Relazione elaborato Assembly

Laboratorio di Architettura degli Elaboratori

A.A. 2023/2024

Martini Anna VR504166
Zanotelli Riccardo VR474635

Consegna dell’elaborato

Si sviluppi in Assembly (sintassi At&t) un software per la pianificazione delle attività di un sistema produttivo, per i successivi prodotti, fino ad un massimo di 10 prodotti, nelle successive 100 unità di tempo dette “slot temporali”.
Il sistema produttivo può produrre prodotti diversi, ma produce un prodotto alla volta. La produzione è suddivisa in slot temporali uniformi, e durante ogni slot temporale solo un prodotto può essere in produzione. Ogni prodotto è caratterizzato da quattro valori interi:

- Identificativo: il codice identificativo del prodotto da produrre. Il codice può andare da 1 a 127
- Durata: il numero di slot temporali necessari per completare il prodotto. La produzione di ogni prodotto può richiedere 1 a 10 slot temporali
- Scadenza: il tempo massimo, espresso come numero di unità di tempo entro cui il prodotto dovrà essere completato. La scadenza di ciascun prodotto può avere un valore che va da 1 a 100
- Priorità: un valore da 1 a 5, dove 1 indica la priorità minima e 5 la priorità massima. Il valore di priorità indica anche la penalità che l’azienda dovrà pagare per ogni unità di tempo necessaria a completare il prodotto oltre la scadenza.

Per ogni prodotto completato in ritardo rispetto alla scadenza indicata, l’azienda dovrà pagare una penale in Euro pari al valore della priorità del prodotto completato in ritardo, moltiplicato per il numero di unità di tempo di ritardo rispetto alla sua scadenza.
Il software dovrà essere eseguito mediante la seguente linea di comando: pianificatore <percorso del file degli ordini>.
Il file degli ordini dovrà avere un prodotto per riga, con tutti i parametri separati da virgola. Ogni file non può contenere più di 10 ordini. Una volta letto il file, il programma mostrerà il menù principale che chiede all’utente quale algoritmo di pianificazione dovrà usare. L’utente potrà scegliere tra i seguenti due algoritmi di pianificazione:
1. Earliest Deadline First (EDF): si pianificano per primi i prodotti la cui scadenza è più vicina, in caso di parità nella scadenza, si pianifica il prodotto con la priorità più alta.
2. Highest Priority First (HPF): si pianificano per primi i prodotti con priorità più alta, in caso di parità di priorità, si pianifica il prodotto con la scadenza più vicina.

L’utente dovrà inserire il valore 1 per chiedere al software di utilizzare l’algoritmo EDF, ed il valore 2 per chiedere al software di utilizzare l’algoritmo HPF. Una volta pianificati i task, il software dovrà stampare a video:
1. L’ordine dei prodotti, specificando per ciascun prodotto l’unità di tempo in cui è pianificato l’inizio della produzione del prodotto. Per ogni prodotto, dovrà essere stampata una riga con la seguente sintassi: ID:Inizio dove ID è l’identificativo del prodotto, ed Inizio è l’unità di tempo in cui inizia la produzione.
2. L’unità di tempo in cui è prevista la conclusione della produzione dell’ultimo prodotto pianificato.
3. La somma di tutte le penalità dovute a ritardi di produzione.

Una volta stampate a video le statistiche, il programma tornerà al menù iniziale in cui chiede all’utente se vuole pianificare la produzione utilizzando uno dei due algoritmi. L’uscita dal programma potrà essere gestita in due modi: si può scegliere di inserire una voce apposita (esci) nel menu principale, oppure affidarsi alla combinazione di tasti ctrl-C. In entrambi i casi però, tutti i file utilizzati dovranno risultare chiusi al termine del programma.

Struttura del software

Il nostro software è strutturato nei seguenti file:

- **algoritmo_edf.s** > implementa l'algoritmo Earliest Deadline First
- **algoritmo_hpf.s** > implementa l'algoritmo Highest Priority First
- **itoa.s** > funzione per la conversione di numeri interi in stringhe ASCII
- **main.s** > gestisce l'input dell'utente e chiama le funzioni
- **programmazione.s** > programma le attività e il calcolo della penalità
- **read_file.s** > legge il file di input

Main

Il "main" è il primo programma che viene avviato. Esso infatti chiama tutti gli altri sottoprogrammi ed in esso è implementata la richiesta all'utente di quale algoritmo di ordinamento desidera fare uso. Infatti dopo la comparsa della stringa *"Seleziona l'algoritmo da utilizzare (EDF: 1, HPF: 2, esci: q): "*, l'utente può inserire, come da richiesta, 1 per l'algoritmo EDF, 2 per l'algoritmo HPF o il carattere q per uscire (quit). Il carattere inserito dall'utente da tastiera viene confrontato con i caratteri predefiniti nella sezione del programma .section .data: qualora essi corrispondano viene chiamata la funzione che onora la richiesta dell'utente, qualora invece i comandi non corrispondano compare un messaggio d'errore (*"Selezione non valida"*). Inoltre, qualora i parametri del file non siano adeguati, il main stampa un ulteriore messaggio di errore (*"Ops! Qualcosa non ha funzionato"*).

Read file

La funzione "read file" si occupa di leggere e verificare i valori presenti nel file. Tramite la system call "read" viene letto un carattere alla volta e posto in un buffer. Tramite una serie di confronti, si determina se si è giunti alla fine del file, se c'è un errore nella lettura, se il carattere corrisponde a una virgola oppure se si va a capo e quindi si passa ad un nuovo prodotto. Qualora infatti il carattere letto sia una virgola o un a capo, il valore è completo e va analizzato, qualora invece il carattere sia una cifra, si procede con la sua conversione da ASCII ad intero e lo si pone in *value*. Nel momento in cui il valore è completo tramite un'ulteriore serie di confronti si determina se sia coerente con le categorie di ID, durata, scadenza e priorità e se questo check da esito positivo si incrementa il numero di elementi, qualora l'esito sia negativo produce un messaggio di errore ("*Ops! Qualcosa non ha funzionato nella lettura del file*")

Algoritmi EDF e HPF

Gli algoritmi di ordinamento sono contenuti in due file separati, le cui funzioni vengono chiamate nel main subito dopo la richiesta all'utente. Entrambi gli algoritmi fanno uso di due cicli *for* l'uno innestato nell'altro che richiamano un *bubble sort* per ordinare i prodotti in base al criterio desiderato. Per implementare gli algoritmi facciamo uso di due puntatori, ispirandosi alla struttura dell'array, per fare due accessi alla memoria distinti. Infatti in entrambi gli algoritmi esistono delle variabili chiamate *[variabile]_attuale* e *[variabile]_prossima* per confrontare due valori e ordinarli. In entrambi gli algoritmi viene calcolato il numero di prodotti (numero_righe) dividendo il numero di byte totali nella pila per

4. Subito dopo vengono inizializzati gli indici per i cicli: l'indice per lo scorrimento a -1, l'indice massimo al numero di righe.

Algoritmo EDF (Earliest Deadline First)

Nel primo ciclo *for* vengono settate le scadenze, attuale e prossima, affinché possano essere confrontate per permettere l'ordinamento. La scadenza attuale e la scadenza prossima vengono inizializzate rispettivamente a 4 e a 20 per "saltare" in maniera corretta nella pila. All'interno del secondo ciclo *for* avviene l'effettivo confronto tra la scadenza del prodotto attuale e quella del prossimo. Si prendono i valori di entrambe le scadenze dalla pila, andando alla cella di memoria puntata dalla somma tra il registro *esp* e il registro corrispondente alla scadenza (*eax* per l'attuale, *ebx* per la prossima). Confrontandoli, qualora la scadenza attuale sia minore della prossima si vanno a scambiare i valori, qualora siano uguali si passa a controllare l'ordine di priorità. Il confronto delle priorità avviene in maniera simile. A partire dalla scadenza attuale e prossima si ottengono le rispettive priorità sottraendo la costante 4. A seguito di un confronto, qualora la priorità attuale sia maggiore della prossima si procede a scambiare i valori. Per lo scambio dei valori, si parte dalle scadenze e da esse si ottiene la priorità, sottraendo 4; dopo lo scambio delle priorità, sommando progressivamente la costante 4, si scambiano scadenze, durate ed ID.

Algoritmo HPF (Highest Priority First)

Nel primo ciclo *for* vengono settate le priorità, sempre attuale e prossima, affinché possano essere confrontate per permettere l'ordinamento. La priorità attuale e la priorità prossima vengono inizializzate rispettivamente a 0 e a 16 per "saltare" in maniera corretta nella pila. All'interno del secondo ciclo *for* avviene l'effettivo confronto tra la priorità del prodotto attuale e quella del prossimo. Si prendono i valori di entrambe le priorità dalla pila, come già visto in precedenza, andando alla cella di memoria puntata dalla somma tra il registro *esp* e il registro corrispondente alla priorità (*eax* per l'attuale, *ebx* per la prossima). Confrontandoli, qualora la priorità attuale sia maggiore della prossima si vanno a scambiare i valori, qualora siano uguali si passa a controllare l'ordine di scadenza. Il confronto delle scadenze avviene in maniera simile. A partire dalla priorità attuale e prossima si ottengono le rispettive priorità sommando la costante 4. A seguito di un confronto, qualora la scadenza attuale sia minore della prossima si procede a scambiare i valori. Per lo scambio dei valori, si parte dalle priorità e dopo averle scambiate, sommando progressivamente la costante 4 si scambiano scadenze, durate ed ID.

Programmazione

Questa sezione del software si occupa di aggiornare la durata totale, di calcolare la penalità complessiva e stampare a video i relativi messaggi insieme all'ID. Facendo uso di un ciclo *for* che termina quando l'indice di scorrimento raggiunge il numero massimo di prodotti diminuito di 1, scorrendo "l'array" precedentemente ordinato, si va prima a rintracciare nella pila il codice identificativo e grazie alla chiamata della funzione *itoa* esso viene stampato a video. In seguito, dopo aver stampato con una system call il carattere ":", si va a prendere nella pila l'unità di tempo in cui inizia la produzione di un prodotto (*durata_totale*) e si chiama

la funzione *itoa* per stamparla a video. Viene quindi incrementata la durata totale sommando la durata del singolo prodotto corrente e si procede con il calcolo della penalità. Per calcolare la penalità, in primo luogo si sottrae alla durata totale la scadenza e qualora questo valore sia negativo, ossia ci sia una effettiva penalità, si procede a moltiplicare la priorità per il ritardo, calcolato prima con la sottrazione e che risiede in *ebx*. Viene perciò aggiornata la variabile *penalita_totale* sommando ad essa il risultato appena trovato. In ogni caso, viene aggiornato l'indice di scorrimento e l'ID.
Alla fine del ciclo si stampano le statistiche totali. Grazie a due system call si stampano le intestazioni "*Conclusione:*" e "*Penalty:*", dichiarate a priori nella sezione .section .data e grazie alle chiamate della funzione *itoa* si stampano la durata totale e la penalità totale.

Itoa

La funzione Itoa (Integer To Ascii) definisce l'algoritmo per convertire un tipo intero in un tipo ASCII, per poi stamparlo a video.
Qualora il valore contenuto in *eax* sia maggiore di 10, poiché il sistema in uso è quello decimale, si divide ricorsivamente per 10 il contenuto di *eax*, il resto della divisione viene caricato in pila e viene aggiornato il contatore delle cifre nello stack (*ecx*). Nel momento in cui il contenuto di *eax* risulta essere minore di 10, ossia non è più necessario dividere, salva nello stack il contenuto di *eax*, ossia l'ultima cifra, incrementa per l'ultima volta il contatore, lo sposta in *ebx* e va alla stampa. Per reiterare il ciclo di stampa bisogna, ad ogni carattere stampato, decrementare il contatore, fino a che non sarà uguale a 0. Agli 8 bit meno significativi di *eax* viene sommata la costante 48 che converte l'intero in ASCII e si procede alla stampa tramite la system call.

Compilazione e run

La compilazione viene attuata attraverso Makefile, presente nel progetto, che compila tutti i file sorgente in file oggetto e successivamente crea un link tra di loro, per creare l'eseguibile "pianificatore". Per il lancio del programma si utilizza il comando ./bin/pianificatore <nome del file>

Test

Per testare il software abbiamo fatto uso di tre file distinti:

- un file chiamato *EDF.txt* che produce penalità uguale a zero con EDF e maggiore di zero con HPF
- un file chiamato *Both.txt* che produce penalità uguale a zero con entrambi gli algoritmi
- un file chiamato *None.txt* che produce penalità maggiore di zero con entrambi gli algoritmi

Esecuzione: file EDF.txt

EDF.txt

```
4,4,30,4
2,2,20,2
16,6,40,1
1,1,15,1
3,3,25,3
8,8,50,3
15,5,35,5
19,9,55,4
110,10,60,5
17,7,45,2
```

Pianificazione EDF:

```
1:0
2:1
3:3
4:6
15:10
16:15
17:21
8:28
19:36
110:45
Conclusione: 55
Penalty: 0
```

Pianificazione HPF:

```
15:0
110:5
4:15
19:19
3:28
8:31
2:39
17:41
1:48
16:49
Conclusione: 55
Penalty: 115
```

Esecuzione: file Both.txt

Both.txt

```
5,5,50,5
4,4,40,4
3,3,30,3
2,2,32,2
1,1,57,1
7,7,70,4
6,6,60,5
```

Pianificazione EDF:

```
3:0
2:3
4:5
5:9
1:14
6:15
7:21
Conclusione: 28
Penalty: 0
```

Pianificazione HPF:

```
5:0
6:5
4:11
7:15
3:22
2:25
1:27
Conclusione: 28
Penalty: 0
```

Esecuzione: file None.txt

None.txt

```
1,1,5,1
5,5,25,5
7,7,35,2
6,6,30,2
9,9,45,4
10,10,50,5
8,8,40,3
2,2,10,2
3,3,15,3
4,4,20,4
```

Pianificazione EDF:

```
1:0
2:1
3:3
4:6
5:10
6:15
7:21
8:28
9:36
10:45
Conclusione: 55
Penalty: 25
```

Pianificazione HPF:

```
5:0
10:5
4:15
9:19
3:28
8:31
2:39
6:41
7:47
1:54
Conclusione: 55
Penalty: 232
```

Scelte progettuali

La prima scelta progettuale è stata l'utilizzo della voce "esci" nel menù, che è possibile selezionare ponendo in input da tastiera la "q" (*quit*).
Negli algoritmi abbiamo inoltre assunto che ogni "categoria" che identifica un prodotto sia ognuna grande 4 byte.