PROGETTO DI SISTEMI OPERATIVI
FASE 2

2013

Progetto realizzato da:
Davide Aguiari – 0000 411115
Jacopo Giacò – 0000 393176
Marco Trotta – 0900 045968

Introduzione

Questa fase consta dei seguenti file:

main.c
handler.c
interrupts.c
scheduler.e
utils.c
p2test.c

utils.h
main.h
const13.h
const13_customized.h
interrupts.h
scheduler.h
handler.h

Makefile

Scelte Progettuali

Boot

Il progetto compilato genera un file kernel.core.umps utilizzabile attraverso l'emulatore uMPS2.
La fase due implementa la fase uno con una nuova astrazione del kernel e consta di:

- uno scheduler
- le syscall lato kernel
- la gestione degli interrupt
- la gestione delle eccezioni (TrapHandler, TLB)

Il main() si occupa di inizializzare le strutture di cui avremo bisogno dal boot in poi:

1. Inizializziamo le aree della ROM. La CPU0 sarà inizializzata nella ROM stessa, tutte le altre CPU verranno allocate nella RAM per gestire gli interrupt, le eccezioni TLB/Trap e le syscall

2. Inizializziamo le strutture di fase1 (semafori e pcb)
3. Inizializziamo le variabili (Process Count, Soft-block Count, Ready Queues, e Current Process) ed i semafori: tra questi i semafori dei device (disk, tape, ethernet, printer, terminal) e lo pseudoclock (il timer del bus)
4. Inizializziamo i semafori delle critical section
    - Scheduler
    - SoftBlock, che tiene conto dei processi bloccati sul intero sistema
    - P e V
    - PseudoClock

Abbiamo deciso di utilizzare 4 semafori mutex per evitare un big-kernel lock e per specializzare il ruolo dei semafori su particolari punti critici del sistema. Questi semafori sono tutti inizializzati a 1 permettendo al primo processo di non bloccarsi.

Scheduler

Lo scheduler si preoccupa di garantire la corretta ed equa esecuzione dei processi (finite progress) attraverso un meccanismo di round-robin. Estrae dalla readyQueue il pcb con priorità massima (situato in testa alla coda, con valori da 0 a 19), imposta le flag giuste (user/kernel mode, maschera interrupt, macchina virtuale, timer) e lo mette in esecuzione con un timeSlice di 4ms. Lo scheduler si preoccupa anche di verificare eventuali situazioni di deadlock o di idle. Se non vi sono più processi in esecuzione, lo scheduler spegne la sistema.

Lo scheduler viene inizializzato dentro init() che inizializza tutti i processori:

1. Lo stato di ogni processo viene settato con gli interrupt disabilitati, kernel mode e macchina virtuale spenta. Il program counter inizializzato a schedule() e lo stack pointer al primo frame disponibile.
2. Allochiamo il pcb per l'entry point (nel nostro caso test()) e lo inseriamo nella readyQueue del CP0

Qualora sia presente un processo nella readyQueue, in una sezione di mutua esclusione (mutex scheduler)

1. Aumentiamo la priorità dinamica di uno, per tutti i processi rimasti nella coda per evitare starvation.
2. Prepariamo il processo da mandare in esecuzione: accendendo il timer, ripristinando la sua priorità statica decisa nel momento della sua creazione, scrivendo nel pcb l'informazione su quale CPU andrà in esecuzione
3. Salviamo il tempo di avvio e settiamo il timeSlice di default (4ms)
4. Mandiamo in esecuzione attraverso la LDST()

Se la readyQueue è vuota ci sono tre casi:

1. Ci sono processi attivi e non sono bloccati allora è deadlock e quindi andiamo in PANIC()
2. Ci sono processi attivi e ci sono processi bloccati oppure non ci sono processi attivi ma stiamo considerando una Cpu diversa da 0, allora si mette in WAIT() aspettando un interrupt
3. Non ci sono processi attivi ma stiamo considerando la CPU0 allora spengo (HALT())

Interrupts

La gestione degli interrupts avviene attraverso la richiesta di due informazioni preliminari. La getCAUSE() che legge il registro cause che ci dice quale interrupt line è pendente a la getPRID() che ci dice su quale Cpu stava girando il processo mentre interveniva l'interrupt.

I tipi di interrupt sono 8 ed in ordine di priorità decrescente da 0 a 7 su: Inter processor, Local timer, Bus interval, Disk, Tape, Network, Printer, Terminal.

INT_IPI: Inter Processor interrupt
La gestione di questo interrupt prevede il semplice ack nell'indirizzo apposito (0x10000400).

INT_LOCAL_TIMER: Processor Local timer
Questo interrupt viene sollevato nel momento in cui un processo finisce il timeSlice. L'interrupt si preoccupa di ripristinare a 4ms il timer.

INT_TIMER: Bus Interval timer
Questo interrupt viene sollevato nel momento in cui scade lo Pseudo clock timer.
Viene fatta una V per ogni processo bloccato sul semaforo dello pseudoClock e quindi reinserito nella propria readyQueue.
Questo interrupt è protetto tramite mutua esclusione con la mutex MUTEX_CLOCK.

INT_DISK, INT_TAPE, INT_UNUSED, INT_PRINTER
L'interrupt viene sollevato quando è stata completata un'operazione su questi device.
Inizialmente si cerca quale device ha scatenato l'interrupt tramite la findDeviceNumber sulla bitmap appropriata. Si preoccupa quindi di sbloccare l'eventuale processo bloccato sul semaforo del device; in tal caso viene reinserito nella readyQueue.
Infine viene fatto l'ack sul device.

INT_TERMINAL
L'interrupt viene sollevato quando è stata completata un'operazione su uno dei terminali.
Inizialmente si cerca quale device ha scatenato l'interrupt tramite la findDeviceNumber sulla bitmap appropriata. Viene poi distinto il ruolo del terminale: read o write verificando quale dei due sia in stato ready.
Infine viene sbloccato l'eventuale processo bloccato sul semaforo e fatto l'ack sul terminale.
Nel caso in cui non sia stata fatta ancora la syscall, l'interrupt salva in una struttura globale (device_write/read_response) lo status del terminale preso in considerazione.

Infine viene verificato se il processo chiamante è stato bloccato o no dall'interrupt; nel caso non sia bloccato viene reinserito nella readyQueue dopo aver ripristinato la oldArea.

Ogni CPU gestisce ha il proprio gestore degli interrupt: non è quindi solo CPU0 a gestirli.

Syscall e Exceptions Handlers

Il modulo handler contiene le funzioni di gestione delle syscall, delle trap exception e delle tlb execption.

Il syscallHandler() è uno switch sul registro a0 del processore chiamante (current_process[cpuID]-

>p_s.reg_a0), dove è contenuto l'identificativo della system call da gestire.

La SYSTEM CALL 1 è la CREATE PROCESS che, quando viene chiamata, crea un processo come figlio del processo chiamante.

La SYS1 ha 3 parametri: lo stato del processo, la priorità, e il numero del processore nel quale avviare il processo creato inserendolo nella apposita ready_queue.

Se nella fase di allocazione, la pcbTable non ha pcb liberi, viene ritornato un valore di errore (-1) tramite il registro v0, altrimenti viene inserito in ready_queue; il processo creato viene, tramite la insertChild, inserito come figlio del processo chiamante.
Abbiamo deciso di accendere una nuova CPU ogni qual volta venga inserito per la prima volta un processo nella readyQueue associata, tramite la INITCPU.

La SYSTEM CALL 2 è la TERMINATE PROCESS che, chiamata la funzione killMe() uccide il processo chiamante e tutta la sua progenie.

Il compito è lasciato alla outchildBlocked (funzione ricorsiva di phase 1) che si occupa di slegare il chiamante da eventuali puntatori a parent e sibling, di chiamare la freePcb e di terminare ricorsivamente anche ogni processo figlio (terminateProcess()).

In dettaglio, la outChildBlocked controlla se il processo corrente è bloccato su un semaforo diverso da quello dei device e, in tal caso, lo rimuove dalla coda dei processi bloccati su quel semaforo. Dopo di che, con la outChild viene rimosso il processo dalla lista dei figli del padre e viene chiamata la terminateProcess, che ricorsivamente visita le foglie dell'albero (la parte della progenie) e le elimina.

Questa fase è protetta dalla mutex Scheduler, per impedire che venga erroneamente rilevato un deadlock durante le operazioni di incremento/decremento di contatori globali e di allocazioni di pcb.

La SYSTEM CALL 3 è la VERHOGEN, che incrementa il semaforo specificato nel registro a1.

Se ci sono processi bloccati, il primo viene tolto dalla coda e messo in readyQueue.

Questa operazione è protetta dalla mutex PV che impedisce che vengano fatte P e V contemporaneamente sulle stesse strutture.

La SYSTEM CALL 4 è la PASSEREN, che decrementa il semaforo specificato nel registro a1.

Se il valore del semaforo è negativo, il processo viene bloccato e accodato inserendolo nella coda del semaforo tramite la funzione insertBlocked(), che lo accoda.

In dettaglio, se il semaforo corrispondente non è presente lista dei semafori attivi, alloca un nuovo SEMD dalla lista di quelli liberi (semdFree) e lo inserisce nella ASL; altrimenti se non è possibile allocare un nuovo SEMD. perché la lista di quelli liberi è vuota, restituisce TRUE. In tutti gli altri casi, restituisce FALSE.

Se il processo viene bloccato viene, di conseguenza, aumentato il softBlock counter e il controllo ritornato allo scheduler; altrimenti, se il valore del semaforo decrementato è >= 0, il processo riprende il controllo.

Questa operazione è protetta dalla mutex PV che impedisce che vengano fatte P e V contemporaneamente sulle stesse strutture.

La SYSTEM CALL 5 è la SPECTRAPVEC che si occupa di specificare gestori di eccezioni personalizzate.

Viene fatto uno switch sul registro a1 dove viene specificato che tipo di eccezione viene sollevata:

- 0 - Tlb execption
- 1 – PgmTrap execption
- 2 – Syscall/Breakpoint execption

Per permettere il salvataggio dei nuovi handler, abbiamo introdotto in pcb_t un array di state al fine di salvare, tramite la SYS5, i gestori tramite la copyState().

Nel registro a1 viene specificato il tipo di eccezione, nel registro a3 abbiamo la NEW_AREA (da utilizzare nel caso si verifichino exceptions o PGMTRAP) e nel registro a2 la OLD_AREA.

Nel caso si tenti di chiamare al più una volta la SYS5 sugli stessi handler, il processo viene immediatamente terminato e il controllo tornato allo scheduler.

La SYSTEM CALL 6 è la GETCPU TIME, che ritorna nel registro v0 il tempo di CPU usato dal processo corrente, sottraendo all'istante in cui viene invocato (usando la GET_TODLOW), l'istante di partenza del processo.

Per recuperare l'istante di partenza del processo, abbiamo aggiunto un campo alla struttura del PCB, chiamato startTime, inizializzato la prima volta che viene tolto dalla readyQueue.

La SYSTEM CALL 7 è la WAITCLOCK che semplicemente esegue una P sullo PSEUDO CLOCK di sistema e, nel caso, si blocca sul semaforo in attesa dello scadere di SCHED_PSEUDO_CLOCK.

la SYSTEM CALL 8 è la WAITIO permette al chiamante di aspettare un INTERRUPT da un device, bloccandosi, se necessario, sul relativo semaforo.

Il device è idenficato da intNo, dnume e waitForTermRead.

Viene quindi cercato il device tramite uno switch sul registro a1; successivamente viene fatta una P() sul semaforo relativo e il processo viene bloccato se precedente all'interrupt relativo.
Il registro a3 è riservato al caso Terminal e indica se è stata richiesta un'operazione di lettura o di scrittura.

Nel caso in cui avvenga un interrupt prima della syscall, la P() non bloccante permetterà al processo di recuperare lo status dalla struttura globale device_read/write_response su cui è stato fatto uno “store off” dall'interrupt.

Nel caso in cui una istruzione privilegiata è stata chiamata in user-mode, viene richiamato il trapHandler() che controlla se il processo che ha chiamato la system call, abbia gia chiamato la sys5 per

un PGM TRAP: lo stato del processore viene copiato dalla PGMTRAP old area nell'area dello stato del processore apposito. Nel caso in cui il processo non abbia chiamato la SYS5, il trapHandler uccide il processo, come se fosse stata chiamata la SYS2 (TERMINATE PROCESS).

Un controllo simile è utilizzato anche nel caso in cui venga chiamata una syscall fuori range.

TlbHandler() è il gestore delle TLB execption, eccezione sollevata dal sistema nel caso in cui μMPS2 fallisca nel tentativo di tradurre un indirizzo virtuale in fisico:

come da specifica, viene salvata l'area TLB_OLDAREA nella struttura apposita del pcb_t (nel caso monocore, altrimenti viene salvata la new_old_areas[cpuID][TLB_OLDAREA_INDEX]) e caricata la TLB_NEW area precedentemente impostata tramite la SYS5.
Altrimenti il processo viene killato.

TrapHandler(), che si preoccupa di gestire le operazioni illegali, è gestita in modo analogo sulle rispettive area.

La gestione dei breakpoint, in questa fase, non è stata implementata.

Al termine della gestione delle syscall il program counter viene aumentata di una WORD (4byte) e quindi ricaricata.