Laboratorio Architettura degli Elaboratori
Elaborato ASM

Università di Verona
Imbriani Paolo - Baldo Chiara
VR500437 - VR500878

A.A. 2023/2024

Contents

1 Introduzione e specifiche del progetto

Si sviluppi in Assembly (sintassi AT&T) un software per la pianificazione delle attività di un sistema produttivo, per i successivi 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. Ad esempio, se il prodotto:

Identificativo: 4; Durata: 10; Scadenza: 25; Priorità: 4;

venisse messo in produzione all'unità di tempo 21, il sistema completerebbe la sua produzione al tempo 30, con 5 unità di tempo di ritardo rispetto alla scadenza richiesta, l'azienda dovrebbe pagare una penalità di $5 * 4 = 20$ Euro.

Il software dovrà essere eseguito mediante la seguente linea di comando:

`pianificatore <percorso del file degli ordini>`

Ad esempio, se il comando dato fosse:

`pianificatore Ordini.txt`

il software caricherà gli ordini dal file Ordini.txt.

Il file degli ordini dovrà avere un prodotto per riga, con tutti i parametri separati da virgola. Ad esempio, se gli ordini fossero:

Identificativo: 4; Durata: 10; Scadenza: 12; Priorità: 4;
Identificativo: 12; Durata: 17; Scadenza: 32; Priorità: 5;

Il file dovrebbe contenere le seguenti righe:

```
4,10,12,4
12,7,32,1
```

Ogni file non può contenere più di 10 ordini. Una volta letto il file, il programma mostrerà il menu 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.

Dunque, nell'esempio precedente, se si utilizzasse l'algoritmo EDF, l'output atteso sarà:

```
Pianificazione EDF:
4:0
12:10
Conclusione: 17
Penalty: 0
```

Mentre, se si fosse usato l'algoritmo HPF:

```
Pianificazione HPF:
12:0
4:17
Conclusione: 17
Penalty: 20
```

In quanto nel primo caso non ci sarebbero penalità, mentre nel secondo caso il prodotto con ID 4 terminerebbe con 5 unità di tempo di ritardo, da moltiplicare per il valore di priorità 4.

L'output del programma dovrà avere la sintassi riportata sopra. 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.

Bonus (Facoltativo): se l'utente inserisce due file come parametri da linea di comando, il file specificato come secondo parametro verrà utilizzato per salvare i risultati della pianificazione, indicando l'algoritmo usato. Ad esempio:

```
pianificatore Ordini.txt Pianificazione.txt
```

Il programma carica gli ordini dal file Ordini.txt e salva le statistiche stampate a video nel file Pianificazione.txt. Nel caso l'utente inserisca un solo parametro, la stampa su file sarà ignorata.

2 Progettazione dell'elaborato

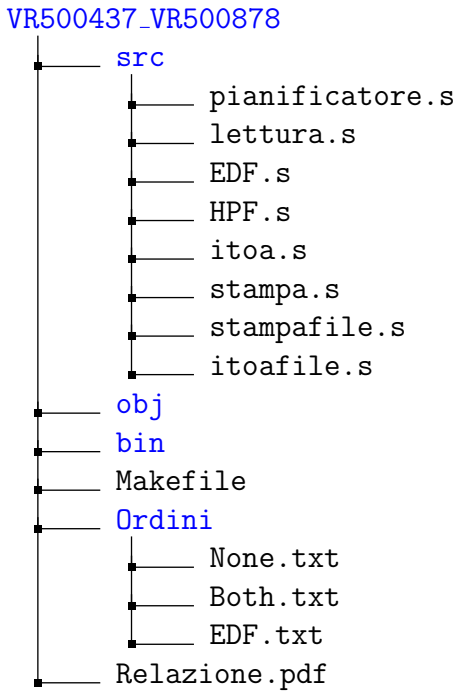
2.1 Organizzazione file

Il global start del progetto si trova in un file *"pianificatore.s"*; alle sue dipendenze ci sono altre funzioni, linkate tramite Makefile, che eseguono operazioni quali la lettura, il calcolo e la stampa.

Pianificatore.s
I compiti principali del "main" sono ricevere da linea di comando i parametri che saranno necessari all'esecuzione del codice (eseguibile e file.txt da cui recuperare i dati), chiamare la lettura e, una volta terminata questa, mandare al loop del Menù. Esso chiede all'utente di inserire l'algoritmo e, una volta ricevuta l'istruzione, chiama le funzioni **EDF** (se è stato inserito 0) o **HPF** (se 1) e subito dopo la funzione di stampa, comune ad entrambi i percorsi, prima di tornare alla richiesta iniziale.
Per terminare è sufficiente scrivere "q" alla richiesta di inserimento algoritmo oppure premere *Ctrl-C* in qualunque momento. Se il programma viene chiuso "correttamente", ovvero inserendo q, il pianificatore procede anche a chiudere i file, svuotare lo stack (qualora sia stato riempito durante l'esecuzione) e mandare l'interrupt per la systemcall exit.
Per quanto concerne la parte facoltativa, il pianificatore ne cura alcuni aspetti: controlla se sono stati inseriti esattamente tre parametri e, se sì, ne controlla la correttezza e eventualmente crea il file su cui stamperà i risultati del programma; a fine esecuzione lo chiuderà anche.
Il pianificatore gestisce inoltre alcuni degli errori principali: l'inserimento di

parametri non validi su linea di comando (o di un numero errato di pramentri) e l'inserimento di input diversi da "1", "2" o "q".

Organizzazione delle cartelle e dei file:



2.2 Funzioni e algoritmi implementati

Le funzioni che abbiamo inserito all'interno del programma sono

- EDF
- HPF
- ITOA (e la sua ver. file)
- STAMPA (e la sua ver. file)
- LETTURA

2.2.1 lettura

La funzione *lettura* ci permette di leggere i prodotti che sono contenuti all'interno del file prescelto dall'utente. Nel momento in cui questa funzione viene invocata, per non perdere l'indirizzo del codice a cui tornare, esso viene salvato all'interno di un registro che poi verrà pushato nuovamente al termine della funzione per avere questo indirizzo in cima allo stack. Questo viene fatto poiché lo stack pointer viene continuamente modificato all'interno della funzione.
Infatti, *lettura* si occupa di riconoscere i dati numerici e pusharli uno ad uno all'interno dello stack affinché possano essere ordinati e utilizzati nelle altre funzioni del programma.
Il file viene scansionato cifra per cifra e quando incotra una virgola o un LF (Line Feed) si accorge che il numero è terminato e provvede a renderlo un intero e a pusharlo nello **stack**. Oltre a ciò tiene un contatore di quanti valori sono stati inseriti affinché essi vengano poi rimossi al termine della pianificazione.

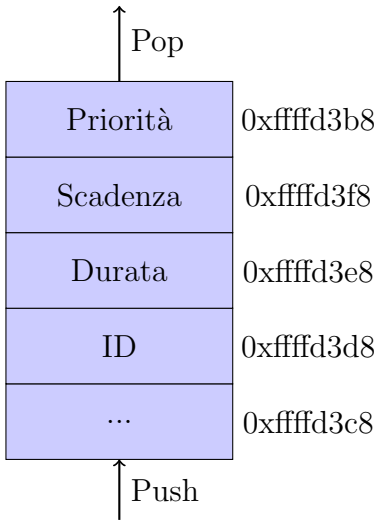


Figure 1: Visualizzazione grafica dello stack durante lettura

Un'altra importante funzione di *lettura* è il riconoscimento degli errori: infatti essa controlla che - per ogni categoria - i valori inseriti non superino quelli stabiliti dalla consegna. Inoltre termina il programma e ordina un ricontrollo nel file da cui leggere se trova incongruenze quali valori non multipli

di quattro (in quanto ogni prodotto possiede quattro categorie) o file vuoto.

2.2.2 Gli algoritmi di ordinamento: EDF e HPF

Gli algoritmi di ordinamento EDF e HPF (spiegati *Introduzione e specifiche del progetto*) hanno entrambi un funzionamento analogo che utilizza la stessa struttura dati ma li ordina per parametri diversi: EDF (Earliest Deadline First) predilige come criterio la scadenza più vicina, mentre HPF (Highest Priority First) ordina a partire dal prodotto con più priorità fino a quello con meno priorità.

Durante il confronto - per esempio - tra due priorità, si possono presentare tre casistiche:

1. la priorità del prodotto più in cima è maggiore, nel qual caso non viene effettuato nessuno scambio;

2. la priorità del prodotto più in cima è minore, nel qual caso bisogna effettuare uno scambio per tutte le categorie del prodotto (ID, Durata, Scadenza, Priorità)

3. le priorità hanno lo stesso valore, nel qual caso per decidere come posizionare i due prodotti bisogna confrontare i valori della scadenza secondo l'algoritmo opposto. Se anche quelle due scadenze fossero uguali, si procede senza nessun criterio particolare.

In caso di HPF, sarà uguale ma scambiando la priorità con la scadenza.
La struttura dati che abbiamo utilizzato per immagazzinare i prodotti è lo **stack**. I singoli dati, già inseriti all'interno della memoria grazie alla funzione *lettura*, vengono - all'interno di queste funzioni - riordinati. Per farlo, ci siamo serviti di un algoritmo di ordinamento, ovvero *Bubble Sort*. Quindi quello che faremo per entrambi gli algoritmi è implementare questo ordinamento ma con le istruzioni assembly e usando lo stack.

Queste funzioni ricevono come parametro il numero dei valori letti da *lettura*. Questa dimensione decide i cicli necessari per ordinare l'intero stack. Abbiamo voluto implementare la versione del Bubble Sort con la *sentinella* che ci permette di uscire dall'algoritmo non appena verrà effettuato un intero ciclo senza uno scambio; questo ci permette di renderlo anche più veloce.

L'effettivo scambio viene effettuato servondoci di due registri come "registri temporanei" per evitare di perdere i dati in memoria.

La strategia che abbiamo utilizzato per muoverci nello stack è stata quella di prendere degli offset - che variano in base all'algoritmo utilizzato e a quale categoria del prodotto si sta prendendo in considerazione - che vengono sommati allo stack pointer, per fargli "puntare" alla posizione in memoria interessata senza muovere lo stack pointer direttamente nè il base pointer.

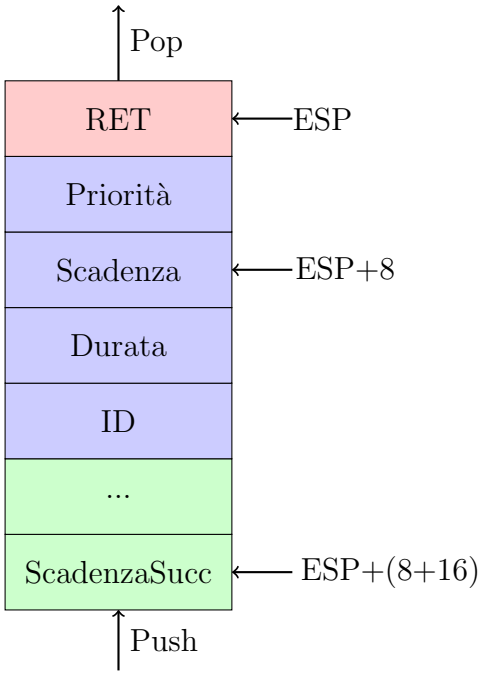


Figure 2: Visualizzazione grafica dello stack durante EDF

2.2.3 stampa e stampafile

La funzione *stampa* e la sua corrispettiva stampafile (la cui differenza è solo quella di stampare su file e non su *stdout*) hanno la principale utilità nel calcolo dei tempi di inizio di produzione di ogni singolo prodotto, del calcolo della penalità e della stampa di tutta la pianificazione dettata dall'algoritmo scelto.
Un puntatore all'interno dello stack preleva il valore *ID* (Si ricordi che in questo punto dell'esecuzione, i prodotti all'interno dello stack sono già ordi-

nati) e lo stampa, servendosi della funzione *itoa*, per poi aumentare, considerare la durata di produzione dello stesso sommarla ad una variabile *tempo* e stampare anche quest'ultima.

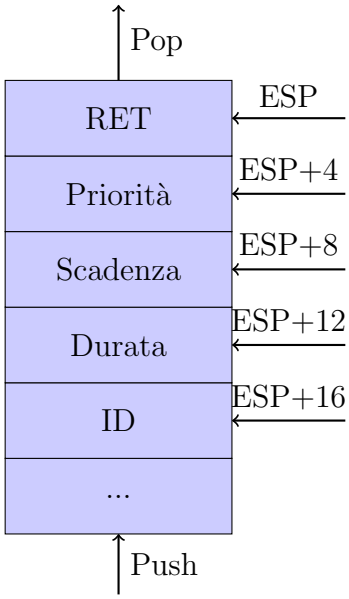


Figure 3: Visualizzazione grafica dello stack

A questo punto la funzione confronta il tempo precedentemente calcolato con la scadenza del prodotto e, qualora quest'ultima dovesse eccedere il tempo previsto, calcola la penalità moltiplicando per la priorità, la differenza tra il tempo e la scadenza. La penalità verrà stampata solo per ultima in quanto deve essere sommata alle penalità di ogni prodotto.

2.2.4 itoa e itoafile

La funzione *itoa* permette di trasformare una variabile da *integer* a *ascii* e di stamparla. Questo passaggio è reso necessario dal fatto che i dati immagazzinati all'interno dello stack sono di tipo intero, mentre in assembly le stampe possono essere effettuate solo con dati di tipo "ascii". La differenza tra le due funzioni presentate è il fatto che itoa si serve dello **stdout** mentre itoafile si serve del **file descriptor** passato come parametro per stamparlo nel file richiesto dall'utente.

3 Benchmark del software

Ci sono tre test principali:

- None.txt che ha penalità maggiore di zero con entrambi gli algoritmi;
- Both.txt che ha penalità uguale a zero con entrambi gli algoritmi;
- EDF.txt che ha penalità uguale a zero con EDF e maggiore di zero con HPF.

`None.txt ha la seguente sequenza di ordini:`

```
35,10,33,4
8,1,39,2
100,3,20,4
43,9,61,1
61,7,8,5
63,3,32,3
108,2,60,1
14,1,16,4
7,8,17,5
107,6,48,3
```

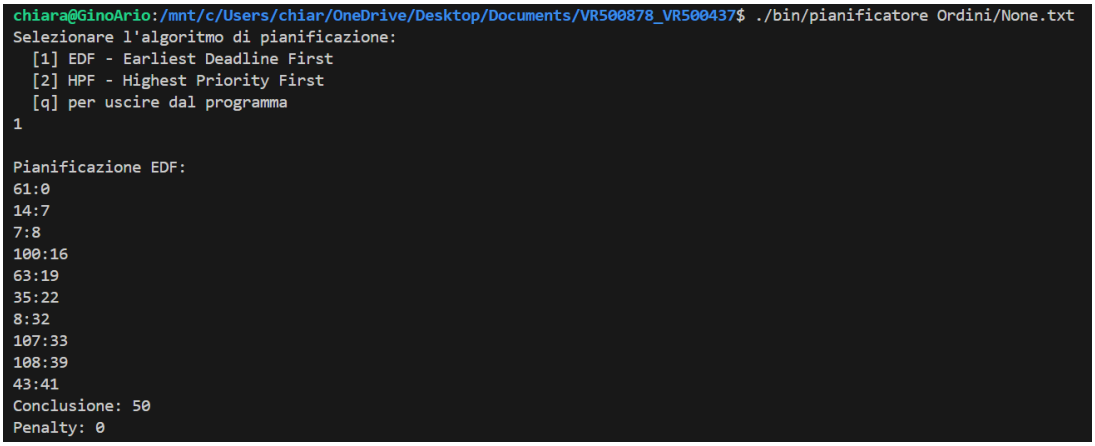


Figure 4: None.txt con EDF

```
Selezionare l'algoritmo di pianificazione:
  [1] EDF - Earliest Deadline First
  [2] HPF - Highest Priority First
  [q] per uscire dal programma
2

Pianificazione HPF:
69:0
16:2
73:10
10:15
34:25
78:34
121:42
19:50
85:53
87:59
Conclusione: 60
Penalty: 215
```

Figure 5: None.txt con HPF

Both.txt ha la seguente sequenza di ordini:

```
12,3,30,5
9,7,92,4
112,1,17,1
33,10,20,3
67,9,48,2
42,4,6,4
3,2,32,4
101,7,32,3
94,7,17,5
32,8,21,1
```

```
chiara@GinoArio:/mnt/c/Users/chiar/OneDrive/Desktop/Documents/VR500878_VR500437$ ./bin/pianificatore Ordini/Both.txt
Selezionare l'algoritmo di pianificazione:
  [1] EDF - Earliest Deadline First
  [2] HPF - Highest Priority First
  [q] per uscire dal programma
1

Pianificazione EDF:
42:0
94:4
112:11
33:12
32:22
12:30
3:33
101:35
67:42
9:51
Conclusione: 58
Penalty: 78
```

Figure 6: Both.txt con EDF

```
Selezionare l'algoritmo di pianificazione:
  [1] EDF - Earliest Deadline First
  [2] HPF - Highest Priority First
  [q] per uscire dal programma
2

Pianificazione HPF:
61:0
7:7
14:15
100:16
35:19
63:29
107:32
8:38
108:39
43:41
Conclusione: 50
Penalty: 0
```

Figure 7: Both.txt con HPF

`EDF.txt ha la seguente sequenza di ordini:`

`121,8,12,2`

```
10,10,67,4
73,5,31,4
85,6,27,1
16,8,50,5
78,8,42,3
19,3,27,2
34,9,18,3
87,1,53,1
69,2,36,5
```

```
chiara@GinoAri:/mnt/c/Users/chiar/OneDrive/Desktop/Documents/VR500878_VR500437$ ./bin/pianificatore Ordini/EDF.txt
Selezionare l'algoritmo di pianificazione:
  [1] EDF - Earliest Deadline First
  [2] HPF - Highest Priority First
  [q] per uscire dal programma
1

Pianificazione EDF:
121:0
34:8
19:17
85:20
73:26
69:31
78:33
16:41
87:49
10:50
Conclusione: 60
Penalty: 0
```

Figure 8: EDF.txt con EDF

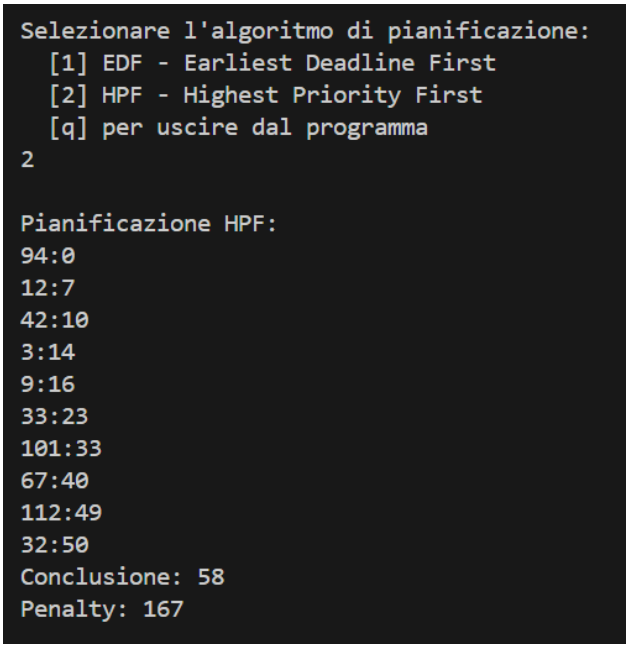


Figure 9: EDF.txt con HPF