INTRODUZIONE

Il progetto è stato suddiviso in due fasi, nella fase 1 abbiamo costruito le strutture dati che poi sono servite a supporto della fase 2. Nella fase 2 abbiamo costruito il nucleo del kernel, il tutto sopra all'architettura dell'emulatore Uarm progettato da Marco Melletti.

STRUTTURA PROGETTO

Phase 1 è strutturata in 3 file .c e 5 header file:
pcb.h : definizione struttura pcb.
alberi.h -> alberi.c : insieme funzioni per gestione della struttura albero del pcb.
coda.h -> coda.c : insieme funzioni per gestione della coda di pcb.
hash.h ->hash.c : funzioni relative all'hashtable di semafori.
const.h : contiene le costanti (anche per phase2).

Phase 2 è strutturata in 6 file .c e 5 header file:
init.h->init.c : il main del programma che inizializza il sistema operativo.
interrupt.h -> interrput.c : gestione delle interrupt del sistema operativo (devices e timer).
scheduler.h -> scheduler.c : scheduler del sistema operativo.
systemcall.h -> systemcall.c : lista delle funzioni per le system calls.
traphadlers.h -> traphandlers.c : gestione delle eccezioni.

DESCRIZIONE SORGENTI PHASE2

init.c

Serve ad inizializzare il nucleo; come prima cosa si occupa di gestire le quattro aree di memoria dedicate ad interrupt e trap; Queste ultime sono impostante settando il program counter relativo alla funzione handler relativa, sp a all'ultimo frame della ram e vengono disabilitati gli interrupt. Vengono inizializzate le strutture dati di phase1 e le variabili utilizzate durante la vita del nucleo. Viene costruito il primo processo, dedito all'esecuzione del test, in base alle specifiche fornite chiamando poi lo scheduler per far partire il primo processo.

scheduler.c

É suddiviso in tre aree logiche:

La prima si occupa di fare l'update delle priorità di tutti i processi nella coda ready ogni 10ms Questo avviene attraverso l'utilizzo della funzione di phase1 forallprocQ che ci permette di eseguire la funzione di updatepriority su tutti i processi.

La seconda area si occupa di gestire il processo attivo controllando quanto tempo di esecuzione gli rimane e settando quindi il timer di conseguenza.

La terza area si occupa di gestire la situazione in cui non ci sono processi attivi (ad esempio a seguito di un timer interrupt) rimuovendo dalla coda ready se è presente il prossimo processo da eseguire e inserirlo in memoria. La struttura di phase1 prevede che l'inserimento di un processo venga eseguito tenendo conto della sua priorita'.
Nel caso in cui non vi siano processi attivi vi sara' un controllo sulle variabili "contatore" che si occupano di gestire i processi attivi e i processi softblocked (individuando eventuali deadlocks).

Prima di settare il timer viene controllato lo pseudo-clock, nel caso il timeslice dello scheduler sia maggiore del tempo rimasto prima del prossimo tick viene settato un timer che consenta di attivare il tick esattamente 100ms dopo il tick passato.

traphandlers.c
Contiene i 3 gestori delle eccezioni che possono verificarsi durante l'esecuzione.
Il gestore di system calls che si occupa di attivare la funzione relativa alla system call richiesta (definite in systemcalls.c). Nel caso in cui la venga richiesta un syscall > 10 verra' passata, se presente, al livello superiore. Vi e' anche un controllo sull'utente che ha richiesto la system call (usermode?).

Il gestore di trap TLB e il gestore trap programma PGM si occupano di passare a livello superiore la gestione delle eccezioni. Nel caso in cui non siano presenti i gestori di livello superiore si esegue una sys2 NULL.

systemcalls.c
Contiene le funzioni che implementano le systemcall specificate dalla consegna.
- (sys1) crateprocess : viene allocato un nuovo pcb, viene inserito come figlio del processo chiamante e messo nella coda ready.
  Viene anche impostata una variabile start_priority utilizzata per reimpostare in futuro la priorita' originale in seguito agli update per aging.

- (sys2) termianteprocess : dato un pcb p, viene poi controllato se il processo padre era in attesa con una waitchild e in seguito si elimina/libera il processo e la sua prole attraverso una funzione ricorsiva liberatutti(pcb_t p).

- (sys3) semp : Se il semaforo e' a zero viene inserito, attraverso l'uso delle funzioni scritte in phase1, il processo corrente nella lista di processi bloccati sul semaforo specificato. Viene evitata la coda per priorita' salvando la priorita' del processo e inserendolo fra i processi bloccati con priorita' zero.

- (sys4) semv : Se vi sono processi bloccati sul semaforo viene estratto il primo e viene inserito nella coda ready.

- (sys5) spechdl : all'interno di ogni struct pcb_t e' sono presenti dei puntatori che si occupano di riferirsi alle aree new e old dei gestori di eccezioni e vengono settati a null inizialmente.
  Se gia' presenti la system call fa return -1, altrimenti si occupa di assegnare i puntatori.

- (sys6) gettime : vengono restituiti, ai parametri passati, i valori di tempo relativi all'esecuzione del processo. Qui viene non semplicemente restituito il valore attuale ma restituito il valore del tempo all'attuale chiamata di system call gettime()

- (sys7) waitclock : il processo chiamante fa una semp sul semaforo relativo al timer. Ad ogni interrupt di tipo timer verra' controllati se sono passati 100 ms dall'ultimo wake dei processi se si verra' svegliato anche il processo che qua chiama la p.

- (sys8) iodevop : si ottiene, dato il registro command fornito, il numero di device a cui la syscall si riferisce; viene quindi fatta un semp sul semaforo relativo al device e infine viene inviato il comando scrivendo nel registro indicato il comando fornito per parametro.
  Sara' compito dell'interrupt handler gestire il valore di ritorno.

- (sys9) getpids: restituisce nei puntatori forniti i riferimenti di pcb e del pcb del padre.

- (sys10) waitchild :vengono restituiti, ai parametri passati, i valori di tempo relativi all'esecuzione del processo. Qui viene non semplicemente restituito il valore attuale ma restituito il valore del tempo all'attuale chiamata di system call gettime()

interrupts.c

Si occupa di gestire gli interrupt, riconoscendo l'interput line ,con priorità più alta, attiva. Nel caso si tratti di timer viene chiamato il gestore del timer altrimenti quello dei devices generici.

Il timer handler: si occupa di rischedulare i processi attivi reinserendo il processo attuale nella coda ready e chiamando poi lo scheduler. Viene, in questo handler, anche controllato se sia scattato il tick di 100 ms e sblocca eventuali processi bloccati sul semaforo timersem.

Il gestore device una volta attivato si occupa di leggere la bitmap relativa alla linea attiva e ne deduce il numero di processo attivo (0-7). Avendo il numero device possiamo ottenere il registro device su cui operare e possiamo calcolare l'indice globale device da utilizzare per attivare il processo in attesa.
Dato il registro esso sara' strutturato diversamente nel caso si tratti di device standard o device terminal. Nel caso di device terminal e' anche necessario controllare se siamo in lettura o in scrittura. Viene salvato lo status del device gestito come valore di ritorno per il processo sbloccato e successivamente viene chiamato lo scheduler.