ELABORATO ASSEMBLY 2023/2024

ARCHITETTURA DEGLI ELABORATORI

VR500307: FRANCESCO FRISON

VR500646: TONI EMILOV PETROV

DESCRIZIONE DELL'ELABORATO

Si richiede lo sviluppo di un software in linguaggio Assembly (sintassi AT&T) che sia in grado di gestire la pianificazione di un sistema produttivo. Per un massimo di 10 prodotti in un massimo di 100 unità di tempo, il sistema decide l'ordine di produzione.

Per ogni prodotto completato in ritardo verrà attribuita una penalità pari al ritardo moltiplicato per la sua priorità.

Il software fornisce due possibili algoritmi di pianificazione: EDF, HPF. Il primo si basa sulla scadenza minore, il secondo sulla priorità maggiore.

STRUTTURA DEL CODICE

File in cui è organizzato (contenuti all'interno della cartella 'src'):

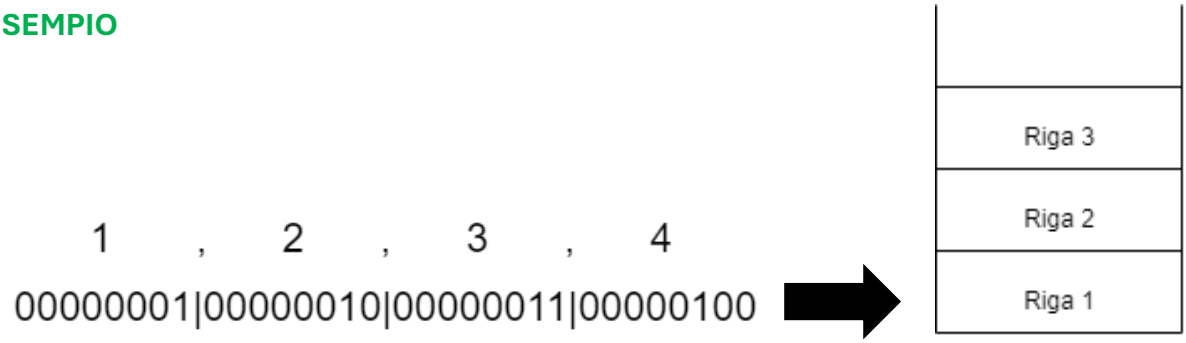
- **main.s** : La parte del programma principale;
- **edf.s** : Algoritmo di ordinamento EDF;
- **hpf.s** : Algoritmo di ordinamento HPF;
- **printfd.s** : Stampa il valore contenuto nel registro EAX;
- **scanfd.s** : Legge un valore da tastiera e lo inserisce in EAX;
- **readl.s** : Interpreta una riga dal file di testo da leggere;
- **printvideo.s** : Si occupa di creare la stringa contenente l'output dell'algoritmo;
- **convert.s** : Converte la riga del file interpretata in 32 bit;
- **revert.s** : Dati 32 bit, fornisce uno dei 4 valori della riga.

GESTIONE DELLA MEMORIA

Ogni singola riga del file è contenuta in una riga nello stack.

Tale riga viene convertita in un valore a 32 bit che rappresenta le 4 caratteristiche di quell'ordine; ciò è possibile grazie al fatto che ogni caratteristica è rappresentabile da 8 bit in memoria (8x4 = 32 bit totali).

ESEMPIO



Solo dopo la conclusione di un algoritmo, lo stack viene liberato dalle variabili che erano state inserite.

LETTURA FILE

Dopo aver aperto il file, esso viene letto carattere per carattere. Ogni carattere viene scritto in uno spazio di memoria chiamato *riga* e utilizzando un puntatore chiamato *index*.

```
next: # SCRITTURA DA FILE A STRINGA
    # scrivo il carattere (attualmente in AL) letto in "riga" alla posizione corrente
    movl index, %ebx
    addl $riga, %ebx # riga + offset indice ( punto al carattere prossimo )
    movb %al, (%ebx) # scrivo nella posizione del carattere prossimo

    # incremento l'indice
    addl $1, index

    jmp read_loop # continuo a leggere
```

Quando viene letto il carattere '/n' viene scritto in *riga* e incrementiamo la variabile *lines*, per tenere conto del numero di ordini presenti nel file. Successivamente, *riga* contiene l'intera riga del file.

Tale riga viene passata alla funzione *convert* per trasformarla in 32 bit, e viene caricata nello stack. Infine, *riga* viene resettata da NULL per continuare la lettura.

```
reset_riga: # PULIZIA STRINGA
    movl $20, %ecx
    movl $riga, %ebx
resetstring_loop:
    movb $0, (%ebx)
    incl %ebx
    loop resetstring_loop
```

ALGORITMO EDF

Utilizza come parametri ESP e *lines*. Grazie ad essi si scende di stack fino alla prima riga. Viene impostato EBP all'ESP passato. Successivamente, si esegue questo calcolo per posizionarsi correttamente:

EBP + 4 * (n. Righe – 1)

```
# imposto nuovo basePointer
movl %eax, %ebp
# INIZIALIZZO IL CONTATORE
movl edf_lines, %eax
movl %eax, edf_count
# NB: edf_lines VALE 1 IN PIU' DI QUANTO DEVO DECREMENTARE LO STACK
movl $4, %edx # grandezza di una riga dello stack
decl %eax # quindi decremento eax (ha valore edf_lines) di 1
mull %edx # es: se ho 3 linee, e' come se facessi $4 * (3-1) = 8 --> 8(%ebp)
addl %eax, %ebp # adesso ebp punta alla prima riga che avevo messo sullo stack
# SALVO EBP CHE PUNTA ALLA PRIMA VARIABILE IN FONDO ALLO STACK
movl %ebp, edf_basePointer
```

Infine, viene salvato il puntatore *edf_basePointer* alla prima riga di stack, in modo da poterlo reimpostare in caso di necessità.

Riguardo l'ordinamento in sé, si utilizzano le variabili *min* e *max2*.

Viene eseguito un primo ciclo in cui si determina la scadenza minore, contenuta in *min*. Successivamente, si conta quante volte si presenta *min*; se solo una volta, viene salvato l'indirizzo e la riga a 32 bit.

Se esiste più di una riga con lo stesso *min*, viene eseguito un secondo ciclo in cui si calcola il *max2* delle priorità solo delle righe con il *min* della scadenza. E come prima, si salva l'indirizzo e la riga.

```
# in eax ho la scadenza
cmpl min, %eax
jne skip_check_min_EDF
# ho trovato una riga che ha il minimo attuale
incw edf_countmin # aumento il contatore
movl (%ebp), %eax
movl %eax, edf_riga_da_stampare  # salva il num a 32bit
movl edf_stmp, %eax # salvo il num di variabile
movl %eax, edf_toDelete
```

La funzione quindi, prima di ritornare la riga a 32 bit scelta da stampare, azzera tale indirizzo di stack, in modo da poterla ignorare alla prossima chiamata (tramite un controllo sui cicli). `movl $0, (%ebp) # azzero tale area di memoria`

Rispetto l'algoritmo HPF, invece, esso rispecchia questo ma invertendo cosa controllare prima e cosa dopo.

CALCOLO PENALITÀ

Per calcolare la penalità di ogni ordine stampato, viene sottratta la scadenza dagli slot temporali. Se questa sottrazione produce un numero positivo, significa che l'ordine è stato completato in ritardo e quindi tale valore rispecchia l'intervallo di tempo in eccesso. Moltiplicandolo per la priorità dell'ordine, si ottiene la penalità che l'azienda deve pagare.

```
subl %eax, %edx # slot temporali - scadenza -> risultato in eax
cmpl $0, %edx
# se il risultato e' positivo quelli sono i giorni di ritardo passati
jle skip_penalty # se non sono in ritardo, salto calcolo penalita'
                 # se eax e' minore o uguale di zero salta
# quindi li moltiplico per la priorita'
movl riga_da_printare, %eax
movl $4, %ebx # prendo priorita'
call revert
mulb %dl # moltiplico il tempo di ritardo per la penalita'
# priorita' * edx (giorni di ritardo) -> risultato in eax
```

SCELTE PROGETTUALI

- Si presuppone che il file degli ordini fornito, se non vuoto, sia scritto correttamente e con tutti i parametri entro i propri valori;
- Nel caso gli ordini siano più di 10, l'algoritmo stampa un warning che avvisa di aver superato tale limite, ma procede comunque correttamente;
- Sono gestiti vari errori, tra cui: quantità di parametri, apertura file e file da leggere vuoto;
- Durante la scelta dell'algoritmo, inserire '0' fa terminare il programma, mentre inserire un valore diverso da ['0', '1', '2'] fa ristampare il menù;
- Se il file (bonus) da scrivere fornito non esiste, esso viene creato.