UNIVERSITÀ
di VERONA

Elaborato Assembly

Corso di Architettura degli Elaboratori
Prof. Fummi Franco, Prof. Lora Michele

A.A. 2023/2024

Ariton Maria Gemma – VR500028
Darie Alexandru – VR500116

Indice

1. Specifiche del progetto **2**
2. Suddivisione dei file **4**
2.1 pianificatore.s 4
2.2 read.s 5
2.3 EDF.s. 6
2.4 HPF.s. 7
2.5 itoa.s. 7
2.6 print.s. 7
3. Risultati degli algoritmi **8**
3.1 EDF 8
3.2. HPF. 9
4. Scelte progettuali **11**

1. Specifiche

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. Il sistema produttivo 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 da1 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: 7; Scadenza: 32; Priorità: 1;

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:

- **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.
- **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:

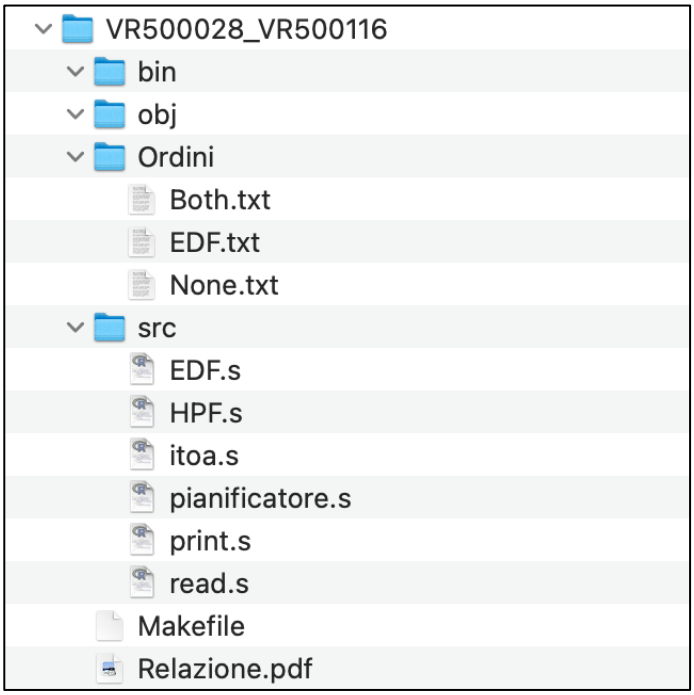
Pianificazione EDF:
4:0
12:10
Conclusione: 17
Penalty: 0
Mentre, se si fosse usato l'algoritmo HPF:
Pianificazione HPF:
12:0
4:17
Conclusione: 17
Penalty: 20

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.

2. Suddivisione dei file

Abbiamo deciso di suddividere il progetto in più file, cosi da semplificarne la comprensione e la scrittura.
L'organizzazione dei file del progetto è la seguente:



Per il corretto funzionamento del programma, al suo avvio nel terminale è fondamentale richiamare il pianificatore e, insieme ad esso, un file *.txt valido.*

2.1 pianificatore.s

Il pianificatore è la componente principale, allo stesso livello di un file ***main*** in C.
Inizialmente si occupa di controllare che i parametri immessi nel terminale non siano più del limite (nel nostro caso 2, *`./bin/pianificatore Ordini/Esempio.txt`*), oppure che non ce ne siano di mancanti, nel caso viene stampato il messaggio di errore opportuno.
Viene controllato il file, tentandone l'apertura per poi proseguire oppure saltando alla funzione di uscita per errore di apertura file (stderror 2) stampando il messaggio d'errore in caso di file non apribile.
Successivamente, si richiama la funzione ***read*** (contenuta in *read.s*) con la quale si effettua la lettura del file e la sua conformità alle specifiche.

Si procede con la stampa a terminale del menù di scelta degli algoritmi (***EDF*** – ***HPF***):

```
id044ilm@aula-virtuale-004:~/Scrivania/ASM$ ./bin/pianificatore Ordini/Both.txt
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
```

E con il successivo inserimento da tastiera dell'algoritmo desiderato.
A seconda dell'algoritmo scelto dall'utente si salterà all'etichetta corrispondente, dove verrà chiamata la funzione opportuna.

Viene, inoltre, chiamata anche la funzione ***print*** (contenuta in print.s) per la stampa a terminale delle specifiche richieste da consegna.
Per esempio (in questo caso si è scelto EDF):

```
1

Pianificazione EDF:
12:0
4:7
14:9
1:18
9:21
24:25
44:35
111:36
10:43
8:47
Conclusione: 57
Penalty: 0
```

Nel caso in cui l'utente inserisca il valore "3" ovvero la funzione d'uscita, il pianificatore salva il numero di valori, per svuotare lo stack da tutti i parametri caricati precedentemente durante la lettura del file, per poi infine chiudere il file e di conseguenza fermare l'esecuzione del programma.

2.2 read.s

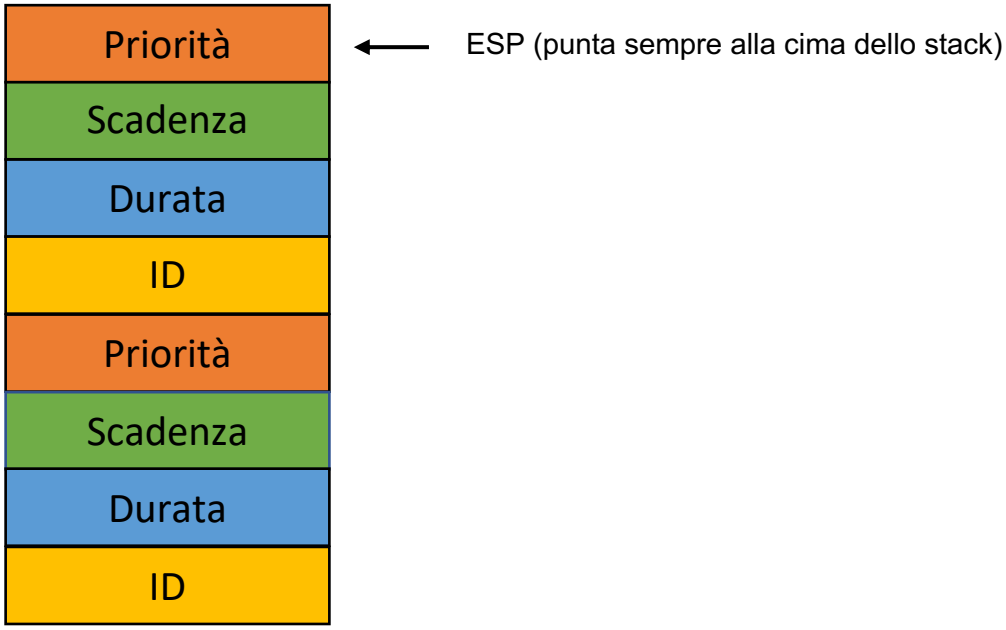
In questo file è contenuta la funzione ***read*** chiamata dal pianificatore.
Inizialmente viene salvato l'indirizzo di ritorno in un registro, così da poter ritornare al punto dove la funzione è stata chiamata. In seguito si controlla ulteriormente che il file sia stato aperto correttamente in *pianificatore* e se contiene qualche valore.

Si comincia il ciclo di lettura dei caratteri all'interno del file, leggendo un carattere alla volta per infine comporne il numero rappresentativo dei singoli argomenti del prodotto da inserire sullo stack (*ID, Durata, Scadenza, Priorità)*.

Per inserire un argomento sullo stack prima viene fatta la conversione da ASCII a intero, e poi concatenate le varie cifre per crearne il valore finale;
Dopo ciascun carattere letto si fa controlla che esso non sia uguale ad una virgola, all'andata a capo (\n) oppure al carriage return (\r). In caso contrario vuol dire che la composizione del valore finale è terminata e quindi lo si può caricare sullo stack incrementandone il contatore.

Nel mentre si compone il valore si utilizza un algoritmo di controllo per ciascun argomento, per controllare che esso sia entro i limiti specificati dalla consegna (*per esempio: 1< ID <127)*.

Nel caso in cui l'argomento non rispetti i limiti, vuol dire che il file contiene elementi non conformi alla consegna e viene stampato un messaggio d'errore apposito.
Alla fine della lettura del file, avremo uno stack cosi composto (*vengono riportati per semplicità solo 2 prodotti)*:



Da notare che questo è lo stack prima che venga effettuato il push di esi e quindi il ritorno alla funzione chiamante.

2.3 EDF.s

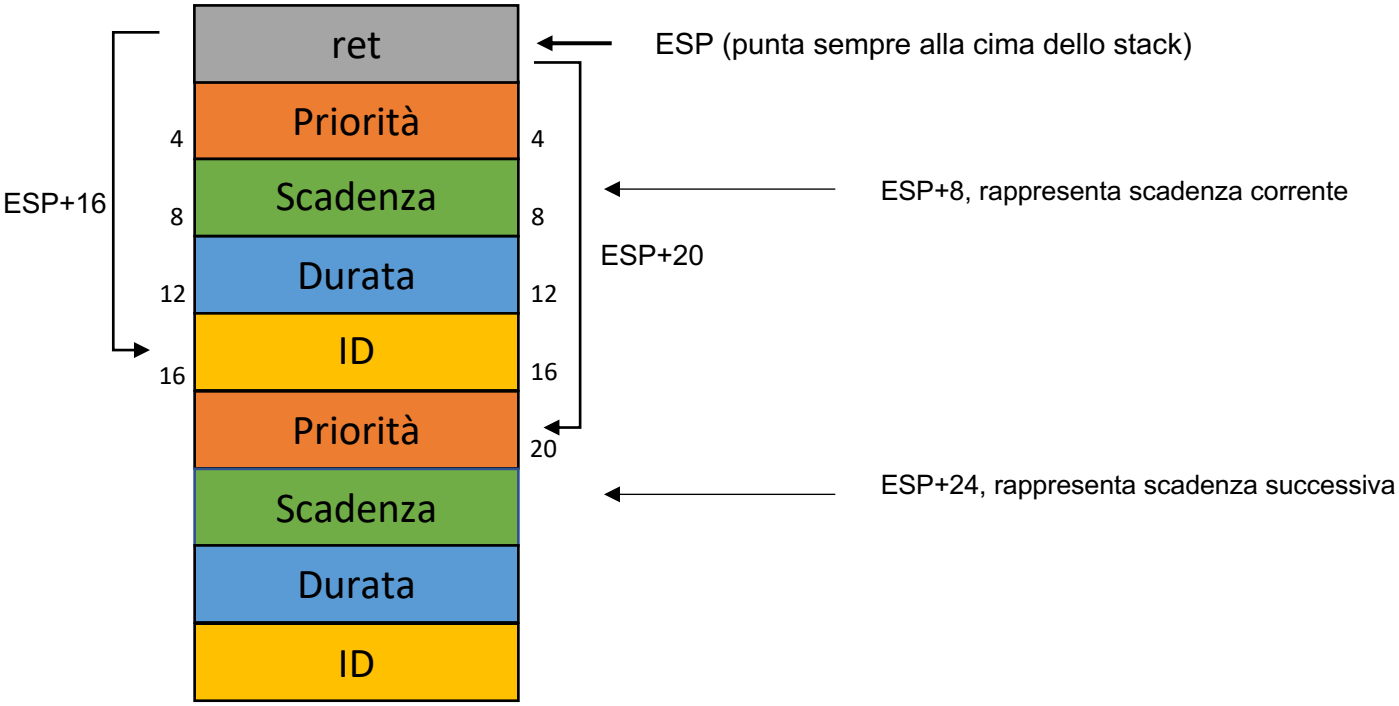
Inizialmente si divide il numero di valori sullo stack per 4(essendo 4 argomenti per prodotto) così da ottenere il numero totali di prodotti da controllare per il riordino.
Si utilizzano le variabili ***scad*** e ***next-scad*** per calcolare i valori di offset, sommati a ESP per spostarci nei vari parametri.

Inizialmente vengono confrontate le *scadenze* dei primi due prodotti, se:

- la **prima** è maggiore vengono scambiate tra di loro
- risultano **uguali**, si passa a controllare quale tra i due prodotti ha la priorità più alta, e nel caso si attua un riordino, se anche la priorità è uguale o in ordine corretto, ci si sposta al prodotto successivo;

Altrimenti, sono in ordine corretto e viene incrementato il contatore dei prodotti riordinati.
Viene inoltre utilizzata una flag per controllare di aver completato il riordino tra tutti i prodotti (nel caso si abbiano tanti prodotti, bisogna assicurarsi che sia stata valutata ogni possibile combinazione).

Se è necessario il riordino modificando man mano i vari valori di offset ci spostiamo nei rispettivi parametri di entrambi i prodotti (per esempio Scadenza di P1 e Scadenza di P2) per scambiarli tra loro, fino al completamento del riordino che è quando si arriva all'ultimo argomento (iniziando da Priorità e finendo in ID).
Al termine del riordino, viene settata la *flag* a 0 e si controlla l'effettivo completamento del riordino tra tutti i prodotti.
Alla fine del riordino, avremo uno stack cosi composto (*vengono riportati per semplicità solo 2 prodotti)*:



2.4 HPF.s

Ha la medesima struttura della funzione *EDF*, solo che non si confronta più per scadenza ma per priorità.
Si confronta per priorità, e se:

- La **prima** è minore della seconda, vengono scambiate tra di loro in quanto la priorità più alta deve essere per prima;
- Risultano **uguali** si confronta per scadenza, dove la scadenza minore andrà per prima, nel caso siano già in ordine o uguali, ci si sposta all'eventuale prodotto successivo.

2.5 itoa.s

Si occupa di convertire un intero in una stringa, ha bisogno che il numero da convertire venga caricato nel registro EAX, cosa già fatta ogni qualvolta venga richiamata la funzione.

2.6 print.s

Inizialmente si inizializza ECX a 4 per poi usarlo per calcolare i vari offset (essendo che ci si sposta di 4 man mano) oltre ad usarlo come contatore per assicurarci di aver trattato tutti i prodotti nello stack.
Ci si sposta al ID del primo prodotto nello stack riordinato, sfruttando gli offset, lo si stampa chiamando *itoa.*
In seguito, si stampa il tempo di inizio che è inizialmente 0 e poi verrà cambiato aggiungendogli man mano la *Durata* del prodotto corrente.

Viene confrontata la *scadenza* con il tempo impiegato se il minuto di conclusione della produzione del prodotto è maggiore della scadenza, dobbiamo passare a calcolare la penalità, altrimenti si passa al prodotto successivo.

Per calcolare la penalità, sempre sfruttando gli offset ci spostiamo al campo della scadenza, la si sottrae dal tempo per trovare cosi le unità di tempo di ritardo. Ci si sposta poi nel campo *priorità* e si effettua la moltiplicazione tra quest'ultima e le unità di tempo.
Il risultato trovato viene aggiunto alla variabile **penality**. Vengono poi stampate secondo le indicazioni della consegna (minuto di conclusione e penalità totale).

3. Risultati degli algoritmi

3.1 EDF

- EDF.txt:

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
1

Pianificazione EDF:
1:0
3:8
14:12
9:14
12:18
6:22
28:24
116:34
7:42
20:44
Conclusione: 54
Penalty: 0
```

EDF con EDF.txt ha Penalità = 0

- Both.txt

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
1

Pianificazione EDF:
12:0
4:7
14:9
1:18
9:21
24:25
44:35
111:36
10:43
8:47
Conclusione: 57
Penalty: 0
```

EDF con Both.txt ha Penalità = 0

- None.txt:

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
1

Pianificazione EDF:
24:0
9:4
23:11
22:14
3:21
127:31
10:35
2:43
71:47
2:51
Conclusione: 60
Penalty: 187
```

EDF con None.txt ha Penalità > 0

3.2 HPF

- EDF.txt:

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
2

Pianificazione HPF:
6:0
116:2
12:10
20:14
3:24
28:28
1:38
14:46
9:48
7:52
Conclusione: 54
Penalty: 161
```

HPF con EDF.txt ha Penalità > 0

- Both.txt

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
2

Pianificazione HPF:
12:0
14:7
4:16
1:18
24:21
9:31
111:35
44:42
8:43
10:53
Conclusione: 57
Penalty: 0
```

HPF con Both.txt ha Penalità = 0

- None.txt:

```
Scegliere l'algoritmo di pianificazione:
  1 - Earliest Deadline First(EDF)
  2 - Highest Priority First(HPF)
  3 - Exit
2

Pianificazione HPF:
9:0
127:7
24:11
2:15
2:19
10:28
71:36
3:40
23:50
22:53
Conclusione: 60
Penalty: 191
```

HPF con None.txt ha Penalità > 0

4. Scelte progettuali

- Abbiamo scelto di gestire l'uscita dal programma nel menù di scelta dell'algoritmo, l'utente inserendo "3" chiuderà il programma.
- Abbiamo notato che non includendo il carriage return, veniva gestita in modo errato la lettura della nuova riga, non bastando solo "\n" e quindi servendoci anche "\r\n".
  Quindi abbiamo implementato che si controlli come prima cosa se è "\n", non lo è e quindi vuol dire che è per forza "\r" che sarà seguito da "\n" quindi completando la lettura della riga.
- Abbiamo scelto di separare il programma in più file, così da semplificare il codice e la comprensione.