Elaborato Assembly

Corso di Architettura degli Elaboratori A.A. 2023/2024
Prof. Franco Fummi, Prof. Michele Lora

Tommi Bimbato VR500751, Antonio Iovine VR504083

24 maggio 2024

Indice

1 Introduzione **1**

1.1 Approccio progettuale 1

1.2 Analisi delle specifiche 1

1.2.1 Input 1

1.2.2 Algoritmi di pianificazione 2

1.3 Sviluppo 3

1.3.1 Acquisizione e struttura dati 3

1.3.2 Ordinamento dei prodotti 4

1.3.3 Output 4

1.3.4 Penalità 4

1.4 Codice 4

1.4.1 File sorgente 4

1.4.2 Analisi di alcune parti fondamentali del codice 5

Sommario

L'obiettivo del progetto è sviluppare un software per la pianificazione delle attività di un sistema produttivo. La produzione è organizzata in slot temporali uniformi, durante i quali un solo prodotto può essere in fase di lavorazione. Il software consentirà di ottimizzare la pianificazione delle attività secondo due algoritmi di pianificazioni differenti. L'intero software è stato sviluppato in linguaggio Assembly (Sintassi AT&T) e testato su un insieme di dati di prova allegati a questa documentazione.

Capitolo 1

Introduzione

1.1 Approccio progettuale

L'elaborato è stato condotto seguendo un approccio metodologico strutturato. Inizialmente, è stata eseguita un'analisi dettagliata per identificare i requisiti e le funzionalità principali del software. Questo processo ha consentito una comprensione completa del contesto operativo e degli obiettivi da raggiungere.

Successivamente, è stata sviluppata una bozza del software utilizzando il linguaggio di programmazione C. Questo ha permesso la traduzione dei requisiti in una struttura logica e l'identificazione dell'architettura generale del software.

Parallelamente, sono stati definiti gli spazi di memoria necessari per la memorizzazione dei dati durante l'esecuzione del programma. Ciò ha garantito un utilizzo efficiente delle risorse disponibili.

Infine, sono stati eseguiti test approfonditi per verificare il corretto funzionamento del programma e identificare eventuali aree di miglioramento. L'iterazione su questo processo ha portato a modifiche e ottimizzazioni fino al raggiungimento di un livello soddisfacente di prestazioni e funzionalità.

1.2 Analisi delle specifiche

La produzione è organizzata in slot temporali uniformi, durante i quali un solo prodotto può essere in fase di lavorazione. Il programma analizza una serie di prodotti, ognuno caratterizzato da un identificativo, una durata, una scadenza e una priorità secondo le specifiche seguenti:

- Identificativo: un codice da 1 a 127;
- Durata: il numero di slot temporali per il completamento (da 1 a 10);
- Scadenza: il limite massimo di tempo entro cui il prodotto deve essere completato (da 1 a 100);
- Priorità: un valore da 1 a 5, che indica sia la priorità che la penalità per il ritardo sulla scadenza[1].

Al termine della pianificazione, il programma calcolerà la penale dovuta agli eventuali ritardi di produzione.

1.2.1 Input

L'utente invoca il programma "pianificatore" e fornisce due file come parametri da linea di comando, il primo viene considerato input, mentre il secondo viene utilizzato per salvare i risultati della pianificazione. Ad esempio:

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

[1] Il valore 5 indica la priorità più alta.

In questo caso, il programma caricherà gli ordini dal file `Ordini.txt` e salverà le statistiche stampate a video nel file `Pianificazione.txt`.

Se l'utente fornisce solo un parametro, il salvataggio della pianificazione su file verrà ignorato.

Il file degli ordini dovrà avere un prodotto per riga, con tutti i parametri separati da virgola. Ad esempio, se l'ordine fosse:

```
ID: 4; Durata: 10; Scadenza: 12; Priorità: 4;
```

Il file dovrà contenere la seguente riga:

```
4,10,12,4
```

1.2.2 Algoritmi di pianificazione

Una volta letto il file, il programma visualizzerà il menu principale, permettendo all'utente di selezionare l'algoritmo di pianificazione desiderato. Le opzioni disponibili sono:

1. Earliest Deadline First (EDF): Si pianificano per primi i prodotti con scadenza più vicina. In caso di parità nella scadenza, si considera la priorità più alta.
2. Highest Priority First (HPF): Si pianificano per primi i prodotti con la priorità più alta. In caso di parità di priorità, si considera la scadenza più vicina.

L'utente può selezionare uno dei due algoritmi per la pianificazione delle attività del sistema produttivo.

1.3 Sviluppo

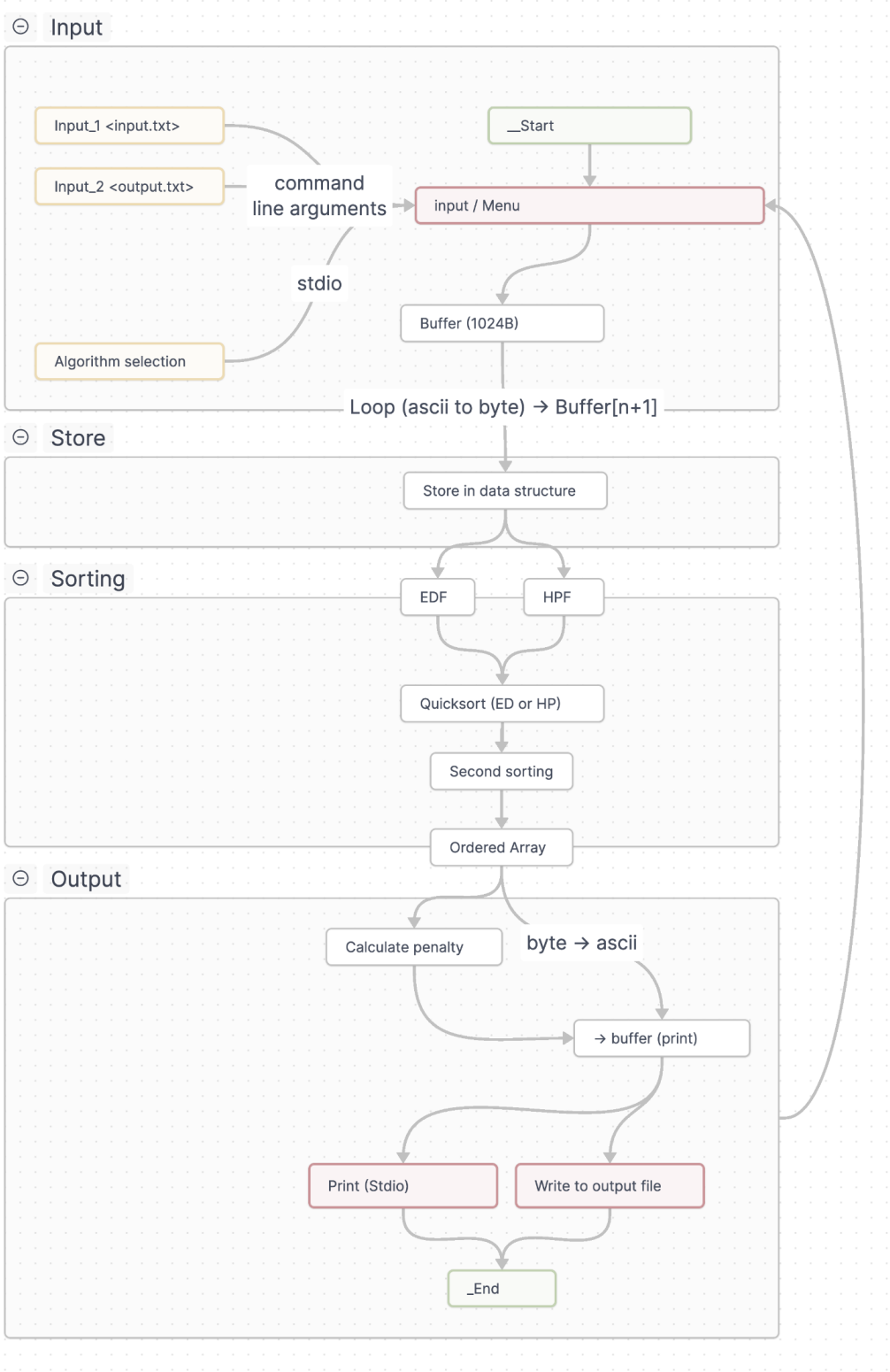

Figura 1.1: Schema del funzionamento del software.

1.3.1 Acquisizione e struttura dati

Il software acquisisce l'intero contenuto del file di input e lo memorizza in una variabile nel buffer, parallelamente ne calcola la lunghezza (numero di righe) per allocare la memoria

necessaria per la struttura dati vera e propria. Successivamente, il programma procede a leggere il contenuto del buffer, converte le informazioni dal formato ASCII al formato numerico e le memorizza in un array di struct. La peculiarità di questa struttura dati è che lo spazio occupato da un prodotto (ID, durata, scadenza e priorità) è di 4 byte totali, pari alla dimensione di un registro interno del processore considerato per l'applicazione del software.

1.3.2 Ordinamento dei prodotti

Acquisita la struttura dati, il programma acquisisce la scelta dell'algoritmo di pianificazione da parte dell'utente e procede con l'ordinamento dei prodotti in base alla scelta effettuata. Il sorting è gestito da una variabile che definisce la priorità di ordinamento (HPF o EDF) che viene passata alla funzione di ordinamento come parametro. La funzione di ordinamento, a sua volta, si occupa di confrontare i prodotti in base alla priorità scelta e di riordinarli in base a tale criterio nella struttura dati stessa.

Per l'effettivo ordinamento dei prodotti, il software utilizza un algoritmo di sorting di tipo "insertion sort" che si è dimostrato il più efficiente per la dimensione dei dati in ingresso. Il secondo ordinamento, ovvero quello che compara il parametro "scadenza" in caso di parità di priorità o "priorità in caso di pari scadenza, è gestito da una funzione basata su algoritmo di bubble sort.

1.3.3 Output

Nel caso venisse specificato un file di output, il software stamperà su di esso i risultati della pianificazione secondo il seguente schema:

```
<Algoritmo di pianificazione scelto>:
<ID del prodotto>:<Inizio produzione>
[...]
Conclusione:<timeslot termine produzione>
Penalty:<penalità totale>
```

1.3.4 Penalità

Se un prodotto non viene completato entro la scadenza, il programma calcola la penalità in base alla priorità del prodotto e alla quantità di slot temporali di ritardo. La penalità è calcolata come segue:

$$\text{Penalità} = \text{Priorità} \times \text{Ritardo}$$

1.4 Codice

1.4.1 File sorgente

Il programma è composto dai seguenti file sorgente in linguaggio assembly:

- `btoa.s`: byte to ASCII, funzione per la conversione di un byte in ASCII;
- `BufferToArray.s`: funzione per la conversione del buffer di lettura in un array di struct;
- `InsertionSort.s`: funzione per il primo ordinamento dei prodotti;
- `EqualSort.s`: funzione per il secondo ordinamento dei prodotti;
- `menu.s`: file contenente la funzione per il menù.
- `main.s`: funzione principale del programma.

A completare la struttura del programma, sono presenti i seguenti file di supporto:

- `Makefile`: file per la compilazione del programma;

- EDF.txt: file di input con penalità uguale a zero con EDF e maggiore di zero con HPF;
- HPF.txt: file di input con penalità uguale a zero con HPF e maggiore di zero con EDF;
- Both.txt: file di input con penalità uguale a zero con entrambi gli algoritmi;
- None.txt: file di input con penalità maggiore di zero con entrambi gli algoritmi.

1.4.2 Analisi di alcune parti fondamentali del codice

```
  Codice in ASM!!!
  int main pippo VOID !!!
  ; Gibberish Assembly Code
  mov eax, 0
  add ebx, eax
  sub ecx, ebx

  mul edx, ecx
  div
  nop
  cmp eax, ebx
  jne label
```

```

ul edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:l edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:l edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:l edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:l edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:l edx, ecx
  div esi, edx
  push eax
  pop ebx
  inc ecx
  dec edx
  jmp label
  label:
```

```
  nop
  cmp eax, ebx
  jne label
  sub ecx, ebx
  mul edx, ecx
  div esi, edx
!

  sub ecx, ebx
```