SISTEMI

Scheduling

Prof. Viglietti Francesco

Classe 4 B info

A.S. 2010-11

Scheduling 1

Un OS può essere considerato come un gestore di risorse: processore, memoria principale, memoria di storage, dispositivi, periferiche, ecc.
Per svolgere i suoi compiti, un OS ha bisogno di strutture dati per mantenere informazioni sulle risorse gestite. Queste strutture dati comprendono:

- tabelle di memoria
- allocazione memoria per OS
- allocazione memoria principale e secondaria per i processi
- informazioni per i meccanismi di protezione
- tabelle di I/O
- informazioni sullo stato di assegnazione dei dispositivi utilizzati dal computer
- gestione di code di richieste
- tabelle del file system
- elenco dei dispositivi utilizzati per mantenere il file system
- elenco dei file aperti e loro stato
- tabelle dei processi

Ovviamente non sarà facile studiare tutte queste strutture. Ora ci occuperemo di una sola di queste, la tabella dei processi e del processo del OS che gestisce l'alternanza fra i vari processi in esecuzione, **lo scheduler**.

Scheduling 2

Un processo si può descrivere tramite:
1. il codice da eseguire (segmento codice)
2. i dati su cui operare (segmenti dati)
3. uno stack di lavoro per la gestione di chiamate di funzione, passaggio di parametri e variabili locali
4. un insieme di attributi contenenti tutte le informazioni necessarie per la gestione del task stesso, incluse le informazioni necessarie per i punti 1-3

Questo insieme di attributi prende il nome di descrittore del task (process control block, PCB). Per gestire al meglio un insieme di task si organizzano tramite una tabella contenente i loro descrittori, ognuno associato ad un task. Le informazioni contenute in un descrittore sono di tre tipi:
1. ***Informazioni di identificazione*:** num. progressivo (pid), pid del padre o di altri task collegati ad esso, id utente che ha richiesto l'esecuzione.
2. ***Informazioni di stato del task***: reg. gen. della CPU, reg. speciali, reg. di stato.
3. ***Informazioni di controllo del processo***: stato del task; informazioni su algoritmo di scheduling; id evento per cui il task è in attesa; valore registri base e limite dei segmenti utilizzati, informazioni di accounting: t. di esecuzione, t. trascorso dall'attivazione; informazioni sulle risorse; informazioni per interprocess communication (IPC): segnali,semafori...

Scheduling 3

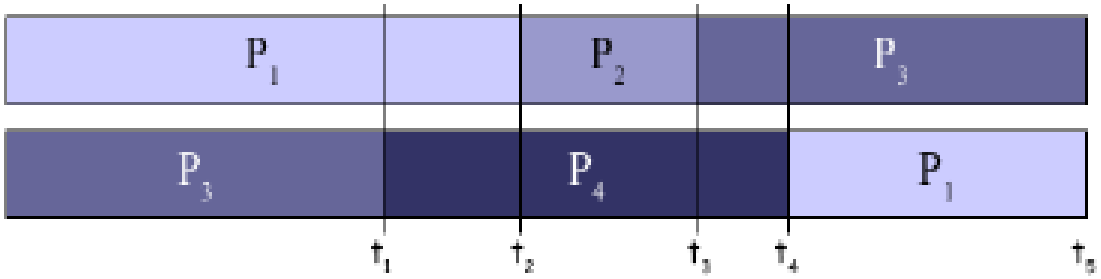
Lo scheduler è la componente più importante del kernel di un OS. Esso si occupa di gestire l'avvicendamento dei processi, deciderne l'esecuzione ad ogni istante, intervenire quando viene richiesta un'operazione di I/O e quando un'operazione di I/O termina, ma anche periodicamente.
Tutte le volte che avviene un interrupt il processore è soggetto ad un mode switching, da user mode a kernel mode. Durante la gestione dell'interrupt vengono intraprese le opportune azioni per gestire l'evento: viene chiamato lo scheduler e se questo decide di eseguire un altro processo, il sistema è soggetto ad un context switching. Lo stato del processo attuale viene salvato nel PCB corrispondente e lo stato del processo selezionato per l'esecuzione viene caricato dal PCB nel processore.

Processi e Thread

Un thread è l'unità base di utilizzazione della CPU. Ogni thread possiede la propria copia dello stato del processore, il proprio program counter e uno stack separato. I thread appartenenti allo stesso processo condividono: codice, dati, risorse di I/O.
La condivisione di informazioni tra thread logicamente correlati rende più semplice l'implementazione di certe applicazioni.
Un OS può implementare user thread o kernel thread. Gli user thread vengono supportati sopra il kernel e vengono implementati da una "thread library" a livello utente. La thread library fornisce supporto per la creazione, lo scheduling e la gestione dei thread senza alcun intervento del kernel.
Ha un'implementazione risultante molto efficiente, ma se il kernel è single-threaded, qualsiasi user thread che effettua una chiamata di sistema bloccante (che si pone in attesa di I/O) causa il blocco dell'intero processo.
I kernel thread vengono supportati direttamente da OS: la creazione, lo scheduling e la gestione dei thread sono implementati a livello kernel. Poiché è il kernel a gestire lo scheduling dei thread, se un thread esegue una operazione di I/O, il kernel può selezionare un altro thread in attesa di essere eseguito, ma l'implementazione risultante è più lenta, perché richiede un passaggio da user mode a kernel mode.

rappresentazione

Per rappresentare uno scheduling si usano i diagrammi di Gantt,che scandisce la linea del tempo tramite la sequenza degli impegni da affrontare.
Nel caso si debba rappresentare lo schedule di più risorse il Gantt risulta composto da più righe parallele.



Eventi che possono causare un context switch:

1. se un task passa da running a waiting (operazione di I/O)
2. quando un processo passa da running a ready (causa di un interrupt)
3. quando un processo passa da waiting a ready
4. quando un processo termina

In 1 e 4, l'unica scelta è quella di selezionare un altro processo per l'esecuzione.
In 2 e 3, è possibile continuare ad eseguire il processo corrente.
Uno scheduler si dice non preemptive se i context switch avvengono solo nelle condizioni 1 e 4. In altre parole: il controllo della risorsa viene trasferito solo se l'assegnatario attuale lo cede volontariamente. ES: Win 3.1, Mac OS <8
Uno scheduler si dice preemptive se i context switch possono avvenire in ogni condizione. In altre parole: è possibile che il controllo della risorsa venga tolto all'assegnatario attuale a causa di un evento. ES: tutti gli scheduler moderni

scelta

E' chiaro dunque che il comportamento dello scheduler abbia un forte impatto sulle prestazioni di un OS nei confronti delle applicazioni che vi vengono eseguite sopra. Sarà dunque molto importante stabilire quali sono i criteri per capire quale scheduler si comporta meglio e ha quindi le maggiori prestazioni.
Vediamo questi criteri:
1. Utilizzo della risorsa (CPU): % di tempo in cui la CPU è occupata ad eseguire processi deve essere massimizzata.
2. Throughput: Rappresenta il numero di processi completati per unità di tempo, dipende dalla lunghezza dei processi e deve essere massimizzato.
3. Tempo di turnaround: Il tempo che intercorre dalla sottomissione di un processo alla sua terminazione deve essere minimizzato.
4. Tempo di attesa: Il tempo trascorso da un processo nella coda ready deve essere minimizzato.
5. Tempo di risposta: Il tempo che intercorre fra la sottomissione di un processo e il tempo di prima risposta, particolarmente significativo nei programmi interattivi, deve essere minimizzato.

Algoritmi di scheduling 1

First Come, First Served (FCFS)

Il processo che arriva per primo, viene servito per primo. Adotta una politica senza preemption. L'implementazione è semplice, tramite una coda (FIFO). I problemi sono degli elevati tempi medi di attesa e di turnaround e il fatto che i processi che utilizzano molto la CPU ritardano pesantemente quelli che hanno elevate necessità di processi I/O.

Shortest Job First (SJF)

Questo algoritmo assegna la CPU al processo in stato ready che ha la minima durata. In realtà esistono due versioni di questo algoritmo

1. Algoritmo Shortest Next CPU Burst First: La CPU viene assegnata al processo ready che ha la minima durata del CPU burst successivo. Adotta quindi una politica senza preemption.
2. Shortest-Remaining-Time First: E' la versione cooperativa (con preemption) dell'algoritmo SJF. Il processo corrente può essere messo nella coda ready, se arriva un processo con un CPU burst più breve di quanto rimane da eseguire al processo corrente.

L'algoritmo SJF è ottimale rispetto al tempo di attesa, in quanto produce il minor tempo di attesa possibile, ma è impossibile da implementare in pratica! Come si fa a sapere qual è il processo che durerà di meno?? (solo statistica).

Algoritmi di scheduling 2 RR

Scheduling Round-Robin

E' basato sul concetto di time slice: un processo non può rimanere in esecuzione per un tempo superiore alla durata del quanto di tempo. L'insieme dei processi pronti è organizzato come una coda. Ci sono due possibilità per un task:

1. può lasciare la CPU, in seguito ad un'operazione di I/O
2. può esaurire il suo quanto di tempo senza completare il suo CPU burst, nel qual caso viene aggiunto in fondo alla coda dei processi pronti.

In entrambi i casi, il prossimo processo da eseguire è il primo della coda dei processi pronti.

La durata del quanto di tempo è un parametro critico del sistema.

Per implementare correttamente lo scheduling RR è necessario che l'hardware fornisca un timer (interval) che agisca come "sveglia" del processore. Il timer è un dispositivo che, è in grado di fornire un interrupt allo scadere del tempo prefissato. RR fornisce le stesse possibilità di esecuzione a tutti i processi.

Usando RR puro la visualizzazione di un video potrebbe essere ritardata da un processo che sta smistando la posta … la mail può aspettare un secondo, il frame video NO!!

Algoritmi di scheduling 3

C'è bisogno di uno scheduling a priorità, ogni processo è associato ad una specifica priorità: lo scheduler sceglie il processo pronto con priorità più alta.
Ovviamente però se la priorità di un processo è statica (non cambia nel tempo) uno scheduler che esegue spesso processi ad alta priorità non avrà mai tempo per quelli con priorità medio-bassa, mettendo di fatto questi processi in starvation. Si utilizza dunque un sistema di priorità dinamica basata su aging.
L'aging è una tecnica che consiste nell'incrementare gradualmente la priorità dei processi in attesa. Posto che il range di variazione delle priorità sia limitato, nessun processo rimarrà in attesa per un tempo indefinito perché prima o poi raggiungerà la priorità massima.

Scheduling a classi di priorità:

E' possibile creare diverse classi di processi con caratteristiche simili e assegnare ad ogni classe specifiche priorità: la coda ready viene quindi scomposta in molteplici "sottocode", una per ogni classe di processi.
L'algoritmo di scheduling in questo caso selezionerà il processo da eseguire fra quelli pronti della classe a priorità massima che contiene processi.

Esempio

Servirà per valutare tutti gli algoritmi.
Ordine di arrivo dei processi nella coda ready: P1 a t=0, P2 a t=1, P3 a t=2.
Durata CPU burst: P1=10ms, P2=6ms, P3=3ms; Time slice per RR 3ms

Algoritmi → Tempo medio↓	***FCFS***	***SNCBF***	***SRTF***	***RR***
Turnaround	14 ms	13 ms	10,3 ms	13,3 ms
Attesa	7,66 ms	6,66 ms	4 ms	7 ms
Risposta	7,66 ms	6,66 ms	0 ms	2 ms

Il turnaround lo calcolo come la media della durata dei singoli processi
FCFS→[(10)+(9+6)+(8+6+3)]/3=14ms; SNCBF→[10+(9+3+6)+(8+3)]/3=13ms
SRTF→[(1+1+3+5+9)+(1+3+5)+(3)]/3=10,3ms;
RR→[(3+6+3+3+4)+(2+6+3)+(4+3)]=13,3ms.
Il tempo di attesa lo trovo come media dell'attesa dei processi fino all'esecuzione.
FCFS→[(0)+(9)+(8+6)]/3=7,66ms; SNCBF→[(0)+(8)+(9+3)]/3=6,67ms
SRTF→[(0+1+3+5)+(0+3)+(0)]/3=4ms; RR→[(0+6+3)+(2+6)+(4)]=7ms
Il tempo di risposta lo trovo come media del tempo necessario all'esecuzione del processo. Nei primi due algoritmi corrisponde al tempo di attesa.
SRTF→[(0)+(0)+(0)]/3=0ms; RR→[(0)+(2)+(4)]/3=2ms