Scelte progettuali

Stefano De Santis, Cristiano Guidotti, Iacopo Porcedda, Jacopo Rimediotti

Alma Mater Studiorum – Università di Bologna

Scelte progettuali

La fase 1.5 del progetto BiKaya è stata sviluppata in linguaggio C, con l'obiettivo di realizzare il funzionamento di un scheduler dei processi funzionante per le architetture uARM e uMPS.
La sua realizzazione è stata strutturata in quattro moduli:

- Inizializzazione del sistema
- Gestione scheduling dei processi
- Gestione delle system call
- Gestione degli interrupt

Nonostante questa strutturazione i moduli interagiscono tra di loro in base alle necessità

Inizializzazione del sistema

La procedura di inizializzazione consiste di quattro fasi principali, svolte nel seguente ordine:

1. Inizializzazione delle New Areas
2. Inizializzazione della lista di PCB liberi
3. Inizializzazione dello scheduler (vedere paragrafo successivo)
4. Inizializzazione del programma di test

Tutte le fasi sono attivate all'interno della funzione `main()`, l'entry point del sistema operativo.
La preparazione delle New Areas riguarda il settaggio degli stati su tali aree per la corretta inizializzazione del processore in caso di sollevamento di un'eccezione, che comprende:

- Abilitazione modalità kernel
- Disattivazione Virtual memory
- Mascheramento Interrupt
- Settaggio PC su entry point degli handler dedicati
- Settaggio SP su RAMTOP (relativo ad entrambe le architetture)

Queste istruzioni sono rappresentate da uno stato del processore che verrà caricato in memoria quando si verificherà una delle seguenti situazioni: syscall/breakpoint, interrupt, operazione per TLB o trap relativa ad un errore. Questa fase si è occupata della strutturazione delle New Area dedicate a interrupt e systemcall: ad ulteriori eccezioni sono assegnati entry point di gestori senza implementazione significativa, poiché non obiettivo principale di questa fase del progetto.
L'inizializzazione della Free List consiste nella creazione della lista contenente 20 PCB settati con valori di default.

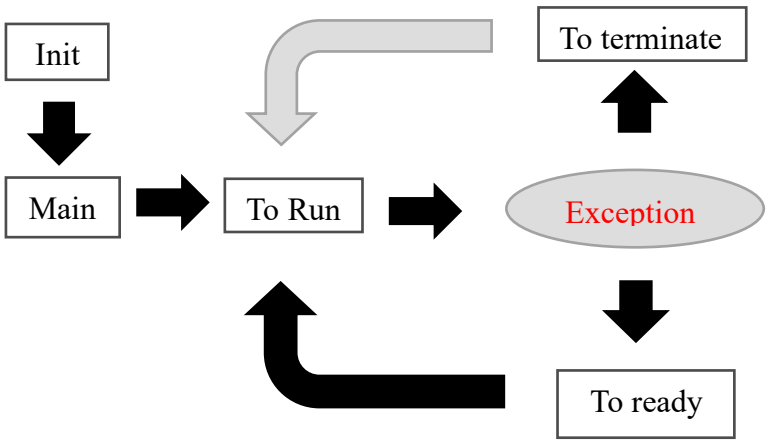
Nell'ultima fase, tre di questi PCB vengono poi estratti dalla Free List, e per ciascuno di essi sono inizializzate come da specifiche: Abilitazione degli interrupt, disattivazione Virtual memory, abilitazione modalità kernel, assegnazione di diverse priorità, SP per assegnazione di frame disponibili e PC che punta ad un entry point diverso per eseguire i programmi di test.
Dopo l'inserimento dei tre blocchi all'interno della Ready Queue, viene avviato lo scheduler.

Gestione scheduling dei processi

Lo scheduling dei processi avviene in sinergia con la gestione delle eccezioni.
Lo scheduler infatti fornisce delle interfacce agli altri gestori per poter richiamare le funzionalità dello scheduler per proseguire il suo ciclo di vita.
Le sue fasi sono:

- Inizializzazione
- Avvio
- Selezione ed esecuzione processo
- Sospensione processo
- Terminazione processo

Il grafico seguente mostra il ciclo di vita dello scheduler e le transizioni da uno stato ad un altro per mezzo delle sue interfacce.



Inizializzazione scheduler

Fase in cui le strutture di dati dello scheduler sono allocate ed inizializzate per essere pronte al suo avvio. In particolare, è inizializzata la ready queue che conterrà i process control block (PCB), contenenti a loro volta gli stati e priorità dei processi in attesa di essere scelti dallo scheduler.

Avvio scheduler

Fase in cui sono inseriti i primi processi da eseguire nella ready queue e lo scheduler avvia la fase di selezione ed esecuzione del processo.
Da questa fase in poi il controllo dell'esecuzione è gestito solo dai gestori delle eccezioni causate dal processore e dallo scheduler stesso.

Selezione ed esecuzione processo

In questa fase lo scheduler prende il primo PCB disponibile nella ready queue che avrà la priorità maggiore di tutti gli altri, impostando un timer di 3 ms sul processore, entro il quale il controllo dell'esecuzione è assegnato al processo scelto; infine carica lo stato del PCB sul processore, aggiornando l'esecuzione.
Nel caso in cui la ready queue sia vuota, il processore è impostato in stato di HALT.

Sospensione processo

In questa fase il controllo dell'esecuzione è passato da un gestore di eccezione allo scheduler, il quale aggiorna la struttura dati del PCB con il nuovo stato sospeso e ne ripristina la priorità originale.
Prima di inserirlo nuovamente nella ready queue, viene effettuata un'operazione di aging sui PCB in attesa, che consiste nell'incremento delle priorità degli elementi nella ready queue. Questa operazione è necessaria per evitare una situazione di starvation tra i processi.
Infine, il processo sospeso è inserito nella ready queue in base alla sua priorità originale.
Dopo questa fase si avvia la fase di selezione ed esecuzione di un processo per proseguire il ciclo di vita dello scheduler.

Terminazione processo

In questa fase il controllo dell'esecuzione è passato da un gestore di eccezione allo scheduler, il quale rimuove e dealloca il PCB che era in esecuzione e tutta la sua progenie dalla ready queue, o alternativamente, se specificato durante la chiamata della procedura, solo il PCB rendendo orfani tutti i suoi processi figli nella ready queue.
Dopo questa fase si avvia la fase di selezione ed esecuzione di un processo per proseguire il ciclo di vita dello scheduler.

Gestione delle system call

Il meccanismo di gestione delle System Call è implementato nei moduli handler del kernel in esame. Quando un processo richiama l'attenzione del sistema operativo attraverso una system call, il gestore tenta di gestire e soddisfare la richiesta.

Situazioni interessanti di cui può risultare conveniente parlare sono le seguenti:

- Supponiamo che il processo invochi una system call non valida (allo stato dell'arte una system call con identificativo maggiore o uguale a 12 o minore o uguale a 0). Allora il gestore manda in PANIC il sistema.

Il comportamento in esame si rende necessario in quanto la richiesta è per ovvie ragioni insoddisfacibile. Si potrà eventualmente pensare in futuro ad una gestione più accurata della situazione in oggetto, terminando eventualmente solo il processo che ha fatto la richiesta invalida, senza dover bloccare l'intero sistema.

- Supponiamo che l'handler intercetti una richiesta, e il processo chiamante è in modalità utente. Allora il gestore manda in PANIC il sistema.

Il comportamento in esame è voluto in quanto l'unica modalità per cui è possibile soddisfare le system call è la modalità kernel. In futuro questo caso sarà da gestire sollevando una trap.

Allo stato attuale, è implementata solo la gestione della system call `TERMINATE_PROCESS`, che richiama i metodi dello scheduler `scheduler_StateToTerminate()` e `scheduler_StateToRunning()`.

Gestione degli interrupt

Il gestore degli interrupt è implementato nei moduli handler relativi alla rispettiva architettura. Si occupa di gestire gli interrupts causati da dispositivi I/O, Processor Local Timer e Interval Timer. Per fare ciò identifica la sorgente dell'interrupt e lo gestisce diversamente in base a questa. In caso di una richiesta non valida verrà mandato in PANIC il sistema.

In questa fase è stata implementata solamente la gestione degli interrupt causati dall'Interval Timer, comune ad entrambe le architetture. Questa si occupa di salvare lo stato del processore prima dell'interrupt e reinserire il processo nella ready queue dello scheduler tramite `scheduler_StateToReady()` e continuare la schedulazione del prossimo processo tramite il metodo `scheduler_StateToRunning()`.

La principale differenza tra le due architetture riscontrata in questa componente è la necessità di decrementare il program counter del processo di una word quando viene sollevato un interrupt su uARM.

La gestione degli interrupt causati dagli altri dispositivi è stata impostata per essere completata nella fase successiva.

Documentazione tecnica

Handler

Lo sviluppo degli handler di Interrupt e System call e altre funzionalità comuni a entrambe le architetture, sono stati strutturati permettere un interfacciamento generalizzato, nascondendo le varie implementazioni delle funzionalità.
In particolare, le interfacce definite in shared.h permettono agli altri moduli di poter accedere in modo uniforme a certe funzionalità indipendentemente dalla loro realizzazione nelle varie architetture.
Di seguito sono elencati i metodi più interessanti del modulo:

@Brief Gestore SYSCALL/BP. Richiama il gestore specifico per SYSCALL/BP se il registro di causa dell'area SYS/BP è valorizzato con uno dei due valori.

```
void Handler_SysCall(void)
```

@Brief Gestore degli Interrupt.

```
void Handler_Interrupt(void)
```

@Brief Gestore specifico per SYSCALL, che richiama l'implementazione della system call richiesta

@param request: puntatore alla SYS/BP New Area

```
void handle_syscall(state_t *request)
```

@Brief Implementazione della syscall TERMINATE_PROCESS. Termina il processo corrente e tutta la sua progenie. Infine, richiama lo scheduler per proseguire la schedulazione

```
void sys3_terminate(void)
```

Scheduler

Lo scheduler offre semplici interfacciamenti limitati ai singoli stati che esso può assumere durante l'esecuzione del sistema, di cui sono elencati le metodologie di accesso di seguito.

@brief Inizializza lo scheduler

`void scheduler_init()`

@brief Avvia lo scheduler, da richiamare solo per il primo avvio dello scheduler (wrapper di scheduler_StateToRunning(), quindi vedere suoi valori di ritorno e condizioni)

@PreCondition Deve essere richiamato dopo scheduler_init() dopo aver inserito almeno il primo processo, per avviare la schedulazione dei processi

`int scheduler_main()`

@brief Avvia l'esecuzione del prossimo processo nella ready queue per un TIME_SLICE (attualmente di 3ms), impostando tale valore al Interrupt Timer.

- Se la coda dei processi ready non è vuota,
    - Sceglie il processo dalla ready queue con priorità più alta
    - Imposta l'interrupt timer a TIME_SLICE
    - e carica lo stato del processo sul processore
- manda Il sistema in HALT se la coda dei processi è vuota

@PreCondition Prima di chiamare questa funzione, è necessario chiamare scheduler_StateToReady o scheduler_StateToTerminate per aggiornare il contesto dell'ultimo processo eseguito

@return int

- -1 se non è stato caricato lo stato di un nuovo processo, senza cambiare il controllo di esecuzione (malfunzionamento LDST)
- Per ogni altro valore non dovrebbe restituire poichè il controllo dell'esecuzione è passato al processo, se ciò avviene è dovuto ad un errore di LDST.

`int scheduler_StateToRunning()`

@brief

- Aggiorna lo stato del processo fornito al processo corrente;
- effettua un'azione di aging per tutti gli elementi nella ready queue.
- Ripristina la priorità originale del processo che era in esecuzione
- Disassocia il processo in esecuzione dallo scheduler

@PostCondition Dopo questo stato è necessario richiamare scheduler_StateToRunning per procedere con la schedulazione dei processi

@param state: stato del processo che era in esecuzione

@return int

- 1 se non c'è alcun processo tracciato dallo scheduler in esecuzione
- 0 altrimenti se l'inserimento in ready queue è avvenuto correttamente

```
int scheduler_StateToReady( state_t* state )
```

@brief Dealloca il descrittore del processo che era in esecuzione, rimuovendo eventualmente la sua progenie

@return int

- 1 se non c'è alcun processo tracciato dallo scheduler in esecuzione
- 0 altrimenti se è avvenuto tutto correttamente

@param b_flag_terminate_progeny:

- Se TRUE rimuove e dealloca dalla ready queue il descrittore del processo in esecuzione e tutta la sua progenie,
- Se FALSE rimuove e dealloca solo il descrittore in esecuzione, ma tutti i suoi figli non avranno padre e ogni figlio non avrà fratelli cioè saranno indipendenti tra loro

```
int scheduler_StateToTerminate( int b_flag_terminate_progeny )
```

@brief Inserisce un descrittore di processo nella ready queue
@PreCondition Da utilizzare solo se p non è presente nella ready queue
@param p descrittore del processo

```
void scheduler_AddProcess( pcb_t *p )
```