UNIVERSITÀ DEGLI STUDI DI VERONA

DIPARTIMENTO DI INFORMATICA

Elaborato ASM

Laboratorio di Architettura degli Elaboratori

Relazione a cura di:
Rebonato Mattia - VR500232
Pellizzari Andrea - VR502128

Anno accademico 2023-2024

1. Sommario

2. Introduzione e specifiche

È richiesto lo sviluppo di un software in assembly, con sintassi AT&T utile per la pianificazione delle attività di un sistema produttivo.
Il sistema si basa su 10 prodotti, realizzati nelle prossime 100 unità di tempo dette “slot temporali”.
Il sistema può realizzare più prodotti, ma solo uno contemporaneamente.
Ogni prodotto è caratterizzato da:

- 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.

I prodotti verranno caricati da un file .txt, contenuto nella cartella “/Ordini/”, per ogni file il limite è di 10 prodotti.
Una volta letto il file di input, 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:

- *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.

Ad ogni esecuzione dell’algoritmo è richiesta la stampa a video delle statistiche. In aggiunta, nel caso l’utente inserisca 2 parametri, il secondo parametro sarà identificato come file di stampa, ad ogni esecuzione dell’algoritmo sarà richiesta anche la stampa su file delle statistiche.

3. Struttura del progetto e organizzazione file

Il progetto è stato strutturato seguendo le specifiche indicate dalla consegna, organizzato in 4 cartelle:

- VR500232_VR502128/
    - src/
        - EDF.s
        - HPF.s
        - itoa.s
        - itoafile.s
        - main.s
        - readfile.s
        - saveparam.s
    - obj/
    - bin/
    - Makefile
    - Relazione.pdf
    - Ordini/
        - EDF.txt
        - Both.txt
        - None.txt

Il progetto, come richiesto, viene compilato da terminale attraverso il comando “make”.

4. Scelte strutturali

File di scrittura

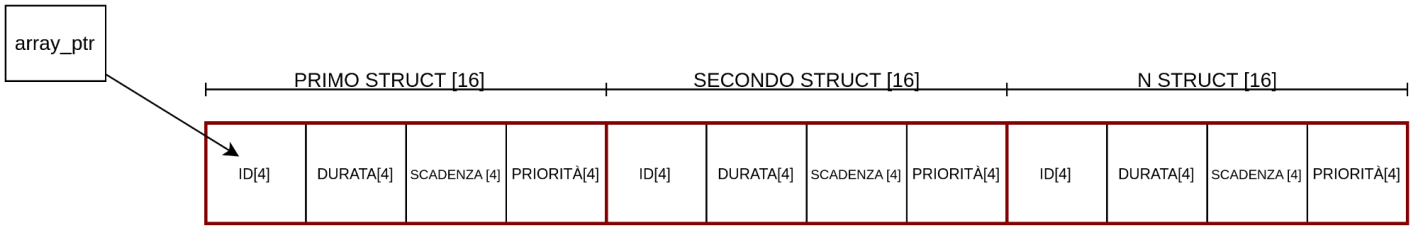
A seconda del numero di parametri, nel caso essi fossero 2, viene immediatamente alzato un flag “sceltascrittura” il quale verrà utilizzato per memorizzare la scelta dell’utente.
Il file viene aperto dopo aver settato il flag e rimarrà aperto per tutta l’esecuzione del programma e si chiude dopo che l’utente sceglie l’opzione di chiusura.

Nel caso in cui il file non esistesse esso verrà automaticamente creato, in caso contrario il file esistente verrà sovrascritto.

L’operazione di scrittura avviene come un semplice scrittura a video, solo che in *%ebx* viene passato il file descriptor al posto della costante *$1*.

Struttura di memoria e salvataggio dei dati

Il salvataggio dei dati letti da file viene effettuato tramite l’implementazione di un array di struct allocato dinamicamente.



La variabile *array_prt* contiene per appunto l’indirizzo del primo elemento dello struct di array.

Per recuperare la grandezza corretta dell’array, abbiamo conteggiato prima di tutto le righe del file, una volta conteggiate, abbiamo moltiplicato il numero di righe * 16 (ogni intero occupa 4 byte, ogni elemento dell’array contiene 4 variabili, 4*4 = 16).

Quindi, nel caso in cui venissero inseriti 10 prodotti, verrebbero allocati dinamicamente:
$numero\ righe\ \cdot (dimensione\ intero\ \cdot\ numero\ variabili\ intere) = 10 \cdot (4byte \cdot 4) = 160byte$

L’allocamento avviene attraverso 2 system call:

- **prima system call:** determina l'attuale limite superiore dell'heap, che viene utilizzato come punto di partenza per la nuova allocazione di memoria.

- **seconda system call:** Aggiunge la quantità di memoria desiderata al break attuale e aggiorna il break con il nuovo valore, aumentando così l'heap disponibile per il programma.

Le system call in questione sono le numero *$45*, brk.

Codice:

```
# Ottengo l'attuale break
movl $45, %eax                  # Syscall 45: brk
xorl %ebx, %ebx                 # Passo 0 per ottenere l'attuale break
int $0x80                       # Effettuo la syscall
movl %eax, %esi                 # Salvo l'attuale break in %esi

# Imposto il nuovo break
addl array_size, %esi           # Aggiungo la dimensione della memoria da allocare al break attuale
movl $45, %eax                  # Syscall 45: brk
movl %esi, %ebx                 # Passo il nuovo break
int $0x80                       # Effettuo la syscall

testl %eax, %eax                # Controllo se il valore richiesto e quello ottenuto sono uguali
jl _exit_error                  # Se fallisce, esco con errore

mov %eax, array_ptr
```

Una volta allocata la memoria, procediamo con la lettura dei parametri da file, i quali vengono convertiti in variabili *long* nel caso in cui venga letto:

- una virgola
- carattere di fine riga
- carattere di fine file

Qualunque altro carattere differente da questi 3 e da un carattere numerico, porterebbe il programma in uno stato di errore, il quale verrà comunicato e l'esecuzione verrebbe interrotta.

Algoritmi e ordinamento degli elementi

Le specifiche di progetto comunicavano la necessità di implementare degli algoritmi che mirassero ad emulare le politiche di *'Earliest Deadline First' e di ' Highest Priority First'*, al fine di gestire la produzione di oggetti da parte di un'azienda. Successivamente si richiede il calcolo delle statistiche rispetto all' esecuzione di ognuno degli algoritmi. Questi due algoritmi si basano sull' ordinamento degli elementi secondo un singolo criterio:
nel caso dell' EDF il criterio è la scadenza (Deadline) per la produzione del prodotto. Verranno ordinati gli elementi in modo tale che i prodotti con scadenza breve siano trattati per primi.

nel caso del HPF il criterio è la priorità(Priority) per la produzione del prodotto. Verranno ordinati gli elementi in modo tale che i prodotti con la priorità più alta siano svolti per primi.

Per implementare l' ordinamento, sia in EDF che in HPF, ci siamo avvalsi dell' algoritmo di ordinamento *'Bubble sort'.* Ognuno dei due file relegati a svolgere i calcoli per i rispettivi algoritmi contiene una versione specifica mirata ad ordinare in base alla scadenza nel primo caso (EDF) e alla priorità nel secondo caso (HPF).

Entrambi i file sopracitati contengono perciò due cicli, contrassegnati con l' etichetta 'loop' e 'loop_2', che implementano lo scorrimento degli elementi voluto dall' algoritmo bubble sort. In caso si verifichi la condizione sopracitata necessaria per lo scambio di due elementi in ordine scorretto, in entrambi gli algoritmi si effettua un salto all' etichetta scambio, il cui compito sarà spostare le informazioni contenute in memoria dinamica del primo elemento in favore del secondo (che salirà nella lista dei processi produttivi da eseguire).
Una volta terminata la fase di ordinamento entrambi gli algoritmi procedono con la fase di calcolo delle statistiche.

La fase di calcolo avviene nell' etichetta 'conteggio', nella quale si effettua un ciclo che scorre gli elementi ed effettua il calcolo della durata e in caso di penalità aggiunge il valore della penalità al denaro da pagare come multa. L' algoritmo si conclude con l' etichetta 'fine', a cui segue la stampa delle statistiche.

5. Fase di testing

- ***File EDF.txt***

Elementi:
1,2,5,3
5,3,15,2
2,4,10,3
7,8,25,2
9,4,30,4
4,6,35,5
2,3,40,3
6,9,45,2
8,1,50,1
3,2,55,5

Algoritmo EDF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
1	0	->	2	5	5	0
2	2	->	6	10	3	0
5	6	->	9	15	2	0
7	9	->	17	25	2	0
9	17	->	21	30	4	0
4	21	->	27	35	5	0
2	27	->	30	40	3	0
6	30	->	39	45	2	0
8	39	->	40	50	1	0
3	40	->	42	55	5	0

Durata: 42 Penalità: 0

Algoritmo HPF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
4	0	->	6	35	5	+0
3	6	->	8	55	5	+0
9	8	->	12	30	4	+0
1	12	->	14	5	3	(14-5)*3 = +27
2	14	->	18	10	3	(18-10)*3 = +24
2	18	->	21	40	3	+0
5	21	->	24	15	2	(24-15)*2 = +18
7	24	->	32	25	2	(32-25)*2 = +14
6	32	->	41	45	2	+0
8	41	->	42	50	1	+0

Durata: 42 Penalità: 83

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 1

Pianificazione EDF:
1:0
2:2
5:6
7:9
9:17
4:21
2:27
6:30
8:39
3:40
Conclusione: 42
Penalty: 0
```

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 2

Pianificazione HPF:
4:0
3:6
9:8
1:12
2:14
2:18
5:21
7:24
6:32
8:41
Conclusione: 42
Penalty: 83
```

Test 'EDF.txt': In EDF.txt la penalità è uguale a zero con EDF e maggiore di zero con HPF, ciò è deducibile dalla prima immagine in cui sono state calcolate le statistiche prima dell' esecuzione, e dalle e immagini che seguono, che sono risultanti dall' esecuzione effettiva degli algoritmi.

- ***File Both.txt***

Elementi:
1,2,20,3
2,3,24,2
3,4,17,3
4,8,33,2
5,4,30,4
6,6,35,5
7,3,40,3
8,9,45,2
9,1,51,1
10,2,55,5

Algoritmo EDF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
3	0	->	4	17	3	0
1	4	->	6	20	3	0
2	6	->	9	24	2	0
5	9	->	13	30	4	0
4	13	->	21	33	2	0
6	21	->	27	35	5	0
7	27	->	30	40	3	0
8	30	->	39	45	2	0
9	39	->	40	51	1	0
10	40	->	42	55	5	0

Durata: 42 Penalità: 0

Algoritmo HPF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
6	0	->	6	35	5	0
10	6	->	8	55	5	0
5	8	->	12	30	4	0
3	12	->	16	17	3	0
1	16	->	18	20	3	0
7	18	->	21	40	3	0
2	21	->	24	24	2	0
4	24	->	32	33	2	0
8	32	->	41	45	2	0
9	41	->	42	51	1	0

Durata: 42 Penalità: 0

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 1

Pianificazione EDF:
3:0
1:4
2:6
5:9
4:13
6:21
7:27
8:30
9:39
10:40
Conclusione: 42
Penalty: 0
```

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 2

Pianificazione HPF:
6:0
10:6
5:8
3:12
1:16
7:18
2:21
4:24
8:32
9:41
Conclusione: 42
Penalty: 0
```

Test 'Both.txt': In Both.txt la penalità è uguale a zero con EDF e anche tramite HPF.

- ***File None.txt***

Elementi:
1,3,4,2
2,2,5,1
3,4,6,3
4,2,7,2
5,3,9,1
6,1,3,5
7,5,8,4
8,2,6,3

Algoritmo EDF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
6	0	->	1	3	5	+0
1	1	->	4	4	2	+0
2	4	->	6	5	1	(6-5)*1 = +1
3	6	->	10	6	3	(10-6)*3 = +12
8	10	->	12	6	3	(12-6)*3 = +18
4	12	->	14	7	2	(14-7)*2 = +14
7	14	->	19	8	4	(19-8)*4 = +44
5	19	->	22	9	1	(22-9)*1 = +13

Durata: 22 Penalità: 102

Algoritmo HPF:

ID	Inizio		Tempo	Scadenza	Priorità	Penalità
6	0	->	1	3	5	+0
7	1	->	6	8	4	+0
3	6	->	10	6	3	(10-6)*3 = +12
8	10	->	12	6	3	(12-6)*3 = +18
1	12	->	15	4	2	(15-4)*2 = +22
4	15	->	17	7	2	(17-7)*2 = +20
2	17	->	19	5	1	(19-5)*1 = +14
5	19	->	22	9	1	(22-9)*1 = +13

Durata: 22 Penalità: 99

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 1

Pianificazione EDF:
6:0
1:1
2:4
3:6
8:10
4:12
7:14
5:19
Conclusione: 22
Penalty: 102
```

```
Scegli che algoritmo utilizzare:
 ➢ 1 = EDF;
 ➢ 2 = HDF;
 ➢ 3 = uscita
Inserisci valore: 2

Pianificazione HPF:
6:0
7:1
3:6
8:10
1:12
4:15
2:17
5:19
Conclusione: 22
Penalty: 99
```

Test 'None.txt': In None.txt la penalità è maggiore di zero con entrambi gli algoritmi.