Laboratorio di Architettura degli Elaboratori
Elaborato Assembly 2024
Pianificatore ordini

GIORGIA ZANINI - VR503052

VADIM MUSTEATA - VR422249

A.A. 2023/2024

Indice

Capitolo 1

Specifiche

Il progetto prevede la realizzazione di un software per la pianificazione delle attività di un sistema produttivo.

Comprende quindi la pianificazione fino a un massimo di 10 prodotti, mandati in produzione uno di seguito all'altro, distribuiti in un massimo di 100 unità di tempo, chiamati "slot temporali".
Ogni ordine è caratterizzato da 4 valori, così organizzati:

1. **Identificativo**: è l'identificativo del prodotto da produrre. Può avere un valore compreso tra 1 e 127.

2. **Durata**: è il tempo che serve al prodotto per essere completato. Espresso in unità di tempo. Può avere un valore compreso tra 1 e 10, il che fa si che non si possano superare un massimo di 100 unità di tempo.

3. **Scadenza**: è l'unità di tempo massima in cui terminare la produzione del prodotto stesso. Può avere un valore compreso tra 1 e 100.
   Nel caso in cui questo venisse superato, la produzione subisce una penalità per ogni prodotto che supera la sua rispettiva scadenza, sommate poi per dare la penalità totale.

4. **Priorità**: Può avere un valore compreso tra 1 e 5, dove 5 indica la priorità massima e 1 la minima.

Tutti gli ordini sono salvati in un unico file di testo txt, dove ogni riga corrisponde a un ordine, e i rispettivi valori sono separati tra loro da una virgola.

Penalità

Per ogni prodotto completato in ritardo rispetto alla scadenza indicata nell'ordine stesso, l'azienda dovrà pagare una penale.
Per calcolarla bisogna tenere conto del tempo di fine, della sua scadenza e priorità, e applicare la formula che segue:

$$Penalty = Priority * (tempo\_fine - Scadenza)$$

Per ottenere poi la penalità totale, bisogna sommare quella ciascun prodotto.

Capitolo 2

Struttura del codice e scelte progettuali

2.1 Descrizione generale del codice

Il programma parte con l'esecuzione della funzione *main*, che si occupa di elaborare i parametri ricevuti da terminale e aprire i relativi file. Se non si verificano errori, procede alla lettura del file degli ordini e li salva all'interno di un array, tramite la funzione *salva_numeri*.
A questo punto esegue la funzione *menu*, che manterrà il "controllo" del programma fino alla sua terminazione. Qui viene stampata a video la scelta tra *"Pianificazione EDF"* (Earliest Deadline First), *"Pianificazione HPF"* (Highest Priority First) o *"Esci"*.
Entrambe le pianificazioni faranno una chiamata alla rispettiva funzione di ordinamento e in seguito a *elabora_ordini* che eseguirà la pianificazione effettiva, per poi ritornare nel *menu*.
La terza scelta porterà alla chiusura dei file aperti e alla terminazione del programma, chiamando *termina_programma*.

2.2 Descrizione *main* e funzioni

2.2.1 main

Il *main* ha il compito di analizzare i parametri passati da linea di comando e aprire, se presenti, i file forniti.
Il *main* esegue dalle 3 alle 4 istruzioni pop:

- La prima istruzione di pop preleva dallo stack il numero di parametri passati da terminale e lo salva nella variabile *counter_parametri*;
- La seconda preleva il nome del programma, che non viene utilizzato;
- La terza preleva il percorso del file degli ordini;
- La quarta preleva il percorso del file per salvare i risultati.

Prelevato il percorso del file degli ordini, viene verificato che questo non sia vuoto e, se non lo è, viene eseguita la system call open, in sola lettura, per aprirlo.
Se il risultato restituito è negativo il programma termina con un errore relativo all'apertura del file.
Se la open ha successo viene verificato il numero salvato nella variabile *counter_parametri*: se il suo valore è uguale a 3 il programma suppone che sia stato fornito in input il percorso del file per salvare i risultati, eseguendo, solo in questo caso, la quarta pop. Viene quindi, come per il file precedente, verificato che il valore estratto non sia vuoto, per poi lanciare una nuova system call open provando ad aprire questo file in sola scrittura.
Anche qui se il risultato è negativo il programma termina con un errore relativo all'apertura del file.
Aperto con successo il file degli ordini (e il file per salvare i risultati), il *main* richiama la funzione *salva_numeri*, e al termine di questa, la funzione *menu*.

2.2.2 salva_numeri

La funzione *salva_numeri*, dato in input il file descriptor associato al file degli ordini, ha il compito di leggere il file e salvarne il contenuto in un array, di dimensione 40.
La funzione legge un carattere per volta, e se questo è un numero, lo salva in un array temporaneo, di dimensione 4, per contenere fino a tre cifre e il terminatore per renderlo una stringa; nel momento in cui viene letto un carattere diverso, che può essere una virgola o un a capo, va a convertire la stringa in un intero, esegue i controlli per verificare che il numero rientri nei parametri previsti (se corrisponde all'id del prodotto, verifica che sia compreso tra 1 e 127, ...), per poi salvarlo nell'array principale.
Terminata le lettura del file e salvati tutti i numeri nell'array degli ordini, la funzione verifica che il numero di ordini non sia 0, non sia a 10, e che ciascun ordine sia composto da 4 valori.
Superati i controlli, la funzione restituisce il puntatore all'array degli ordini e la dimensione di quest'ultimo (numero di ordini * 4).
In caso di errore, che sia relativo al range di un valore letto o al numero degli ordini, stampa l'errore relativo e termina.

2.2.3 menu

La funzione *menu*, prende in input il file descriptor del primo e del secondo file, il puntatore all'array degli ordini e la dimensione di quest'ultimo, e fornisce la scelta tra i due possibili ordinamenti e l'uscita dal programma.
Nel caso degli ordinamenti, rimanda alla funzione rispettiva all'ordinamento scelto, e una volta terminata l'esecuzione di tale ordinamento e la conseguente pianificazione dell'ordine, ritornerà nella scelta proposta dal *menu*; con l'opzione "Esci", essa chiamerà la funzione *termina_programma*, per terminare il programma.

2.2.4 ordinamento_EDF

Prende in input il puntatore all'array degli ordini e la rispettiva dimensione, e ordina l'array in base alla scadenza (più è vicina, prima verrà prodotto), ordinando in maniera crescente rispetto al terzo numero di ognuno. Nel caso in cui due ordini abbiano la medesima scadenza, verrà controllata la priorità di quella coppia, ovvero il quarto numero dei due gruppi. In questo caso i due ordini verranno ordinati in modo decrescente. Infine, se sia la scadenza che la priorità risultano uguali, i due ordini confrontai rimangono nella disposizione di partenza.
La funzione utilizza il metodo del bubble sort per eseguire l'ordinamento.

2.2.5 ordinamento_HPF

Prende in input il puntatore all'array degli ordini e la rispettiva dimensione, e ordina l'array in base alla priorità (più è alta, prima verrà prodotto), ordinando in maniera decrescente rispetto al quarto numero di ognuno. Nel caso in cui due ordini abbiano la medesima priorità, verrà controllata la scadenza di quella coppia, ovvero il terzo numero dei due gruppi. In questo caso la coppia verrà ordinata in maniera crescente. Infine, se sia la priorità che la scadenza risultano uguali, i due ordini confrontati rimangono nella disposizione di partenza.
La funzione utilizza il metodo del bubble sort per eseguire l'ordinamento.

2.2.6 elabora_ordini

Prende in input il file descriptor del file per salvare i risultati, il puntatore all'array degli ordini e la sua dimensione. Ha il compito di eseguire la pianificazione degli ordini.
Per ogni ordine stampa a video e su file (se fornito) l'id del prodotto (primo valore) e il relativo tempo di inizio produzione, che equivale al tempo di fine del prodotto precedente (o al tempo 0 se si tratta del primo prodotto). Per ogni prodotto in elaborazione, calcola il tempo di termine della sua produzione, dato dal tempo di inizio più la rispettiva durata (secondo valore) e calcola la possibile penalità, nel caso in cui terminasse oltre la sua scadenza (terzo valore). Elaborato l'ultimo ordine, stampa inoltre il tempo di fine e la penalità totale ottenuta, data dalla penalità, se presente, di ciascun ordine.

2.2.7 converti_str_a_int

Prende in input la stringa da convertire.
Partendo dall'inizio della stringa, prende una cifra, che va poi a sommare al numero salvato finora (inizializzato a 0) moltiplicato per 10:

$$numero\_salvato = (numero\_salvato * 10) + nuova\_cifra$$

Ripete questo loop fino a quando trova un terminatore.
Ritorna poi l'intero ottenuto.

2.2.8 converti_int_a_str

La funzione prende in input un intero.
Scompone il valore numerico passatogli, prendendo l'ultima cifra e convertendola in ascii, per poi salvarla in un array, partendo dalla penultima cella. Esegue questo loop finché non saranno salvate tutte le cifre.
Nell'ultima cella viene poi posto un terminatore.
Ritornerà poi il puntatore dell'array alla cella corrente, che corrisponderà all'inizio della stringa.

2.2.9 stampa_stringa

Questa funzione prende in input il puntatore a una stringa da stampare e il file descriptor associato al file per salvare i risultati. Se questo è diverso da -1 la stampa avviene anche nel file, altrimenti solo su terminale.
Dato il puntatore alla stringa, che deve sempre terminare con \0, la funzione conta il numero di caratteri che la compongono (escluso il terminatore) e chiama poi la system call write, fornendo come dimensione da stampare il numero contato.

2.2.10 termina_programma

Questa funzione prende in input i file descriptor associati ai due file passati da linea di comando, li chiude se aperti e termina l'esecuzione.
Un file viene considerato aperto se il valore del suo file descriptor è maggiore di -1.

2.2.11 stampa_array

(funzione di debug) Prende in input il puntatore all'array e la sua dimensione, e, tramite *converti_int_a_str* e *stampa_stringa*, converte i valori dell'array e li stampa solo su terminale, intervallati da spazi.

2.3 Disposizione file

- VR503052_VR422249/
  - src/
    - converti_int_a_str.s
    - converti_str_a_int.s
    - elabora_ordini.s
    - main.s
    - menu.s
    - ordinamento_EDF.s
    - ordinamento_HPF.s
    - salva_numeri.s
    - stampa_array.s
    - stampa_stringa.s

    - termina_programma.s
  - obj/
  - bin/
  - Ordini/
    - EDF.txt
    - Both.txt
    - None.txt
  - Makefile
  - Relazione.pdf

In *src* sono presenti i file *.s* che compongono il programma.
Invece *obj* e *bin* saranno rispettivamente le destinazioni dei file *.o* compilati e l'eseguibile, nominato *"pianificatore"*, creati da un *Makefile* da eseguire.
quest'ultimo i occupa anche della pulizia delle stesse cartelle.
I file da passare in input sono contenuti dentro la cartella *Ordini*.
I file degli ordini forniti sono:

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

2.4 Scelte progettuali

1. **Apertura e lettura file, e chiusura degli stessi**
   I file vengono aperti una sola volta, per poi venire chiusi solo alla terminazione del programma. In questo modo si evitano eventuali superflue chiusure e riaperture intermedie, essendo i file sempre gli stessi dall'inizio alla fine dell'esecuzione del programma.
   Secondo questa logica, anche il lettura dal file degli ordini avverrà una sola volta, e i numeri presenti in esso verranno salvati in un array grande al massimo 40, che varrà poi gestito nel *menu* in base alla necessità.

2. **Terminazione programma**
   Il programma non termina nel *main*, ma grazie alla chiamata a un'altra funzione apposita, chiamata *termina_programma*, che si occupa di chiudere i file se aperti e terminare il programma stesso.
   In questo modo la chiusura del programma potrà essere chiamata in qualsiasi momento, anche in caso di errore.
   Per la terminazione effettiva, invece è stata integrata una terza scelta nel *menu*, nominata *"Esci"*, che chiama anch'essa *termina_programma*.

3. **Ordinamento nell'array stesso**
   Per eseguire l'ordinamento verrà utlizzato lo stack, come variabile temporanea, in modo da effettuarlo nell'array stesso.

4. **Utilizzo dei registri**
   Per passare i diversi parametri tra le funzioni, invece dello stack, vengono utilizzati i seguenti registri:

   - EAX: per il file descriptor del file degli ordini;
   - EBX: per il file descriptor del file per salvare i risultati;
   - ESI: per il puntatore all'array degli ordini;
   - ECX: per la dimensione dell'array degli ordini.

   Fanno eccezione le funzioni *stampa_stringa*, *converti_str_a_int*, *converti_int_a_str*, *stampa_array*, in cui EAX viene utilizzato per passare l'intero o il puntatore alla stringa o all'array.

5. **Scrittura su terminale e file**
   Per la scrittura su terminale e su file è stata creata un'unica funzione generica di stampa. Essa esegue la system call write su terminale, e, se il valore del file descriptor passatogli è diverso da -1, allora la esegue anche su file.
   In questo modo si può decidere se stampare solo su terminale o anche su file, se fornito.

6. **Passaggio da uno a due file tramite comando da terminale**
   Eseguendo il codice, si può decidere se passare in input un solo parametro, che corrisponde al file degli ordini, inserendo la seguente riga di comando da terminale:
   `./bin/pianificatore <percorso del file degli ordini>`
   Oppure si può decidere di inserire come parametro anche un file di destinazione, dove viene la pianificazione degli ordini secondo l'algoritmo scelto:
   `./bin/pianificatore <percorso del file degli ordini> <percorso del file di pianificazione>`