Università degli studi di Bologna

CORSO DI LAUREA IN INFORMATICA

Sviluppo del sistema operativo Panda+

Fase 2

Relatori | Marco Coppola

Valerio Pio De Nicola

ANNO ACCADEMICO 2022/2023

Inizializzazione

Moduli Fase 1

Vengono chiamate le routine per inizializzare le strutture dei PCB (Process Control Block), dei semafori e dei namespace.

Pass Up Vector

L'area di memoria del Pass Up Vector viene popolata con i puntatori alle routine da eseguire in caso di eccezione. In particolare, gestiamo l'eccezione di TLB-refill (fornita dal test) e la routine per gestire tutte le altre, il cui comportamento varia in base al tipo di eccezione sollevata.

Scheduler

Inizializziamo le strutture necessarie per il funzionamento dello scheduler. Queste includono il conteggio dei processi, il numero di processi bloccati e la coda dei processi pronti da eseguire. Inoltre, configuriamo il timer in modo da generare un'eccezione ogni 100 ms, che verrà utilizzata per risvegliare i processi che chiamano la SYS7 WaitForClock. Infine, inizializziamo il processo root con tutti gli interrupt abilitati e in modalità kernel. Il processo root viene quindi aggiunto alla coda dei processi pronti ed eseguirà la funzione di test definita nel file p2test.c fornito.

Scheduler

l funzionamento dello scheduler dipende principalmente dalla presenza di processi nella coda dei processi pronti, e ciò comporta due comportamenti molto diversi.

- Se la coda dei processi pronti è vuota, ci sono tre casi possibili:
  - Se il conteggio dei processi è uguale a 0, significa che tutti i processi sono stati terminati correttamente e non c'è più nulla da eseguire, quindi il sistema va in **HALT**.
  - Se ci sono ancora processi bloccati, dobbiamo attendere che almeno uno di essi venga sbloccato, quindi il sistema va in **WAIT**.
  - Se ci sono ancora processi, ma nessuno è in stato di soft_block, allora siamo in una situazione di deadlock, e il sistema va in **PANIC**.

Se ci sono processi pronti, estraiamo il primo, resettiamo il PLT e carichiamo nella CPU il prossimo processo da eseguire.

System Calls

Le system call vengono richiamate dai processi per eseguire attività a livello kernel, interagendo con esso attraverso registri predefiniti. Per semplificare l'accesso a questi registri, abbiamo creato delle macro come **REG_A0, REG_A1, REG_A2** e **REG_A3.**

Nelle descrizioni delle system call, elenchiamo solo le scelte implementative degne di nota:

- Nella gestione della system call SYS2 (Terminate Process), abbiamo adottato un approccio ricorsivo per garantire la terminazione completa della progenie di un processo. Uno dei compiti più complessi che abbiamo affrontato riguarda il rilascio delle risorse associate ai semafori utilizzati dalla progenie, risorse che non sarebbero state rilasciate automaticamente. In questa situazione, abbiamo implementato un controllo per ogni processo appartenente alla progenie per assicurarci che nessuno di essi sia bloccato su un semaforo. Nel caso in cui scopriamo che ci sono processi bloccati, verifichiamo attentamente che il processo da terminare sia l'unico bloccato su quel particolare semaforo. Questa precauzione è fondamentale per evitare il rilascio involontario di processi. Se le condizioni sono soddisfatte, procediamo al rilascio del semaforo, impostando il suo valore a 1.

- Nella gestione della system call SYS5 (Do IO), ci siamo concentrati principalmente sulla gestione dei dispositivi terminali, affrontando quindi il problema di riconoscere, partendo dal comando richiesto, a quale semaforo associato al terminale ci stiamo riferendo. Per farlo abbiamo implementato un approccio basato sull'offset: tramite l'indirizzo del primo terminale, il numero totale di dispositivi terminali e la distanza tra le aree di memoria che li descrivono, otteniamo in modo semplice l'indice del terminale eseguendo il seguente calcolo:

$$TermIndex = \frac{Indirizzo_{comando} - TERM0ADDR}{DEVREGSIZE}$$

  Visto che abbiamo deciso di salvare i semafori dedicati ai terminali in un array, dopo aver ricavato l'indice del terminale possiamo accedere in modo semplice al suo semaforo tramite esso.

Interrupt

La gestione degli interrupt sollevati dai terminali è stata resa molto semplice e efficiente grazie alla gestione dei semafori per i terminali attraverso degli array in quanto possiamo accedere immediatamente all'indice del dispositivo che ha sollevato l'eccezione e ottenerne il semaforo associato.

Osservazioni finali

Rispetto alla Fase 1, nella Fase 2 abbiamo intensivamente utilizzato la suite di strumenti di debugging fornita da umps3. Questo è stato necessario poiché dovevamo accedere a specifiche aree di memoria in momenti critici del processo di sviluppo.
Per semplificarci il lavoro, abbiamo sfruttato l'uso dei "breakpoints" che ci hanno permesso di identificare le chiamate alle system call e di individuare il punto esatto nel flusso di esecuzione in cui si verificavano problematiche, consentendoci di risolverle con precisione. Inoltre, abbiamo introdotto funzioni ausiliarie nei file della Fase 1. Queste funzioni hanno servito sia a scopi di debugging che a semplificare le operazioni più complesse necessarie per lo sviluppo della Fase 2.