Corso di *Architetture degli Elaboratori* – A.A. 2015/2016

Lista esercizi di progetto:

1. Simulatore di chiamate a procedura
2. Scheduler di processi

Esercizio di progetto (1): simulatore di chiamate a procedura

Sia S una stringa che possa rappresenta una qualsiasi combinazione delle funzioni somma, sottrazione, prodotto e divisione (intera) fra due numeri interi.

Esempi validi di stringa S sono:

- "divisione(5,2)"
- "prodotto(prodotto(3,4),2)"
- "somma(7,somma(sottrazione(0,5),prodotto(divisione(7,2),3)))"
- etc...

Utilizzando QtSpim, scrivere e provare un programma in assembly MIPS che prenda in input la stringa S letta da file (composta da al massimo 150 caratteri), e che:

- implementi le funzioni somma, sottrazione, prodotto e divisione (intera) fra due numeri interi, come procedure MIPS;
- simuli la sequenza delle chiamate alle funzioni nell'ordine in cui appaiono nella stringa in input (da sinistra a destra);
- visualizzi su console la traccia con la sequenza delle chiamate annidate (con argomento fra parentesi) ed i valori restituiti dalle varie chiamate annidate (valore restituito fra parentesi).

Inoltre, mostrare e discutere nella relazione l'evoluzione dello stack nel caso specifico in cui S=*"somma(somma(1,2),somma(3,4))"*.

NOTA BENE:
Il programma, come da specifica, deve simulare la sequenza delle chiamate nell'ordine in cui appaiono nella stringa in input (da sinistra a destra). Quindi, data ad esempio la seguente stringa di input S="prodotto(somma(1,sottrazione(2,3)),4)", il programma dovrà richiamare le procedure (tramite JAL – passando gli argomenti opportuni) nel seguente ordine:
-->prodotto
-->somma (richiamata dentro la procedura prodotto)
-->sottrazione (richiamata dentro la procedura somma)

Esempio di output su console nel caso in cui S=*"divisione(5,2)"*:

--> divisione(5,2)
<-- divisione-return(2) (infatti divisione(5,2)=2) ← ricordarsi che si tratta di divisione intera

Esempio di output su console nel caso in cui S=*"prodotto(prodotto(3,4),2)"*:

--> prodotto(prodotto(3,4),2)
--> prodotto(3,4)
<-- prodotto-return(12) (infatti prodotto(3,4)=12)
<-- prodotto-return(24) (infatti prodotto(12,2)=24)

Esempio di output su console nel caso in cui
S=*"somma(7,somma(sottrazione(0,5),prodotto(divisione(7,2),3)))"*:

--> somma(7,somma(sottrazione(0,5),prodotto(divisione(7,2),3)))
--> somma(sottrazione(0,5),prodotto(divisione(7,2),3))
--> sottrazione(0,5)
<-- sottrazione-return(-5) (infatti sottrazione(0,5)=-5)
--> prodotto(divisione(7,2),3)
--> divisione(7,2)
<-- divisione-return(3) (infatti divisione(7,2)=3)
<-- prodotto-return(9) (infatti prodotto(3,3)=9)
<-- somma-return(4) (infatti somma(-5,9)=4)
<-- somma-return(11) (infatti somma(7,4)=11)

NOTA BENE:

- Le chiamate alle procedure che implementano somma, sottrazione, prodotto, divisione **devono essere realizzate utilizzando l'istruzione jal (jump and link), e rispettando le convenzioni fra procedura chiamante/chiamata**.
- Si assuma che la stringa S in input sia **sintatticamente corretta**.
- Per divisione si intende **divisione intera** (div).
- Il file di input deve chiamarsi "chiamate.txt" e deve risiedere nella stessa cartella in cui è presente l'eseguibile QtSpim.
- La visualizzazione delle tracce così come riportata negli esempi non costituisce parte essenziale dell'esercizio ma solo un ulteriore punto di merito. In particolare, sono ammesse anche visualizzazioni più semplici (e non indentate) di questo tipo:
  prodotto(prodotto(3,4),2)-->prodotto(3,4)-->prodotto-return(12)-->prodotto-return(24)

Esercizio di progetto (2): scheduler di processi

In un sistema operativo, la gestione dei task (o processi) è una questione di primaria importanza che viene spesso riferita col nome di scheduling. Per i nostri scopi, un task è definito dalle seguenti informazioni: un identificatore numerico (ID), un nome (NOME TASK, stringa che può contenere al massimo 8 caratteri), una priorità (PRIORITA', numero compreso fra 0 e 9) e il numero di cicli di esecuzione che richiede per essere completato (ESECUZIONI RIMANENTI). Si supponga che un task possa richiedere al più 99 cicli di esecuzione per essere completato.

Lo scheduler è una funzionalità (del sistema operativo) che gestisce una coda in cui i task vengono mantenuti ordinati in base alla politica di scheduling adottata. In questo esercizio supporremo l'esistenza di due possibili politiche di scheduling:

- a) Scheduling su PRIORITA' (scheduling di default); si ordinano i task in modo decrescente rispetto al valore di PRIORITA' ad essi associato (quindi da 9 a 0). Il task da eseguire per primo sarà quello a cui è stato associato il numero di PRIORITA' più basso.
    - I task con uguale valore di PRIORITA' dovranno essere ordinati in modo decrescente rispetto al loro identificatore numerico ID.
- b) Scheduling su ESECUZIONI RIMANENTI; si ordinano i task in modo crescente rispetto al numero di ESECUZIONI RIMANENTI necessarie per il loro completamento. Il task da eseguire per primo sarà quello che necessita del maggior numero di ESECUZIONI RIMANENTI per essere completato.
    - I task con lo stesso numero di ESECUZIONI RIMANENTI dovranno essere ordinati in modo decrescente rispetto al loro identificatore numerico ID.

Utilizzando QtSpim, scrivere e provare un programma in assembly MIPS che simuli la funzionalità di scheduling dei task di un sistema operativo. In particolare il programma, attraverso un menu di opzioni, dovrà consentire di:

1. inserire un nuovo task, richiedendo la sua priorità, il nome ed il numero di esecuzioni necessarie per il suo completamento (l'identificatore ID è univoco e viene assegnato automaticamente dal programma). Il task inserito dovrà poi essere posizionato nella coda a seconda della politica di scheduling utilizzata;
2. eseguire il task che si trova in testa alla coda;
3. eseguire il task il cui identificatore ID è specificato dall'utente;
4. eliminare il task il cui identificatore ID è specificato dall'utente;
5. modificare la PRIORITA' del task il cui identificatore ID è specificato dall'utente;
6. cambiare la politica di scheduling utilizzata, passando dalla a) alla b) e viceversa;
7. uscire dal programma.

L'esecuzione di un task con ESECUZIONI RIMANENTI = 1 comporta l'eliminazione del task stesso dalla coda, altrimenti il numero di ESECUZIONI RIMANENTI viene decrementato di 1 e l'ordinamento della coda aggiornato a seconda della politica di scheduling utilizzata.

Ad ogni opzione del menu dovrà corrispondere una chiamata alla opportuna procedura utilizzando l'istruzione jal (jump and link). Alla scelta 7) corrisponderà l'uscita dal programma.

Al termine di ogni operazione richiesta dall'utente, si deve stampare la coda risultante dei task sotto forma di elenco ordinato, come nel seguente esempio (i task ad essere eseguiti per primi sono quelli verso il basso):

• caso di scheduling su PRIORITA'

ID	PRIORITA'	NOME TASK	ESECUZ. RIMANENTI
17	7	OUTPOST	3
2	5	SVCHOST	11
5	2	REGSVC	2
4	2	INTERNAT	3

(Il task con ID=4 sarà il primo ad essere eseguito)

• caso di scheduling su ESECUZIONI RIMANENTI

ID	PRIORITA'	NOME TASK	ESECUZ. RIMANENTI
5	2	REGSVC	2
17	7	OUTPOST	3
4	2	INTERNAT	3
2	5	SVCHOST	11

(Il task con ID=2 sarà il primo ad essere eseguito)

Inoltre, mostrare e discutere nella relazione l'evoluzione della memoria dinamica (heap) in alcuni casi di interesse (per esempio in corrispondenza di un inserimento o eliminazione di un task, o nel caso in cui si cambi la politica di scheduling utilizzata).

NOTA BENE:

- Il campo NOME TASK può contenere al massimo 8 caratteri.
- Il campo PRIORITA' è numero compreso fra 0 e 9.
- Il campo ESECUZIONI RIMANENTI è un numero compreso fra 1 e 99.
- Non essendo noto a priori il numero massimo di processi che il sistema operativo può eseguire contemporaneamente**, si richiede di utilizzare l'allocazione dinamica dei dati all'interno dello heap** (utilizzare la chiamata di sistema *sbrk*).

- E’ consigliabile NON impiegare troppo tempo per ottenere la perfetta visualizzazione tabellare dello scheduling: non rappresenta un punto fondamentale dell’esercizio, ma solo un eventuale punto di merito aggiuntivo. A tal proposito, è ammissibile anche una visualizzazione più semplice di questo tipo:

|ID|PRIORITA'|NOME TASK|ESECUZ. RIMANENTI|
|5|2|REGSVC|2|
|17|7|OUTPOST|3|
|4|2|INTERNAT|3|
|2|5|SVCHOST|11|

Modalità di consegna degli esercizi assegnati e della relazione:

- Per sostenere l'esame è necessario consegnare preventivamente il codice e una relazione (in pdf) sugli esercizi assegnati
    - Ogni gruppo di lavoro dovrà consegnare un'unica relazione
    - Il codice consegnato deve essere funzionante sul simulatore QtSpim, anche se sviluppato e testato con altri simulatori (es, MARS)
- Un archivio contenente il codice e la relazione dovrà essere caricato sul sito moodle del corso seguendo l'apposito link che verrà reso disponibile alla pagina del corso
    - Non vengono prese in considerazione altre modalità di consegna
- La scadenza esatta della consegna verrà resa nota di volta in volta
- Discussione e valutazione: la discussione degli elaborati avverrà contestualmente all'esame orale e prevede anche domande sull'assembly e su tutti gli argomenti di laboratorio trattati a lezione.

Struttura della relazione (in formato pdf):

- Info su autori e data di consegna
    - gli autori (e loro indirizzo e-mail – preferibilmente quello universitario@stud.unifi.it)
    - la data di consegna
- Per ciascun esercizio
    - Descrizione della soluzione adottata, trattando principalmente i seguenti punti:
        - Descrizione ad alto livello dell'algoritmo (in linguaggio naturale, con flow-chart, in pseudo-linguaggio, etc.), delle strutture dati utilizzate (liste, vettori, etc.) e delle procedure utilizzate (argomenti, funzionalità svolte, risultati prodotti)
        - Uso dei registri e memoria (stack, piuttosto che memoria statica o dinamica)
        - Motivazione delle scelte implementative
    - Simulazione
        - In questa sezione andranno gli screenshot commentati di una o più simulazioni-tipo, anche discutendo l'evoluzione del contenuto del "user data segment" e dello "user stack" durante l'esecuzione del codice. Mostrare anche il funzionamento dell'algoritmo in corrispondenza di input sbagliati (es, inserisco da tastiera un carattere al posto di un numero).
    - Codice MIPS assembly implementato e commentato in modo chiaro ed esauriente. (nota: nella relazione va inserito TUTTO il codice)

Codice

- Il codice deve essere suddiviso in cartelle (una cartella per ogni esercizio) ed i nomi dei files devono permettere di identificare facilmente l'esercizio a cui si riferiscono. Unico file per esercizio.
- Ricordarsi di inserire anche nel codice gli autori (e loro indirizzo e-mail) e la data di consegna.
- Seguire fedelmente le convenzioni sull'uso della memoria e sulle procedure.
- Commentare il codice in modo significativo (move $s4,$s3 non significa solo che "copio il contenuto del registro $s3 in $s4"....).