MIKONOS Operating System Project

documentazione phase 1A,B

Alberti Michele
Cozza Giovanni
Pomili Luigi

Febbraio 23, 2008

Introduzione:

Questo documento vuole descrivere, in breve, l'implementazione della phase1 del progetto “MIKONOS Operating System”. Viene dato maggiore spazio alle scelte e politiche implementative riguardanti le strutture dati utilizzate e in generale la struttura del codice.

Layout del codice:

Il tree del progetto, in breve, puo' essere letto nel file README.
Piu' in dettaglio:

```
lso08az13                  /
|
|- AUTHORS                 gli sviluppatori e rispettive mail
|- LICENSE                 licenza di utilizzo
|- Makefile                file per una compilazione guidata, con check
                           dell'architettura host
|- README                  informazioni in breve sul progetto
|- doc/                    documentazione del progetto
   |
   |- DOCphase1AB          questo file

|- h/                      tutti gli header files utili al progetto
   |
   |- base.h               definizioni di tipi, per comodita'
   |- const.h              definizioni di costanti e macro
   |- our_const.h          definizioni di nostre costanti
   |- types.h              definizioni di tipi riguardanti l'emulatore
   |- mikonos.h            definizioni delle strutture dati riguardanti
                           direttamente il progetto di MOKONOS

|- phase1/                 tutti i sorgenti riguardanti phase1A,B
   |
   |- e/                   tutte le interfacce utilizzate in phase1
      |
      |- tcb.e             interfaccia per la gestione dei Thread Control
                           Block e delle code di TCB
      |- tid.e             interfaccia per la gestione dei Thread ID e non
                           non solo

   |- test/                tutti i sorgenti e interfaccie dei test
                           ufficiali
      |
      |- p1atest.c         file di test per la phase1A
      |- p1btest.c         secondo file di test per la phase1A
      |- p1b2test.c        file di test per la phase1B
      |- p1b2test.e        interfaccia per il file di test p1b2test.c
      |- Makefile          file per la compilazione guidata dei test

   |- src/                 tutti i sorgenti da noi sviluppati
      |
      |
```

```
|- tcb.c            gestione Thread Control Block e codi di TCB
|- tid.c            gestione dei Thread ID e funzioni direttamente
                    utilizzabili per gestione dei TCB
|- boot.c           gestione della sequenza di boot di MIKONOS
                    Operating System
|- Makefile         file per la compilazione guidata dei sorgenti
```

Il processo di compilazione e' gestita dal Makefile contenuto nella directory lso08az13. Per avere informazioni dettagliate su tale processo guardare il file README.

Scelte e politiche implementative:

1. Struttura dati tcb_t.

   Sono stati aggiunti due campi alla struttura di base:

   - *t_prev, permette l'utilizzo di code doppiamente linkate per un maggiore efficienza nelle varie operazioni riguardanti le code di TCB.
     Permette funzioni di inserimento/rimozione dei TCB in modo semplice e veloce.
   - numero_bit, permette una maggiore velocita' e facilita' nel processo di deallocazione dei TCB in quanto questo campo, immaginato come maschera di bit, mantiene il bit nella bitmap associato al TCB in fase di allocazione. Grazie a questo campo, il quale risulta leggero nell'economia di occupazione di memoria perche' unsigned int, la freeTcb risulta facile di comprensione e computazionalmente veloce [complessita' costante, O(1)].

   Come viene utilizzato numero_bit?

   Il campo “numero_bit” risulta particolarmente utile nella deallocazione dei TCB. Il processo di deallocazione deve comportare come ultimo stadio l'azzeramento del bit nella bitmap legato al TCB. Grazie a “numero_bit” tutto questo e' stato implementato con complessita' costante, O(1), tramite operazioni di bitwise. Vedere a riguardo la funzione freeTcb.

   Perche'?

   Abbiamo scelto di inserire un campo in piu' in tcb_t, “numero_bit”, perche' ci e' sembrata la soluzione piu' lineare e pratica, nonche' intuitiva da capire. Inoltre sapendo di dover limitare la memoria utilizzata, in quanto programmazione kernel, aggiungere un campo di 4B per ogni TCB (non piu' di 32) e' sembrato il compromesso migliore tra memoria utilizzata – efficienza.

2. #define MAXTHREAD MAXPROC.

   Questa #define contenuta nel file our_const.h permette l'utilizzo di MAXPROC come costante per la scelta del numero massimo di Thread

Control Block attivi nel kernel.
Allo stesso tempo permette l'utilizzo della costante MAXTHREAD all'interno del codice sviluppato, in linea con le specifiche.
Abbiamo scelto di poter supportare un massimo di 32 Thread Control Block lasciando pero' liberta' di poterne avere un numero inferiore.
La scelta di avere un massimo di 32 TCB e' dovuta all'utilizzo di un tipo unsigned int per la variabile “bitmap”.
Vogliamo comunque dare due idee per possibili implentazioni future:

- si puo' pensare ad una bitmap come un array di unsigned int, o char, ottenendo in questo modo una struttura scalabile. Il tutto pero' ci e' sembrato troppo pesante (sia come codice che come soluzione implementativa) e poco lineare facendoci arrivare alla conclusione di non adottare questa implementazione, almeno per l'immediato;
- utilizzare un tipo unsigned long int che incrementi i bit utili nella bitmap a 64. Soluzione che risulta meno scalabile della precedente, ma piu' snella e lineare.

3. Mappatura dei TID ai TCB.

   Il kernel deve permettere l'utilizzo di 255 tid, i quali permettono l'identificazione univoca di un TCB. Abbiamo scelto di implementare la struttura che mappa i TID ai TCB come un array di MAXTID puntatori a strutture tcb_t. E' stata adottata questa soluzione perche' e' di facile utilizzo ma sopratutto risulta di gran lunga la piu' efficiente, anche se comporta un utilizzo maggiore di memoria. Permette la gestione della mappatura dei TID ai TCB in maniera elegante ed intuitiva, nonche' computazionalmente molto efficiente [O(1)].

   Come avviene la gestione della mappatura?

   Tramite la variabile “tid_assegnato” viene mantenuto il numero dell'ultimo TID utilizzato nella piu' recente allocazione di un TCB. Notare che i TID possono essere riutilizzati; per fornire questa possibilita' il calcolo del TID viene eseguito utilizzando la funzione modulo con MAXTID.
   Una volta allocato un TCB gli viene assegnato il corrispondente TID e nell'array “assegna_tid” utilizzato per la mappatura, nella posizione corrispondente a “tid_assegnato”, viene salvato il puntatore alla struttura tcb_t appena allocata.
   Viceversa, quando si dealloca un TCB tramite il suo TID si trova nella posizione TID-esima dell'array “assegna_tid” il puntatore al tcb_t da liberare. Il risultato e' un'implementazione di killTcb e resolveTcb intuitiva ed efficiente [complessita' costante, O(1)].

4. Boot.

   Per quanto concerne la fase di boot del sistema operativo, abbiamo scelto di implementare l'inizializzazione delle New Areas nel ROM Reserved Frame tramite una semplice funzione, per risparmiare codice e scriverlo in maniera piu' pulita. Utilizziamo la funzione STST

definita in /usr/include/uMPS/libumps.e, al fine di eseguire lo “store del processore” per una giusta gestione dell'eccezione.
La costante STATUS_CP0 definita in our_const.h inizializza il registro status di CP0 ad 0x10000000, cioe' kernel mode, CP0 utilizzabile, VM off ed interrupt disabilitati.

5. Makefile.

   Il codice e' fornito di una serie di Makefile per la compilazione “guidata”. Si e' scelto di utilizzare una struttura di Makefile di tipo “gerarchico”, dove il Makefile principale, quello contenuto nella root directory, richiama i Makefile contenuti nelle cartelle src/ e test/. Questo per mantenere i compiti divisi uttuando una politica di scalabilita' e leggerezza di codice.

Ottimizzazioni:

Tutto il codice da noi sviluppato risulta compilabile con flag di ottimizzazione, i quali sono gia' previsti all'interno dei Makefile. Cioe':

```
GCC_OPTIONS_PLUS = -ansi -pedantic -Wall -O2
GCC_OPTIONS = -ansi -pedantic -Wall -O0
```

Ultima nota:

Abbiamo dovuto cambiare nei file di test solo i path ai vari header file e interfacce, per adattare il tutto all'organizzazione del tree di progetto; sappiamo che soltanto questa modifica poteva essere fatta.
Percio' **non** sono state apportate altre modifiche a file di test, come richiesto.