Relazione Pkaya 2012

Componenti Gruppo:

Alessandro Calò
Alberto Scarlino
Manuela Corina
Antonello Antonacci

La fase1 di Pkaya si basa sulla gestione dei Process Control Blocks (pcb) e dei descrittori di semafori (semd).

Sono stati realizzati i seguenti moduli:

pcb.c

In questo modulo viene gestita l'inizializzazione, l'allocazione e la deallocazione e la gestione delle code e degli alberi dei pcb utilizzando le seguenti procedure:

Liste di pcb

La funzione **initPcbs(void)** inizializza la lista dei pcb liberi. I pcb disponibili sono presenti nel vettore **pcbFree_table**. Tale lista può essere modificata con l'inserimento di un elemento tramite la funzione **void freePcb(pcb_t *p)** o la rimozione di un elemento tramite la funzione **pcb_t *allocPcb(void)** usando, a sua volta, la macro *container_of()* il quale preleva il pcb dalla lista, lo rimuove e lo restituisce.

Code di pcb

La funzione **mkEmpyProcQ(struct list_head *head)** inizializza una coda di pcb vuota e ne viene effettuato il controllo su di essa tramite la funzione **emptyProcQ(struct list_head *head)**.
Per inserire un pcb viene utilizzata la funzione **insertProcQ(struct list_head *head, pcb_t *p)** il cui inserimento deve avvenire tenendo conto della priorità di ciascun pcb (con il campo **priority**). La coda dei processi deve essere ordinata in ordine decrescente.
Per rimuove il primo pcb dalla coda dei pcb viene utilizzata la funzione **removeProcQ(struct list_head *head)**.
Se si volesse rimuovere un determinato pcb dalla coda dei pcb si fa uso della funzione **outProcQ(struct list_head *head, pcb_t *p)**.
E' possibile avere la testa della coda dei pcb tramite la funzione **headProcQ(struct list_head *head)**.

Alberi di tcb

Gli alberi vengono gestiti tramite liste.
La funzione **emptyChild(pcb_t *this)** controlla se il pcb non ha figli ed è possibile assegnare un pcb figlio ad un pcb genitore tramite la funzione **insertChild(pcb_t *prnt, pcb_t *p)**.
Se si volesse rimuovere il primo figlio di un pcb genitore specificato si fa uso della funzione **removeChild(pcb_t *p)**, mentre per rimuovere dalla lista un figlio specificato si utilizza la **outChild(pcb_t *p)**.

asl.c

In questo modulo viene gestita l'inizializzazione, l'allocazione e la deallocazione dei descrittori di semafori attivi o inattivi utilizzando le seguenti procedure:

La lista dei descrittori di semafori liberi o inutilizzati è identificata da **semdFree**,la cui testa è presente in **semdFree_h**.
La lista dei descrittori di semafori attivi è identificata da **semd_h**.

La funzione **initASL()** inizializza la lista dei semdFree in modo da contenere tutti gli elementi dell'array **semdTable** e si può allocare un semaforo rimuovendolo dalla lista dei semdFree tramite **allocSemd(struct list_head *head)**.

La funzione **getSemd(int key)** restituisce il puntatore al descrittore di semaforo la cui chiave è pari a Key.
Il pcb viene inserito nella coda dei processi attraverso la funzione **insertBlocked(int key, pcb_t *p)** verificando che sia presente nella lista e qualora non fosse presente si alloca un nuovo semaforo dalla lista di quelli liberi (semfFree) e lo inserisce nella lista.

La rimozione avviene in due modi differenti:
removeBlocked(int key) che rimuove il primo pcb dalla coda dei processi bloccati, cercandolo attraverso il campo **s_ProcQ** e attraverso la funzione **outBlocked(pcb_t *p)** che rimuove un pcb dalla coda dei semafori su cui è bloccato, e se non fosse presente in tale coda restituisce errore.

Di conseguenza troviamo la funzione **headBlocked(int key)** che restituisce il puntatore al pcb che si trova in testa alla coda dei pcb associata al semaforo.
Infine, la funzione **outChildBlocked(pcb_t *p)** rimuove dalla coda dei Pcb un figlio specificato.

SCELTE PROGETTUALI

Per evitare dei *"warning",* incompatibilità con le liste di linux presenti nel file "*listx.h*", è stato scelto di non inserire i flag "*-pedantic"* e *"-ansi"* nel comando di compilazione.

Il progetto è suddiviso nelle seguenti cartelle :

./Pkaya2012: contiene i file di creazione, "makefile", "relazione", "Schemi ", Thread Queue, Message Queue e Thread Tree", "members".
I seguenti sono file di configurazione creati dalla macchina virtuale umps2 "kernel.core.umps", "kernel", "disk0.umps2", "tape0.umps2", "kernel.stab.umps", "term0.umps","Kaya2011".

./umpsinclude: contiene i file header "listx.h", "const.h", "types11.h", "uMPStypes.h", "base.h", "cost11.h".

./umpsInclude/ePhase1: contiene i file di definizione "msg.e", "tcb.e".

./supporto: file della distribuzione di μMPS2
"elf32ltsmip.h.umpscore.x","crtso.o""libumps.o"

./phase1: contiene i file sorgenti "msg.c", "tcb.c", "p1test.0.2.1.c", "makefile".

FASE 2

La fase 2 di pKaya si basa sulla gestione di eccezioni (BreakPoints, PgmTrap,TLB Exceptions), syscall, interrupt sulla base delle strutture dati realizzate in Fase 1; tenendo conto inoltre che la gestione del sistema viene gestita su un ambiente multiprocessore ed evitando Race Condition.

Sono stai realizzati i seguenti moduli:

initial.c

Questo modulo si occupa dell´inizializzazione del sistema.
Nel popolare 4 nuove Aree nella ROM Reserved Frame, tramite la funzione **popolaArea (memaddr area, memaddr gestore)**, vengono salvati l´area passata come parametro (con la variabile state_t); lo stato corrente del processore tramite la funzione **STST**; viene impostato il registro PC con l´indirizzo **SP** a **RAMTOP**.
Un ulteriore registro Status verrá impostato per mascherare tutti gli interrupt, disattivare la memoria virtuale ed essere in kernel-mode.
Vengono,inoltre, richiamate le funzioni di Phase1 **initPcbs()** e **initASL()**, che inizializzano le strutture dati dei Pcb e dei Sem.
Verranno poi inizializzate tutte le variabili globali utili al funzionamento dell´intero progetto quali: il contatore dei processi attivi **processCounter**, il contatore dei processi bloccati per I/O **soft_blockCounter** e il timer **pseudo_clock** tutti impostati a 0.
Si procede con la creazione e inizializzazione dei Semafori uno per ogni device inizializzati a 0 e uno per lo pseudo-clock timer.
É instanziato il processo di gestione del test tenendo conto delle liste di inizializzazione dei processi, e inizializzazione della CPU con la funzione **void INITCPU**.
Successivamente viene settato PC all´entry-point del test con **pstate** ponendo Interrupt attivati smascherati, Memoria Virtuale spenta, Kernel-Mode attivo e process local timer abilitato. Si procede poi, con l´inserimento del processo nella lista dei processi pronti,la **readyQueue** incrementando il contatore dei processi pronti; verrá settato il tempo di inizio del timer tramite la macro GET_TODLOW che provvede a convertire il tempo del tickTimer in microsecondi concludendo con la chiamata allo **scheduler()**.

scheduler.c

Questo modulo gestisce il context switch tra processi. Ad ogni processo é asseganto un time-slice di 5 millisecondi, inoltre rileva eventuali stati di deadlock.
In particolare lo scheduler svolge due compiti: il primo é quello di tenere sotto controllo l´aggiornamento del tempo trascorso dal punto di partenza del processo sulla CPU all´ultimo pseudo-clock tick tramite il campo **time_cpu**, settando l´interval Timer con il minor tempo tra lo pseudo-clock tick rimanente e il time slice tramite la macro **SET_IT**. Infine verrá caricato lo stato del processo corrente attraverso la macro **LDST**.

Il secondo compito dello scheduler consiste nell´effettuare un controllo sulla coda dei processi verificando se é vuota e in tal caso provoca un arresto del sistema con la chiamata **HALT()**. Se vi sono presenti dei processi nel sistema ma nessuno di questi é bloccato allora si é in uno stato di deadlock e verrá invocato l´arresto di emergenza con la chiamata PANIC(). Altrimenti, se vi sono presenti dei processi e qualcuno di questi é bloccato, allora si é in uno stato di attesa e si cercherá di sbloccarli attraverso gli interrupts attivati e non mascherati. Fatto ció, si preleva il primo processo e si calcola il quanto di tempo trascorso della CPU e successivamente verrá caricato lo stato del processo corrente.

syscall.c

Questo modulo si occupa nell´implementare una serie di servizi per conto del nucleo.
Inizialmente vi é una funzione **void saveState (state_t *current, state_t *old)** che provvede a salvare lo stato corrente in un nuovo stato passato per parametro.
Le syscall in causa sono le seguenti:

CREATEPROCESS

Determina la creazione di un processo figlio del processo chiamante da inserire nella readyQueue utilizzando una funzione della fase precedente (**allocPcb()**). Passando come parametri l´indirizzo fisico dello state_t del nuovo processo e la prioritá del nuovo processo. Se la creazione va a buon fine ritorna 0 altrimenti ritorna errore (-1).

CREATEBROTHER

Determina la creazione di un processo fratello del processo chiamante. Passando come parametro l´indirizzo fisico dello state_t del nuovo processo e la prioritá del nuovo processo. Se la creazione va a buon fine ritorna 0 altrimenti ritorna errore (-1).

TERMINATEPROCESS

Termina il processo corrente e tutta la sua progenie. Se il currentProcess é nullo allora restituisce errore **PANIC**, altrimenti termina tutti i processi figli del processo corrente liberando tutte le risorse utilizzate e verificando se un processo é bloccato su qualche semaforo. Se cosí fosse lo si elimina dal semaforo stesso e si provvederà quindi alla terminazione del processo corrente.

VERHOGEN

Esegue una ¨V¨, cioé un incremento del valore, sul semaforo con chiave semKey passato per parametro, in caso in cui il valore del semaforo era negativo allora rimuove il processo e lo inserisce nella readyQueue.

PASSEREN

Esegue una ¨P¨, cioé un decremento del valore, sul semaforo con chiave semKey passato per parametro. Nel caso il valore sia negativo inserisce il processo corrente in coda al semaforo specificato.

GETCPUTIME

Aggiorna il tempo di cpu (in microsecondi) usato dal processo corrente indicando per quanto tempo è stato nella CPU, accedendo al campo **time_cpu** e aggiornando con la differenza tra il tempo attuale e quello di partenza della CPU (tramite la variabile **todProc**).

WAITCLOCK

Esegue una ¨P¨ sul semaforo associato allo pseudo-clock timer. La V su tale semaforo deve essere eseguito dal nucleo ogni 100 millisecondi (tutti I processi in coda su tale semaforo devono essere sbloccati).

WAITIO

Esegue una ¨P¨ sul semaforo associato al device identificato dai parametri intNo, dnum, waitForTermRead. Tale funzione ritorna lo stato del device specificato tenendo presente il numero di device.

SPECPRGVEC

Consente di definire gestori di PmgTrap per il processo corrente passando come parametri: l´indirizzo della vecchia area in cui salvare lo stato corrente del processore e l´indirizzo della nuova area del processore che verrá utilizzata qualora si verichi una PmgTrap.

SPECTLBVEC

Consente di definire gestori di TLB Exception per il processo corrente passando come parametri: l´indirizzo della vecchia area in cui salvare lo stato corrente del processore e l´indirizzo della nuova area del processore che verrá utilizzata qualora si verichi una TLB Exception.

SPECSYSVEC

Consente di definire gestori di SYS/BP Exception per il processo corrente passando come parametri: l´indirizzo della vecchia area in cui salvare lo stato corrente del processore e l´indirizzo della nuova area del processore che verrá utilizzata qualora si verichi una SYS/BP Exception.

PRGTRAP

Viene richiamata nel momento in cui viene sollevata un'eccezione di tipo prgtrap individuando il valore della causa salvandolo nella OldArea. Se non è stata eseguita, allora il processo viene terminato e si effettua una chiamata allo scheduler. Altrimenti viene salvato lo stato nella OldArea del processo (dedicata alle pgmTrap) per l'esecuzione.

TLBTRAP

Viene richiamata nel momento in cui viene sollevata un'eccezione di tipo tlbtrap individuando il valore della causa salvandolo nella OldArea. Se non è stata eseguita, allora il processo viene terminato e si effettua una chiamata allo scheduler. Altrimenti viene salvato lo stato nella OldArea del processo (dedicata alle pgmTrap) per l'esecuzione.

SYSBPHANDLER

La funzione **saveState** provvede a salvare lo stato corrente in un nuovo stato passato per parametro. Nel momento in cui viene sollevata un'eccezione si rende necessario caricare lo stato della vecchia area nello stato del processo corrente.
Verrà incrementato di 4 il registro PC dato che esso punta all'indirizzo dell'istruzione che ha causato l'eccezione e restituisce il controllo al processo che ha richiesto la syscall.
A questo punto si verifica il tipo di eccezione avvenuta (tramite la macro **CAUSE_EXCCODE_GET**) che la recupera dal registro "cause" e la modalità in cui è avvenuta.
Vengono salvati i parametri delle syscall (contenuti nei registri da a1 a a3), mentre dal registro a0 si estrapola la syscall stessa. Se in quest'ultimo non vi è presente nessuna delle syscall si controlla se è stata eseguita una **SPECSYSVEC** e in tal caso si termina il processo corrente. Ogni syscall una volta terminata chiamerà lo **scheduler()**.
Nel caso in cui il gestore di eccezioni SYS/BP non riconosca la causa avviene un **PANIC()**.

interrupts.c

Diverse cause possono scatenare interrupts. Per vedere quale tipo di interrupts è avvenuto, si usa la macro **CAUSE_IP_GET**.
Si utilizza **Processor Local Timer** (**PLT**) timer locale ad ogni processore (1 per ogni processore), **Interval Timer** timer del bus di sistema identificando innanzitutto la sorgente dell´interrupt e un ack nel registro del device.
Vengono tenuti in considerazione i bit che indicano i device con interrupt pendenti tramite la **PENDING_BITMAP_START** che restiuirà l'indice del device.
Nello specifico si effettua una V sul semaforo associato al device e restuituisce lo stato del device al processo. Nel caso la V non sblocchi alcun processo bisogna salvare lo stato del dispositivo.
Una volta terminate tali operazioni si ritorna allo **scheduler()**.

SCELTE PROGETTUALI

Abbiamo commentato il controllo sul return della waitIO alla fine della funzione print del test perchè, come lei ben sa, abbiamo avuto dei problemi con il valore da restituire nella waiIO.
Abbiamo implementato un processo wait, del file wait.c, per l'attessa dei processori che non hanno processi da eseguire.
Per evitare dei *"warning",* incompatibilità con le liste di linux presenti nel file "*listx.h*", è stato scelto di non inserire i flag "*-pedantic"* e *"-ansi"* nel comando di compilazione.

Il progetto è suddiviso nelle seguenti cartelle :

./Pkaya2012: contiene i file di creazione, "makefile", "relazione", "Schemi ", Thread Queue, Message Queue e Thread Tree", "members".
I seguenti sono file di configurazione creati dalla macchina virtuale umps2 "kernel.core.umps", "kernel", "disk0.umps2", "tape0.umps2", "kernel.stab.umps", "term0.umps","Kaya2011".

./umpsinclude: contiene i file header "listx.h", "const.h", "types11.h", "uMPStypes.h", "base.h", "cost11.h".

./umpsInclude/ePhase2: contiene i file di definizione "initial.e", "scheduler.e" "syscall.e", "interrupt.e", "wait.e".

./supporto: file della distribuzione di μMPS2
"elf32ltsmip.h.umpscore.x","crtso.o""libumps.o"

./phase2: contiene i file sorgenti "initial.c", "scheduler.c", "syscall.c", "interrupt.c", "wait.c", "p2test2012_v1.0.c", "makefile".