesa in cui un processo rimane bloccato in un loop infinito fino al verificarsi di una condizione e senza liberare le risorse al sistema. Questo comporta un costante utilizzo di CPU. Una soluzione è offerta dai semafori.

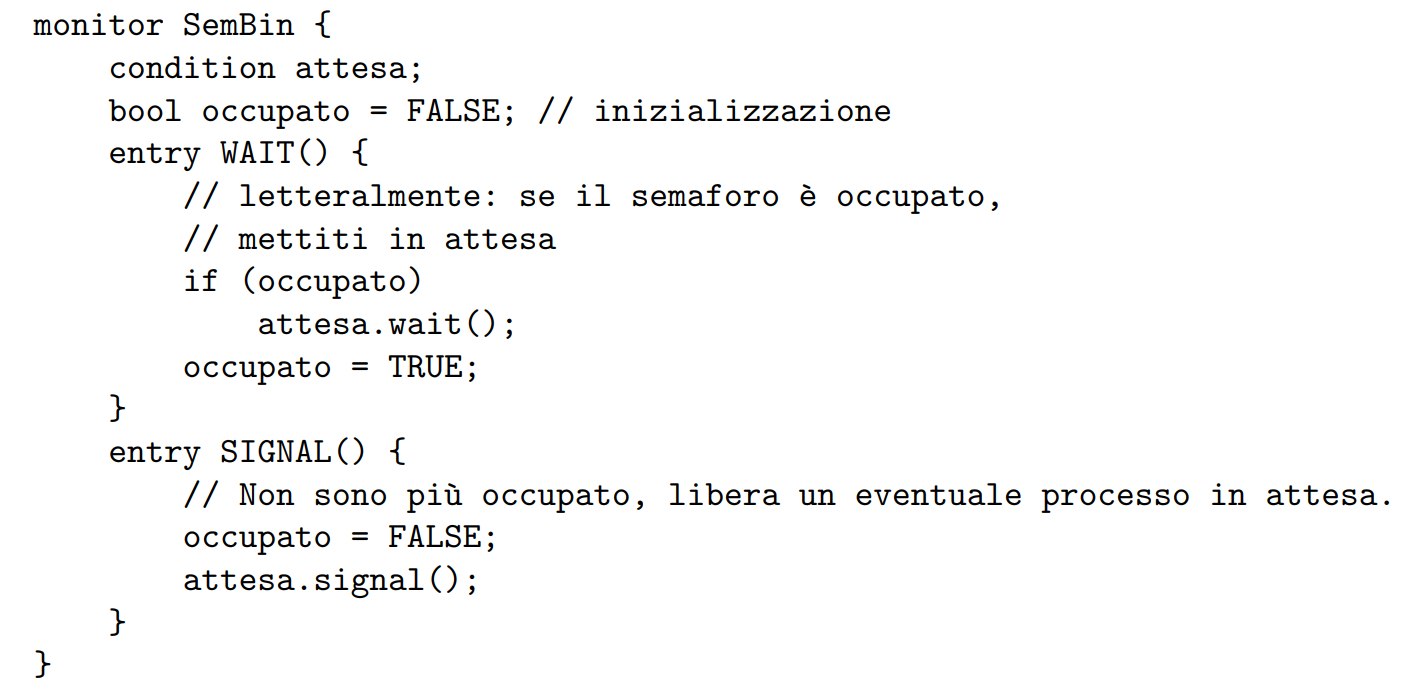
***(ESAME 01/02/2016)***

***Si descrivano in dettaglio le tecniche (indicizzata, lista, contigua) per l’allocazione dello spazio su disco ai file.***

L’allocazione di spazio contigua su disco prevede di riservare un numero di blocchi sufficientemente elevato per la memorizzazione dati, purché essi siano contigui. Questo tipo di allocazione ha alcuni vantaggi come, ad esempio, il fatto che la directory debba puntare solamente al primo blocco dato che disponendo della lunghezza (numero blocchi) è immediato l’accesso ai successivi. Inoltre, rende particolarmente rapidi gli accessi sequenziali oltre che quelli casuali. L’unico svantaggio è la frammentazione esterna: dovendo occupare blocchi contigui di memoria, alcune successioni di blocchi troppo piccoli potrebbero non venire mai occupati. L’allocazione a lista è una tecnica che prevede l’allocazione di blocchi di memoria non necessariamente contigui: la directory punterà al primo e all’ultimo blocco (necessari per l’append) e ogni blocco sarà dotato di un puntatore al successivo. Questa tecnica riduce la frammentazione esterna. I principali svantaggi di questo tipo di allocazione sono l’impossibilità di effettuare accessi casuali (è necessario scorrere tutta la lista e fermarsi al punto/blocco desiderato) e il rischio che se un blocco dovesse essere corrotto, i successivi blocchi non sarebbero più raggiungibili (soluzione: lista doppiamente concatenata). L’allocazione indicizzata prevede l’utilizzo di una tabella in cui l’i-esimo blocco si trova all’i-esima riga. Eredita i vantaggi descritti nelle due tecniche precedenti. L’unico svantaggio è la dimensione: per file molto grandi, la tabella degli indici inizierà ad avere un peso non indifferente.

***(ESAME 20/06/2012)***

**Si implementi il funzionamento delle primitive di sincronizzazione wait e signal di un monitor usando i semafori. Suggerimento: si consideri una variabile condition x, e si usino i semafori per scrivere lo pseudo-codice che verrebbe eseguito chiamando x.wait e x.signal.**



***Si descrivano le diverse tipologie di architettura***

* STRUTTURA MONOLITICA: dove è presente un unico modulo con un solo strato software tra l’hardware e l’utente. Un modello simile è quello usato da UNIX, dove il numero ridotto di strati e componenti distanti rende il sistema difficile da mantenere ma riduce l’overhead al minimo
* STRUTTURA A STRATI: questa tipologia sfrutta il “Principio del Minimo Privilegio”, dove ogni componente dovrebbe avere il minimo privilegio necessario per svolgere la propria funzione. Solitamente viene usata una struttura ring a 4 strati, dove, per esempio, lo scheduler dei processi dovrà probabilmente avere tutti i privilegi necessari per l’accesso alle tabelle di paginazione della memoria; mentre per una semplice scrittura sul disco sarà probabilmente necessario solamente l’accesso al sottosistema I/O. Alcune interpretazioni pongono al centro l’hardware e nei livelli intermedi mettono l’hardware. Avremo una sicurezza maggiore ma un overhead più alto
* STRUTTURA A MICROKERNEL, in questa tipologia, il kernel viene ridotto a una piccola collezione di componenti essenziali (scheduling, gestione della memoria), riducendo appunto a microkernel. Una componente importante è quella che coordina i vari componenti a livello utente attraverso l’astrazione del message passing. Questi sistemi sono molto protetti dagli errori ma il loro overhead è sempre più alto.
* STRUTTURA A MODULI, con questa architettura, si conservano i vantaggi dell’architettura a microkernel, ma si rinuncia alla protezione dello usermode. La modalità è quella di implementare le funzionalità a moduli, ossia componenti software eseguite in kernelmode e che poggiano su un’interfaccia uniforme di chiamate di sistema. Con questa architettura, solo il codice necessario a gestire l’hardware viene caricato in memoria, ma l’overhead dovuto alle varie interazione è sempre maggiore.

***Spiega in dettaglio come funziona la tabella delle Pagine Invertita.***

La tabella delle pagine invertita è una tabella che mappa i frame del sistema sui vari processi e pagine che li possiedono. Una page-table invertita quindi avrà un numero di righe pari ai frame del sistema e ogni riga sarà composta dalla terna

Il compito della MMU sarà quello di, data la coppia , trovare il numero di riga corrispondente che rappresenterà il frame corrispondente alla ricerca. Una ricerca sequenziale sarebbe proibitiva, è comunque possibile istituire una funzione hash sulle pagine che ci consentirebbe di invertire ulteriormente la tabella.

***Descrivere cosa sono gli i-node.***

Gli i-node in UNIX sono delle strutture dati del filesystem che serve ad archiviare informazioni e attributi su file e directory.

Si presenta come una lista di indici, dove i primi 10/12 indici sono diretti a blocchi di dati, che permettono l’accesso immediato ai primi blocchi di un file, dopodiché c’è un puntatore a una lista di primo livello, contenente 128 indici (512 KB), e un puntatore a un’altra lista di secondo livello che può contenere fino a 64 MB di memoria.

In queste liste sono contenuti vari attributi e informazioni, ad esempio:

* la dimensione del file e la sua locazione fisica;
* il proprietario e il gruppo di appartenenza;
* le informazioni temporali di modifica, ultimo accesso e di cambio di stato;
* il numero di collegamenti fisici che referenziano l'inode;
* i permessi d'accesso
* un puntatore allo spazio del disco che contiene i file veri e propri.