Progetto di Sistemi Operativi

Anno accademico: 2013/2014
Docente: Prof. Renzo Davoli

Componenti del gruppo

- Tommaso Ognibene (tommaso.ognibene@studio.unibo.it)
- Alessio Gozzoli (alessio.gozzoli@studio.unibo.it)
- Alessandro Panipucci (alessandro.panipucci@studio.unibo.it)
- Neta Kedem (neta.kedem@studio.unibo.it)

Introduzione

Il presente progetto si basa sul sistema operativo Kaya e sull'emulatore uARM. Risulta strutturato in 2 fasi :

Fase 1. Implementazione delle strutture dati sottostanti, segnatamente il *Process Control Block* (*PCB*) e l'*Active Semaphore List* (*ASL*) .

Fase 2. Implementazione del *lifecycle* di un sistema operativo minimale: inizializzazione, *scheduling*, gestione delle eccezioni e degli *interrupt*.

Entrambe le fasi sono state testate con successo mediante *p1test.c* e *p2test.c*.

Fase 1

La struttura dati *Process Control Block* (*PCB*) consente di rappresentare, tramite un vettore, il numero massimo di processi gestibili in modo concorrente. Le funzioni ad essa associate sono le seguenti:

1. *initPcbs*: inizializza il vettore e la lista dei blocchi liberi;
2. *allocPcb*: estrae e inizializza un *PCB* libero, se disponibile;
3. *insertProcQ*: inserisce un dato *PCB* in una *process queue*;
4. *removeProcQ*: rimuove il primo *PCB* da una *process queue*;
5. *outProcQ*: rimuove un dato *PCB* da una *process queue*;
6. *insertChild*: definisce la relazione padre-figlio tra due dati processi;
7. *removeChild*: rimuove il primo processo figlio da un dato *PCB*;
8. *outChild*: rimuove un dato processo figlio da un dato *PCB*;

Per quanto concerne l'*Active Semaphore List* (*ASL*), è stato scelto di impiegare un *dummy header*, al fine di migliorare la chiarezza del codice. Le funzioni ad essa associate sono le seguenti:

1. *initASL*: inizializza il vettore dei semafori liberi;
2. *insertBlocked*: inserisce un dato *PCB* in un dato semaforo;

3. *removeBlocked*: rimuove il primo *PCB* da un dato semaforo;
4. *outBlocked*: rimuove un dato *PCB* da un dato semaforo;
5. *headBlocked*: restituisce il primo *PCB* contenuto nella *process queue* di un dato semaforo.

Fase 2

La fase 2 si compone di quattro moduli, corrispondenti a 4 *files* distinti:

1. *init.c* ;
2. *scheduler.c*;
3. *interrupts.c*;
4. *exceptions.c*.

INIT.C

init.c è il modulo che concerne l'inizializzazione del nucleo (*booting*). In sintesi:

- Vengono popolate 4 nuove aree nella *ROM Reserved Frame Area* per la gestione delle eccezioni, mediante la funzione *populateNewArea()*. A tal fine, viene salvato lo stato corrente del processore, viene inizializzato il registro SP (*Stack Pointer*) e assegnato al registro PC (*Program Counter*) l'indirizzo dell'*exception handler*. Vengono disabilitati memoria virtuale e *interrupt*, e impostata la modalità *kernel*.
- Vengono invocate *initPcbs()* e *initASL()* per inizializzare le strutture dati della fase 1, e vengono inizializzate le seguenti variabili globali:
  - *ReadyQueue*: la coda dei processi pronti;
  - *CurrentProcess*: il puntatore al processo corrente;
  - *ProcessCount*: il contatore dei processi;
  - *SoftBlockCount*: il contatore dei processi bloccati in attesa di *interrupt*;
  - *ProcessTOD*: tempo iniziale del processo;
  - *TimerTick*: contatore dei *tick* per l'*Interval Timer*;
  - *StartTimerTick*: valore iniziale dei *tick* per lo *Pseudo-Clock*;
  - *PseudoClock*: valore del semaforo per lo *Pseudo-Clock*;
  - *Semaphores*: semafori dei dispositivi per ogni linea di *interrupt*.

  I dispositivi sono: *disk*, *tape*, *network*, *printer*, *terminalR* (per la ricezione) e *terminalT* (per la trasmissione).
- Viene allocato il processo iniziale, denominato *init*, in modalità kernel, con gli *interrupt* abilitati e la memoria virtuale disabilitata.
- Viene incrementato il *Process Counter*, inserito *init* nella *Ready Queue* e invocato lo *scheduler*.

SCHEDULER.C

scheduler.c si occupa, come traspare dal nome, dello *scheduling* dei processi concorrenti. Si distinguono 2 fattispecie:

1. Sussiste un processo corrente:
   - viene aggiornato il *CPU time* del processo corrente;
   - viene salvato il tempo corrente;
   - viene aggiornato il tempo trascorso per il *Timer Tick*;
   - viene impostato il nuovo valore dell'*Internal Timer*;
   - viene caricato il processo corrente e restituita l'esecuzione.
2. Non sussiste un processo corrente:
   - viena controllata la presenza di processi nella *Ready Queue*;
   - se la *Ready Queue* è vuota, si distinguono i casi di terminazione per assenza di processi disponibili, terminazione anomala per presenza di *deadlock*, posizionamento del processore in stato di attesa (*idle state*) per l'addivenire di un *interrupt*.
   - se la *Ready Queue* non è vuota, viene:
     - estratto il primo processo;
     - calcolato il tempo trascorso dall'ultimo *tick* dello *Pseudo-Clock*;
     - azzerato il *CPU time* del processo;
     - impostato il *ProcessTOD*;
     - impostato l'*Internal Timer*
   - viene caricato lo stato del nuovo processo sulla *CPU*, in modo da lanciarne l'esecuzione.

INTERRUPT.C

interrupt.c implementa la gestione degli *interrupt* (*Interrupt Exception Handling*). Il modulo consta delle seguenti 3 funzioni:

1. *intTimer*:
   Si distinguono i casi in cui:
   - il *Time Slice* associato al processo corrente risulta esaurito;
   - il *Time Slice* associato al processo corrente risulta non esaurito;
   - non sussiste un processo corrente.

   Se il *Time Slice* associato al processo corrente risulta esaurito, si inserisce il processo corrente nella *Ready Queue* e si invoca lo *scheduler*.
   Di converso, se il *Time Slice* è ancora valido, viene controllato il valore del semaforo dello *Pseudo-Clock*, al fine di rilasciare tutti i processi bloccati in esso. Nel caso in cui non sussiste alcun processo bloccato, il valore del semaforo viene decrementato.

2. *intDevice*:
   Gestisce gli *interrupt* causati dai dispositivi diversi dal *timer* e dal *terminal*.
   Viene effettuata una *V* sul semaforo corrispondente al dispositivo con priorità più alta, tra quelli affetti da un *interrupt* pendente. In questo modo si risolve il problema della presenza multipla e simultanea di *interrupt*. Inoltre viene salvato nel registro *a1* lo stato del dispositivo.
3. *intTerminal*:
   Gestisce gli *interrupt* causati dal *terminal*.
   Comportamento affine a *intDevice*, tuttavia occorre differenziare tra modalità trasmittente e ricevente, in quanto a tali modalità corrispondono due distinti semafori.

EXCEPTION.C

exception.c afferisce alla gestione delle eccezioni (*TLB*, *PgmTrap*, *SYS/Bp Exception Handling*).

System Call & Break Point Handling
Sono definite le seguenti *system call*:

1. *createProcess*: determina un nuovo processo figlio per il processo chiamante;
2. *terminateProcess*: termina il processo corrente e il relativo sotto-albero dei figli;
3. *verhogen*: esegue una *V* su un dato semaforo;
4. *passeren*: esegue una *P* su un dato semaforo;
5. *specTrapVec*: questa funzione può essere invocata una sola volta per ogni processo. Alla seconda chiamata, il processo viene terminato.
6. *getCPUTime*: aggiorna il tempo trascorso nella *CPU* da parte del processo corrente.
7. *waitClock*: esegue una *P* sullo *Pseudo-Clock*;
8. *waitIO*: esegue una *P* sul semaforo del dispositivo la cui linea di interrupt è passata per parametro.

Si distingue tra richiesta di *system call* in *User Mode* e in *System Mode*. In particolare, se una *system call* privilegiata (ovvero una delle precedenti), viene richiesta da parte di un processo in *User Mode*, allora viene invocata un'opportuna eccezione di tipo *Program Trap*. Di converso, nel caso di *system call* non privilegiate, eseguite sia in *User Mode* sia in *System Mode*, così come nel caso di *Break Point*, occorre distinguere il caso in cui la *system call #5* sia stata precedentemente effettuata oppure no. In quest'ultimo caso il processo verrà terminato, richiamando conseguentemente lo *scheduler*.

Program Trap Handling

Viene salvato lo stato della vecchia area nello stato del processo corrente, se la *system call #5* non è stata ancora effettuata, relativamente a tale eccezione, il processo viene terminato.

TLB (Translation Lookaside Buffer) Exception Handling

Parimenti al caso del gestore delle *Program Trap*, anche qui viene salvato lo stato della vecchia area nello stato del processo corrente, se la *system call #5* non è stata ancora effettuata, relativamente a tale eccezione, il processo viene terminato.