THREAD

Un thread è un percorso di controllo d'esecuzione all'interno di un processo.

Un thread è l'unita di base d'uso della Cpu e comprende un identificatore di thread, un contatore di programma, un insieme di registri e una pila. Condivide con gli altri thread che appartengono allo stesso processo la sezione del codice, i fati, altre risorse. Un processo tradizionale, chiamato anche processo pesante, è composto da un solo thread. Un processo multithread è composto da più thread. Molti programmi per i moderni pc sono predisposti per essere eseguiti da processi multithread. Ad esempio una pagina web può assegnare un thread per la richiesta dei dati, uno per la visualizzazione a video, ecc.
I thread hanno anche il ruolo primario nei sistemi che impiegano le RPC; si tratta di un sistema che permette la comunicazione tra processi, fornendo un meccanismo di comunicazione simile alle normali chiamate di sistema. Molti kernel di sistemi operativi sono multithread.

Vantaggi

I vantaggi della programmazione multithread si possono classificare secondo 4 fattori principali:

- Tempo di risposta: rendere multithread un'applicazione interattiva può permettere a un programma di continuare la sua esecuzione, anche se una parte di esso è bloccata o sta eseguendo una operazione particolarmente lunga.
- Condivisione delle risorse: normalmente i thread condividono la memoria e le risorse del processo cui appartengono. Il vantaggio della condivisione del codice consiste nel fatto che un'applicazione può avere molti thread di attività diverse, tutti nello stesso spazio degli indirizzi
- Economia: assegnare memoria e risorse per la creazione di nuovi processi è costoso: poiché i thread condividono le risorse del processo cuoi appartengono, è molto più vantaggioso creare thread e gestirne i cambi di contesto
- Uso di sistemi multiprocessore: i vantaggi della programmazione multithread aumentano notevolmente

SEMAFORI CONTATORI CON IMPLEMENTAZIONE BINARIA

Un semaforo contatore può essere realizzato mediante l'uso di due semafori binari.
Le varie soluzioni hardware al problema della sezione critica basate su istruzioni quali TestAndSet e Swap complicano l'attività del programmatore. Per ovviare questo problema si fa uso dei semafori.
Un semaforo S è una variabile intera che si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche: wait e signal

```
wait(S){
        while(S<=0)
        ;//non-op
        S--;
        }
        signal(S){
        S++;
        }
```

Tutte le modifiche del semaforo sono contenute nelle operazioni wait e signal.

USO DEI SEMAFORI

Si usa distinguere tra semafori contatore il cui valore è illimitato e semafori binari il cui valore è 0 o 1. I semafori sono utilizzati per risolvere il problema della sezione critica con n processi che condividono un semaforo chiamato mutex inizializzato a 1.

```
do{
```

```
        wait(mutex);
        sezione critica
        signal(mutex);
        sezione non critica
} while(1);
```

I semafori contatore vengono utilizzati nel controllo dell'accesso a una data risorsa. I processi che vogliono utilizzare il semaforo usano wait decrementando il valore. Quando restituiscono la risorsa effettuano la signal incrementando il valore.

REALIZZAZIONE

Il principale svantaggio dell'uso dei semafori è che richiede una condizione di attesa attiva. Mentre un processo si trova nella propria sezione critica, qualsiasi altro processo che tendi di entrarvi si trova sempre nel ciclo del codice della sezione di ingresso. Questo costituisce per un sistema a multiprogrammazione poiché l'attesa attiva spreca cicli alla CPU. Per superare l'attesa attiva si possono modificare signal e wait . quando un processo deve attendere anziché entrare nella attesa attiva si blocca se stesso e quindi la CPU sceglie un altro processo. Un processo bloccato che attende a un semaforo si riavvia attraverso l'operazione wakeup.
La lista rappresenta i processi in attesa a un semaforo. L'operazione signal preleva un processo da tale lista e lo attiva. L'operazione block sospende il processo e l'operazione wakeup pone in stato di pronto per l'esecuzione il processo bloccato. Se il valore del semaforo è negativo, la sua dimensione è data dal numero dei processi che attendono a quel semaforo. I semafori devono essere eseguiti in modo atomico. Nei sistemi multiprocessore è necessario disabilitare le interruzioni di tutti i processori perché altrimenti le istruzioni dei diversi processi in esecuzione su processori distinti potrebbero interferire fra loro. Le operazioni wait e signal non eliminano completamente l'attesa attiva ma si limitano a rimuoverla dalla sezione di ingresso.

```
typedef struct{
        int valore;
        struct processo *L;
} semaforo

void wait(Semaforo S){
        S.valore--;
        if(S.valore<0){
        aggiungi questo processo a S.L;
        block(); }}

void signal(semaforo S){
        S.valore++;
        if(s.Valore<=0){
        togli un processo P da S.L;
        wakeup(P);
}}
```

Stallo e attesa indefinita

La realizzazione di un semaforo con coda d'attesa può condurre a situazioni in cui ciascun processo di un insieme di processi attende indefinitamente un evento che può essere causato solo da uno dei processo dello stesso insieme. Quando si verifica una situazione di questo tipo ci troviamo in una situazione di stallo. Un insieme di processi è in stallo se ciascun processo dell'insieme attende un evento che può essere causato solida un altro processo dell'insieme. Un'altra questione connessa alle situazioni d istallo è quella dell'attesa indefinita.

TABELLE DELLE PAGINE

PAGINAZIONE GERARCHICA

La maggior parte dei moderni calcolatori dispone di uno spazio d'indirizzi logici molto grande(da 2^32 a 2^64 elementi). Chiaramente sarebbe meglio evitare di collocare la tabella delle pagine in modo contiguo in memoria centrale. Una semplice soluzione a questo problema consiste nel suddividere la tabella delle pagine in parti più piccole; questo risultato si può ottenere in molti modi. Un metodo consiste nell'adottare un algoritmo di paginazione a due livelli, in cui la tabella stessa è paginata.

TABELLA DELLE PAGINE DI TIPO HASH

Un metodo di gestione molto comune degli spazi d'indirizzi relativi ad architetture oltre 32 bit consiste nell'impiego di una tabella delle pagine di tipo hash, in cui l'argomento della funzione hash è il numero della pagina virtuale.
Ciascun elemento è composto da 3 campi: il numero della pagina virtuale, l'indirizzo del frame corrispondente alla pagina virtuale, un puntatore al successivo elemento. Si applica la funzione hash al numero della pagina virtuale contenuto nell'indirizzo virtuale, identificando un elemento della tabella. Si confronta il numero di pagina con il primo campo degli elementi della lista e se coincidono si usa l'indirizzo relativo ala frame per generare l'indirizzo fisico. Ho soluzione efficace per indirizzi sparsi

TABELLA DELLE PAGINE INVERTITA

Generalmente si associa una tabella delle pagine a ogni processo e tale tabella contiene un elemento per ogni pagina virtuale che il processo sta utilizzando , oppure un elemento per ogni indirizzo virtuale a prescindere dalla validità di quest'ultimo. Poiché la tabella è ordinata per indirizzi virtuali il sistema operativo può calcolare in che punto della tabella si trova l'elemento dell'indirizzo fisico associato e usare direttamente tale valore. Uno degli inconvenienti insiti in questo metodo è costituito dalla dimensione di ciascuna tabella delle pagine, che può contenere milioni di elementi e occupare grandi quantità di memoria fisica,, necessaria proprio per sapere com'è impiegata la rimanente memoria fisica. Per risolvere questo problema si può fare uso della tabella delle pagine invertita. Una tabella delle Che pagine invertita ha un elemento per ogni pagina reale o frame. Ciascun elemento è quindi costituito dell'indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria con informazioni sul processo che possiede tale pagina. Ciascun indirizzo virtuale è una tripla del tipo seguente: <id-progesso,numero pagina, scostamento>. Viene ricercato tramite il pid il valore nella tabella della pagine se viene trovato lo scostamento più il valore il numero i.
Sebbene riduca la quantità di memoria necessaria per memorizzar ogni tabella delle pagine, aumenta pero il tempo di ricerca nella tabella. Poiché la tabella delle pagine invertita è ordinata per indirizzi fisici, mentre la ricerca si effettua per indirizzi virtuali, si deve effettuare la ricerca sull'intera tabella. Per limitare il problema si può usare una tabella hash che riduce la cerca a un solo o a pochi elementi della tabella delle pagine. Nei sistemi che adottano le tabelle delle pagine invertite l'implementazione è difficoltosa.

TEST AND SET

In generale si può affermare che qualunque soluzione al problema richiede l'uso di un semplice strumento detto lock. Il corretto ordine degli accessi alle strutture dati del kernel è garantito dal fatto che lezioni critiche sono protette da lock. In altri termini un processo per accedere alla propria sezione critica deve ottenere il permesso di un lock.

In un sistema dotato di una singola CPU tale problema si potrebbe risolvere semplicemente se si potessero interidre le interruzioni. Non si potrebbe eseguire nessunaltra istruzione quindi non si potrebbe apportare alcuna modifica inaspettata alle variabili condivise. È questo l'approccio seguito dal kernel senza diritto di prelazione.

Le interruzioni portano uno spreco del tempo e per questo motivo molte architetture offrono particolari istruzioni che permettono di controllare e modificare il contenuto di una parola di memoria oppure di scambiare il contenuto di due parole di memoria.

```
boolean TestAndSet(boolean &obiettivo){
        boolean valore = obiettivo;
        obiettivo=true;
        return valore;
}
```

L'istruzione testAndSet è eseguita atomicamente, cioè come un'unità non soggetta ad interruzioni

```
do {
        while(TestAndSet(blocco));
        sezione critica
        blocco=false;
        sezione non critica
} while (1);
```

ALLOCAZIONE DEI FRAME

In un sistema multiprogrammato, come distribuiamo i frame (pagine di memoria) disponibili fra i processi? Ci sono varie soluzioni detti algoritmi di allocazione:

Allocazione uniforme: lo stesso numero di frame a tutti i processi: n frame, p processi, n/p frame a ciascun processo

Allocazione proporzionale: tiene conto del fatto che i processi hanno dimensioni diverse. Se le dimensioni in pagine di tre processi sono: P1 = 4, P2 = 6, P3 = 12 e ci sono 11 frame disponibili, l'allocazione sarà: P1 = 2, P2 = 3, P3 = 6.

Allocazione proporzionale in base alla priorità: tiene conto del fatto che i processi hanno priorità diverse Una volta deciso come allocare le pagine , dobbiamo decide in quale gruppo di pagine (di che processo) possiamo/dobbiamo scegliere la vittima da rimuovere dalla MP. Anche qui ci sono diverse soluzioni.

Allocazione globale: scegliamo la vittima fra tutte le pagine della memoria principale (di solito, escluse quelle del SO). Potremmo in questo modo portare via una pagina ad un processo diverso da quello che ha generato il page fault.

Allocazione locale: scegliamo la vittima fra le pagine del processo che ha generato page fault. In questo modo il numero di frame per processo rimane costante. Attenzione però, se si danno troppe (relativamente) pagine ad un processo, si può peggiorare la situazione del sistema perché gli altri processi genereranno più page fault.

PAGINAZIONE

La paginazione è un metodo di gestione della memoria che permette che lo sazio degli indirizzi fisici di un processo non sia contiguo. Elimina il gravoso problema della sistemazione di blocchi di memoria di diverse dimensioni in memoria ausiliaria. Il problema insorge perché quando alcuni frammenti di codice o dati residente in memoria centrale devono essere scaricati, si deve trovare lo spazio necessario in memoria ausiliaria. I problemi di frammentazione relativi alla memoria centrale valgono anche per la memoria secondaria con la differenza che in questo caso l'accesso è molto più lento e quindi è impossibile eseguire la compattazione.

Metodo di base

Il metodo di base per implementare la paginazione consiste nel suddividere la memoria fisica in blocchi di dimensione costante detti anche grame o pagine fisiche e nel suddividere la memoria logica in blocchi di pari dimensioni detti pagine. Quando si deve eseguire un processo si caricano le sue pagine nei grame disponibile prendendole dalla memoria ausiliaria divisa in blocchi di dimensione fissa, uguale a quella dei frame della memoria.

Ogni indirizzo generato dalla CPU è diviso in due parti: numero di pagina (p) e uno scostamento (d). il numero di pagina serve come indice per la tabella delle pagine contente l'indirizzo base in memoria fisica di ogni pagina. Questo indirizzo si combina con lo scostamento d che forma l'indirizzo fisico. La dimensione della pagina è definita dall'architettura e varia tra i 512 byte a 16 MB. La paginazione non è altro che una forma di rilocazione dinamica: ogni indirizzo logico l'architettura di paginazione fa corrispondere un indirizzo fisico. L'uso della tabella delle pagine è simile all'uso di una tabella di registri base uno per ciascun frame.

Con la paginazione si può evitare la frammentazione esterna: qualsiasi frame libero si può assegnare a un processo che ne abbia bisogno. Pero c'è il problema della frammentazione interna nel caso peggiore in cui si ha un processo che necessita di n pagine + 1 byte e si devono allocare n+1 pagine con una frammentazione interna media di mezza pagina per processo. Questo fa si che si convenga usare pagine di piccole dimensioni.

Quando si deve eseguire un processo si esamina la sua dimensione espressa in pagine. Se necessita di n pagine, devono essere disponibili almeno n frame.

Un aspetto importante della paginazione è la netta distinzione tra la memoria vista dall'utente e l'effettiva memoria fisica: il programma utente vede la memoria come un unico spazio contiguo, contenente anche altri programmi. La differenza tra la memoria vista dall'utente e la memoria fisica è colmata dall'architettura di traduzione degli indirizzi.

Poiché il SO gestisce la memoria fisica, deve essere informato dei relativi particolari di allocazione: quali grame sono assegnati, quali sono disponibili, il loro numero totale e cosi via. In genere queste informazioni sono contenute in una struttura dati chiamata tabella dei frame contenenti un elemento per ogni frame, indicante se sia libero oppure assegnato e se è assegnato a quale pagina di quale processo. Il SO conserva una copia della tabella delle pagine per ciascun processo, cosi come conserva una copia dei valori contenuti nel contatore di programma e nei registri. Questa coppia si usa per tradurr gli indirizzi logici in fisici ogni volta che il SO deve associare esplicitamente un indirizzo fisico a uno logico. La paginazione quindi fa aumentare il cambio di contesto.

STRUTTURE RAID

L'evoluzione tecnologica ha reso le unita a disco progressivamente più piccole e meno costose. Oggi è possibile equipaggiare sistemi con più dischi senza spendere somme esorbitanti. Attraverso la configurazione raid è possibile da un lato velocizzare l'accesso e da un altro effettua copie di dischi qualora uno potrebbe rompersi.
Miglioramenti dell'affidabilità tramite ridondanza
La possibilità che un disco si rompa si guasti è molto alta molto più alta della possibilità che uno specifico disco isolato presenti un guasto. La soluzione dell'affidabilità sta nell'introdurre una ridondanza e quindi in caso di guasto basta sostituire il disco. Il metodo più semplice è quello del mirroring ogni disco logico consiste di due fisici e ogni scrittura si effettua in entrambi id dischi. Ciò è sufficiente a risolvere i problemi dipesi da un guasto relativo alla rottura di un disco no pero nel caso di un disastro naturale.
Nel caso in cui si hanno interruzioni di corrente e una scrittura no n andasse a buon fine, si deve avere il modo per non far corrompere il file system. una soluzione prevede la scrittura in uno solo dei dischi e solo successivamente nell'altro in modo da avere nella copia sempre scritture portate a termine.
Miglioramento delle prestazioni tramite il parallelismo
L'access in parallelo di più dischi può portare grossi vantaggi. Con la copiatura speculare dei dischi la frequenza con la quale si possono gestire le richieste di lettura raddoppia poiché ciascuna richiesta si può inviare indifferentemente a uno dei due dischi. Per migliorare la capacita di trasferimento i dati vengono distribuiti in sezione su più dischi. Questa forma nota come sezionamento dei dati, consiste nel distribuire i bit di ciascun byte su pi dischi facendo si che la quantità di dati letti o scritti sai 8 volte superiore. Questo pero porta ad un grave svantaggio, nel caso in cui uno dei dischi si rompa, tutti i dati sono irrecuperabile.

ALGORITMO DEL FORNAIO

```
do{
        scelta[i] = true;
        numero[i]=max(numero[0], numero[1], ...numero[n-1])+1;
        scelta[i] = false;
        for(j=o;j<n;j++){
        while(scelta[j]);
        while((numero[j] != 0)&&(numero[j,j]<numero[i,i]));
}
        sezione critica
        numero[1] =0;
        sezione non critica
} while (1);
```

L'algoritmo 3 risolve il problema della sezione critica per due processi, mentre l'algoritmo del fornaio lo risolve per n processi. È basato su uno schema di servizio usato nella panetteria dove si deve evitare la confusione dei turni. Al suo ingresso nel negozio ogni cliente riceve un numero. Si serve progressivamente il cliente con il più basso. A parità di numero si serve il cliente con il nome minore.
Boolean scelta [n]
Int numero
(a,b)<(c,d) se a<c oppure se a=c e b<d

SOLUZIONE DI PETERSON

La soluzione di Peterson si applica a 2 processi, ognuno dei quali esegue alternativamente la propria sezione critica e la sezione rimanente. Vengono introdotte la variabile:

int turno (segnala di chi è il turno)

boolean pronto[2] (indica se un processo sia pronto ad entrare nella propria sezione critica)

```
do{
        pronto[i] = true;
        turno = j
        while(pronto[j] && turno == j);
        Sezione non critica
        pronto[i]=false;
        sezione non critica
} while (1);
```

Per dimostrare che la mutua esclusione è preservata si osserva che Pi acceda alla propria sezione critica solo se pronto [j]=false oppure turno=i.

Poiché tuttavia Pi non modifica il valore della variabile turno durante l'esecuzione dell'istruzione while, Pi entrerà nella sezione critica (pregresso) dopo che Pj abbia effettuato non più di un ingresso (attesa limitata).

INTERROGAZIONE CICLICA

Il protocollo completo per l'interazione fra la CPU e un controllore può essere intricato, ma la fondamentale nozione di negoziazione (handshaking) è semplice, ed è illustra con un esempio.

Il controllore specifica il suo stato per mezzo del bit busy del registro status; pone a 1 il bit busy quando è impegnato in un'operazione, e lo pone a 0 quando è pronto a eseguire il comando successo. La CPU comunica le sue richieste tramite il bit comand-ready nel registro command; pone questo bit a 1 quando il controllore deve eseguire un comando.

1. La CPU legge ripetutamente il bit busy finché non valga 0.
2. La CPU pone a 1 il bit write del registro dei comandi e scrive un byte nel registro data-out
3. La CPU pone a 1 il bit command-ready .
4. Quando il controllore si accorge che il bit command-ready è posto a 1, pone a 1 il bit busy
5. Il controllore legge il registro dei comandi e trova il comando write; legge il registro data-out per ottenere il byte da scrivere, e compie l'operazione di scrittura nel dispositivo
6. Il controllore pone a 0 il bit command-ready, pone a 0 il bit error nel registro status per indicare che l'operazione di I/O ha avuto esito positivo, e pone a 0 il bit busy per indicare che l'operazione è terminata. La sequenza appena descritta si ripete per ogni byte.

Durante l'esecuzione del passo 1, la CPU è in attesa attiva (busy-waiting) o in interrogazione ciclica (polling): itera la lettura del registro status finché il bit busy assume il valore 0. Se il controllore e il dispositivo sono veloci, questo metodo è ragionevole, ma se l'attesa rischia di prolungarsi, sarebbe probabilmente meglio se la CPU si dedicasse a un'altra operazione.

Quando, ad esempio, i dati affluiscono in una porta seriale o della tastiera. Il piccolo buffer del controllore diverrà presto pieno, e se la CPU attende troppo a lungo prima di riprendere la lettura dei byte, si prenderanno informazioni.

In molte architetture di calcolatori sono sufficienti tre cicli di istruzioni di CPU per interrogare ciclicamente un dispositivo: read, lettura di un registro del dispositivo; logical-and, configurazione logica usata per estrarre il valore di un bit di stato, e branch, salto a un altro punto del codice se l'argomento è diverso da zero.

Anziché richiede alla CPU di eseguire un'interrogazione ciclica, può essere più efficiente far si che il controllore comunichi alla CPU che il dispositivo è pronto. Il meccanismo dall'architettura che permette tale comunicazione si chiama interruzione della CPU o interrupt.