Elaborato ASM – Laboratorio Architettura degli Elaboratori

UniVR - Dipartimento di Informatica

Software pianificatore

Mattia Arganetto - VR502412
Fabio Irimie - VR501504

2° Semestre 2023/2024

Indice

1 Struttura del codice

1.1 Funzioni ausiliarie

1.1.1 printStr.s

La funzione presente in questo file serve a stampare una qualsiasi stringa che finisca con il carattere '\0' presente nel registro $\%ecx$.

1.1.2 writeStrFile.s

La funzione presente in questo file serve a scrivere una stringa in un file. I parametri da passare alla funzione sono:

- $\%ebx$: File descriptor del file aperto
- $\%ecx$: Stringa da scrivere

1.1.3 printInt.s

La funzione presente in questo file serve a convertire un intero presente nel registro $\%eax$ in una stringa e successivamente stamparla.

1.1.4 writeIntFile.s

La funzione presente in questo file serve a scrivere un intero in un file. I parametri da passare alla funzione sono:

- $\%esi$: File descriptor del file aperto
- $\%eax$: Intero da scrivere

1.1.5 atoi.s

La funzione presente in questo file serve a convertire una stringa presente nel registro $\%esi$ in un intero e restituirlo nel registro $\%eax$.

1.1.6 bubbleSort.s

La funzione presente in questo file serve a riordinare una lista di interi presente nello stack in base all'algoritmo scelto dall'utente:

- Se viene scelto EDF (Earliest Deadline First) la lista di prodotti viene riordinata in base alla scadenza crescente, oppure nel caso in cui le scadenze siano uguali, i prodotti vengono riordinati in base alla priorità decrescente.
- Se viene scelto HPF (Highest Priority First) la lista di prodotti viene riordinata in base alla priorità decrescente, oppure nel caso in cui le priorità siano uguali, i prodotti vengono riordinati in base alla scadenza crescente.

I parametri da passare alla funzione sono i seguenti:

- $\%ecx$: Offset dal primo elemento di un prodotto per indicare che valori controllare per l'ordinamento:
  - 1: Identificativo

  - 2: Durata
  - 3: Scadenza
  - 4: Priorità

- %*esi*: Offset dal primo elemento di un prodotto per indicare che valori controllare nel caso in cui i valori di %*ecx* siano uguali:
  - 1: Identificativo
  - 2: Durata
  - 3: Scadenza
  - 4: Priorità

- %*edx*: Numero di valori presenti nello stack (numero di prodotti * 4)
- %*edi*: Numero di prodotti

La funzione fa affidamento ad un'altra funzione presente nel file **swapOrders.s** (1.1.7).

1.1.7 swapOrders.s

Questa funzione serve a scambiare due prodotti presenti nello stack e viene utilizzata dall'algoritmo di ordinamento. I parametri da passare alla funzione sono:

- %*eax*: Indice del primo valore del primo prodotto
- %*ebx*: Indice del primo valore del secondo prodotto
- %*ebp*: Indice del primo valore del primo prodotto nello stack

1.2 File principali

1.2.1 main.s

Questo è il file principale del programma, cioè il primo che viene eseguito. All'interno di questo file vengono ottenuti i parametri passati dalla riga di comando e successivamente si procede con l'apertura del file passato come parametro. Se l'apertura non va a buon fine viene stampato un errore e il programma viene terminato chiudendo così il file. Abbiamo deciso di aprire il file una sola volta e di tenerlo aperto per tutta la durata del programma, chiudendolo soltanto alla fine dell'esecuzione. Questo per evitare di aprire e chiudere il file più volte.

1.2.2 menu.s

In questo file è presente il loop che stampa il menu ogni volta che viene completata una pianificazione o quando il programma viene eseguito per la prima volta. L'esecuzione del menu presuppone un file descriptor nel registro %*eax* in modo da poterlo passare poi agli altri file che necessitano di un file descriptor per leggere dal file. Ogni volta che viene stampato il menu viene chiesto all'utente di inserire un'opzione, se l'opzione è valida viene eseguita la funzione corrispondente, altrimenti viene stampato un messaggio di errore e l'input viene chiesto nuovamente. Le opzioni disponibili sono:

- 1: Pianifica gli ordini con l'algoritmo EDF (Earliest Deadline First)
- 2: Pianifica gli ordini con l'algoritmo HPF (Highest Priority First)
- 3: Esci dal programma

1.2.3 plan.s

In questo file avviene la pianificazione degli ordini. I parametri per questo file sono da passare nello stack e sono i seguenti:

- File descriptor del file aperto
- Algoritmo da eseguire inserito in input dall'utente

Una volta ottenuti i parametri si procede con la lettura del file, però siccome il file rimane sempre aperto per tutta l'esecuzione del programma è necessario posizionare il puntatore del file all'inizio del file ogni volta che si vuole fare una nuova pianifica; questo viene fatto con la system call **lseek**:

```
mov $19, %eax  ; syscall lseek (riposiziona il read/write offset)
mov fd, %ebx   ; File descriptor
mov $0, %ecx   ; Valore di offset
mov $0, %edx   ; Posizione di riferimento (0 = inizio del file)
int $0x80
```

La lettura del file viene effettuata leggendo un carattere alla volta inserendolo in un buffer. Ogni volta che viene rilevato un carattere '$\backslash n$' si aumenta il contatore dei prodotti.
Leggendo il file si ottengono tutti i valori dei prodotti e vengono inseriti nello stack. Questo viene effettuato concatenando ogni intero letto dal file all'interno di un buffer che verrà poi convertito in un intero quando viene letto una virgola o un'andata a capo.
Una volta che tutti i prodotti sono stati inseriti nello stack si procede con l'ordinamento della lista dei prodotti in base all'algoritmo scelto dall'utente, cioè inserendo i parametri corretti per la funzione **bubbleSort.s** come definito sopra (1.1.6).
Quando la lista è ordinata non resta che stampare in sequenza i prodotti calcolando i tempi di inizio e la penalità per ogni prodotto. Questo viene fatto nel file **output.s** (1.2.4).
Dopo aver completato la pianifica vengono tolti dallo stack tutti i valori inseriti.

1.2.4 output.s

Questo file serve a stampare i prodotti in sequenza (presupponendo che la lista sia già ordinata) e a calcolare i tempi di inizio e la penalità per ogni prodotto. I parametri da passare a questa funzione sono:

- $\%eax$: Algoritmo scelto dall'utente
- $\%ebx$: Numero di valori presenti nello stack (numero di prodotti * 4)
- $\%ecx$: Numero di prodotti

Questa funzione dopo aver salvato i parametri procede a calcolare il tempo di inizio di ogni prodotto scorrendo la lista e sommando la durata dei prodotti precedenti. Ogni tempo di inizio viene quindi inserito nello stack.
Successivamente viene stampato l'output secondo la specifica richiesta e infine viene calcolata la penalità totale sommando le penalità di ogni prodotto e stampandola. Una volta completata la stampa dell'output vengono tolti tutti i tempi di inizio dallo stack.

2 Simulazione ordini

2.1 Controllo dei valori

All'interno di ogni file di testo viene eseguito (in lettura) un controllo per verificare la validità dei valori presenti all'interno. Se essi non rispettano le regole imposte il programma resituisce un errore e viene interrotto.
In caso contrario il programma continua la sua esecuzione senza intoppi.

2.2 Ordini obbligatori

I seguenti ordini sono stati fatti per testare la correttezza del programma in situazioni in cui la lista di ordini è corretta.

2.2.1 EDF.txt

Il seguente file di testo contiene una lista di ordini che devono essere organizzati in modo tale da garantire una penalità uguale a `0` con l'algoritmo di ordinamento EDF e una penalità maggiore di `0` con l'algoritmo di ordinamento HPF.

2.2.2 Both.txt

Il seguente file di testo contiene la lista di ordini che devono essere organizzati in modo da garantire una penalità maggiore di `0` per entrambi gli algoritmi di ordinamento presenti nel programma. In particolare questa lista di ordini prevede delle scadenza più corte del singolo tempo di produzione dell'ordine in modo da riportare una penalità elevata in entrambi gli algoritmi.

2.2.3 None.txt

Il seguente file di testo contiene la lista di ordini che dovranno essere organizzati in modo da garantire una penalità uguale a `0` per entrambi gli algoritmi di ordinamento. In questo caso le scadenze sono state impostate ad un tempo elevato in modo da favorire il completamento degli ordini per entrambi gli algoritmi di ordinamento.

2.3 Ordini extra

I seguenti ordini sono stati fatti per testare la correttezza del programma in situazioni particolari.

2.3.1 noInput.txt

Il seguente file di testo è vuoto e non contiene alcun ordine. Il programma restituisce un errore e termina l'esecuzione.

2.3.2 oneLine.txt

Il seguente file di testo contiene un solo ordine. Il programma viene eseguito come previsto e restituisce un output corretto.

2.3.3 wrongInput.txt

Il seguente file di testo contiene una stringa al posto dei valori degli ordini. Il programma restituisce un errore e termina l'esecuzione.

2.3.4 wrongId.txt

Il seguente file di testo contiene un identificativo fuori dall'intervallo consentito. Il programma restituisce un errore e termina l'esecuzione.

2.3.5 wrongDuration.txt

Il seguente file di testo contiene una durata fuori dall'intervallo consentito. Il programma restituisce un errore e termina l'esecuzione.

2.3.6 wrongDeadline.txt

Il seguente file di testo contiene una scadenza fuori dall'intervallo consentito. Il programma restituisce un errore e termina l'esecuzione.

2.3.7 wrongPriority.txt

Il seguente file di testo contiene una priorità fuori dall'intervallo consentito. Il programma restituisce un errore e termina l'esecuzione.

3 Scelte progettuali

3.1 Apertura del file

Abbiamo deciso di aprire il file una sola volta e di tenerlo aperto per tutta la durata del programma, chiudendolo soltanto alla fine dell'esecuzione. Questa decisione è stata presa per risparmiare tempo e risorse, evitando di aprire e chiudere il file più volte.

3.2 Lettura del file

Abbiamo deciso di leggere il file ogni volta che viene richiesta una nuova pianificazione in modo da permettere all'utente di modificare il file di input senza dover riavviare il programma. Siccome il file è aperto solo con permessi di lettura si può modificare il file di input tranquillamente senza errori.