**MINIX3**

Immagine che contiene testo, diagramma, schermata, Piano

Descrizione generata automaticamente

Lo scheduler di MINIX 3 utilizza un sistema di code multilivello. Sono definite sedici code, anche se è semplice ricompilare il sistema per utilizzare un numero diverso di code. La coda di priorità più bassa è utilizzata solo dal processo IDLE, che viene eseguito quando non c'è nulla altro da fare. Di default, i processi utente iniziano in una coda di priorità più alta rispetto a quella più bassa.

Viene usato il round robin in ogni coda. Se un processo in esecuzione esaurisce il suo quantum, viene spostato in fondo alla sua coda e gli viene assegnato un nuovo quantum. Tuttavia, quando un processo bloccato viene risvegliato, viene messo in testa alla sua coda se aveva parte del suo quantum rimasto quando si è bloccato. Tuttavia, non gli viene dato un nuovo quantum completo; riceve solo ciò che gli era rimasto quando si è bloccato. L'esistenza dell'array `rdytail` rende efficiente l'aggiunta di un processo alla fine di una coda. Ogni volta che un processo in esecuzione diventa bloccato o un processo eseguibile viene terminato da un segnale, quel processo viene rimosso dalle code del pianificatore. Solo i processi eseguibili vengono messi in coda.

Data la struttura delle code appena descritte, l'algoritmo di scheduling è semplice: trova la coda di priorità più alta che non è vuota e scegli il processo in testa a quella coda. Il processo IDLE è sempre pronto ed è nella coda di priorità più bassa. Se tutte le code di priorità più alte sono vuote, viene eseguito il processo IDLE..

• Priorità allineate con i valori “nice” in UNIX

– Un numero più basso indica una priorità più alta, con 0 che rappresenta la priorità più alta.

• I processi in Idle vengono eseguiti al livello di priorità più basso(livello 15)

* I task del sistema e del u-kernel clock vengono eseguiti alla priorità più alta (livello 0).
* I driver e i processi di sistema vengono eseguiti a livelli di priorità 1-4.
* I processi utente vengono eseguiti a livelli di priorità da 7 a 14.
* La priorità dei task del kernel, dei driver e dei processi di sistema è fissata.
* La priorità dei processi utente varia a seconda del comportamento

– +1 se un quantum intero viene usato

– -1 se si blocca prima di usare un quantum

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

Il "Restart" è il punto comune raggiunto dopo l'avvio del sistema, gli interrupt o le chiamate di sistema. Il processo più meritevole (che può essere e spesso è un processo diverso da quello interrotto precedentemente) viene eseguito successivamente. Non è mostrato quanto accade per gli interrupt che si verificano mentre il kernel stesso è in esecuzione.

**Minix Scheduler Implementation**

• L’implementazione è sparsa attraverso diversi file

• main.c - chiamata a restart

– rdy\_heqd, rdy\_tail: arrays con inizio/fine della coda di ogni livello

• table.c - coda iniziale dei processi durante l’avvio del sistema

– enqueue, dequeue: usato per aggiungere/rimuovere enry alla coda

• Function sched: determina quale processo dovrebbe stare in quale coda

* Verifica il quantum usato dal processo e aggiusta la priorità +1/-1

• Function pick\_proc: testa la “reach” queue e trova la prima coda non vuota

• proc.c

– lock\_send, lock\_enqueu, lock\_dequeue, lock\_notify

• usato per blocco e sblocco base delle code

• Clock task monitors all processes

– Quantum expires: mette il processo alla fine della coda

– i driver e i server ricevono quanti di tempo (quanta) più lunghi rispetto ad altri processi, il che significa che hanno più tempo di CPU a disposizione prima di essere prelazionati. Tuttavia, non sono esenti dalla prelazione.

**OS161**

OS161 ha un sistema di scheduling che viene utilizzato per decidere quale processo deve essere eseguito sulla CPU in un determinato momento. Il suo scheduler è basato su un algoritmo di tipo "round robin", che è un tipo di scheduler a tempo condiviso. Ecco alcune delle caratteristiche principali dello scheduling in OS161:

* **Round Robin:** L'algoritmo di scheduling di OS161 utilizza un approccio round robin. In questo schema, i processi vengono eseguiti per un periodo di tempo fisso chiamato "quantum", e poi vengono prelazionati per dare opportunità a altri processi di eseguire. Questo garantisce un trattamento equo ai processi e impedisce che un processo monopolizzi la CPU.
* **Priorità fissa:** In OS161, tutti i processi hanno la stessa priorità. Ciò significa che il sistema operativo assegna priorità uguali a tutti i processi e il scheduling si basa principalmente sul principio del round robin.
* **Gestione dei processi in attesa:** OS161 utilizza un approccio basato su code di attesa (queues) per gestire i processi in attesa di esecuzione. Ci sono diverse code di attesa a seconda dello stato del processo, come la coda dei processi pronti, la coda dei processi in attesa e la coda dei processi in esecuzione.
* **Interruzioni di timer:** OS161 utilizza timer per prelazionare i processi dopo che hanno esaurito il loro quantum. Quando scade il timer, il sistema operativo effettua una prelazione e seleziona il prossimo processo da eseguire.
* **Gestione delle priorità:** Anche se tutti i processi hanno la stessa priorità di base, OS161 può gestire le priorità dei processi in modo dinamico. Ad esempio, il sistema operativo può impostare priorità più alte per i processi interattivi per migliorare la responsività del sistema.
* **Supporto per l'allocazione di CPU:** OS161 offre supporto per l'allocazione della CPU su architetture multiprocessore, il che significa che può gestire in modo efficiente l'esecuzione dei processi su più CPU.