ELABORATO ASSEMBLY-LABORATORIO ARCHITETTURA DEGLI ELABORATORI

Daniel Josue Prando (VR500813), Luca Grella (VR502889)

Specifiche:

Si sviluppi 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” con i seguenti requisiti:

- 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 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 realizzato in ritardo l’azienda dovrà pagare una penale pari al valore della priorità del prodotto completato in ritardo moltiplicato per il ritardo;
- L’utente deve scegliere tra due metodi di pianificazione della serie di prodotti:
  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.

Organizzazione dei file:

main.s:

sezione principale del codice che coordina le varie funzioni
del programma e calcola la penale totale.

file.s:

apertura, parsing, e chiusura del file dato come parametro.

menu.s:

permette all'utente di decidere l'algoritmo con il quale
ordinare la serie di prodotti, o eventualmente di uscire dal programma.

order.s:

ordinamento dell'array di prodotti in base all'algoritmo scelto.

print.s:

stampa dei risultati ottenuti o scrittura sul file di output.

error.s:

stampa dei messaggi d'errore.

Strutture dati:

- elementi in *number_array:*
  dato che nessun dato di un array può essere superiore a 127 possiamo utilizzare un byte per ogni campo, utilizzando 32 bit per contenere ogni dato di un prodotto, e utilizzanzo le operazioni di rotazione per specificare quale parametro usare

```
0x0       0x1       0x2       0x3
[ident...][ durata ][scadenza][priorità]
```

- elementi in *output_array:*
  contiene l'identificatore di un prodotto e l'intervallo di tempo in cui comincia la sua produzione. dato che *int_to_str* considera %ax come numero da aggiungere al *buffer* possiamo contenere in 32 bit entrambi i dati.

```
0x0       0x1       0x2       0x3
[         ][ident...][         ][ inizio ]
```

Funzioni:

main.s:

- *_start:*

  estrae i parametri dallo stack
  chiama *read_file* sul primo parametro salvando l'output in *number_array*
  chiama *check_bounds* per verificare che ogni valore appartenga al giusto range
  salva la quantità di prodotti nel file
  chiama *menu* e salva il codice dell'algoritmo da usare in *choice*
  chiama *order* per *number_array* utilizzando l'algoritmo specificato in *choice*
  per ogni prodotto:
  aggiunge il suo ID e tempo di inizio di produzione in *output_array*
  calcola la penale se necessario
  chiama *print_report* per *output_array*
  ripete dalla chiamata a *menu*

file.s:

- *read_file:*

  apertura e lettura del del file in *buffer*
  per ogni carattere:
  se il carattere è ',' o '\n' o EOF comincia la conversione del prossimo numero
  se è un carattere numerico lo utilizza per la conversione dl numero corrente
  se è un altro carattere esce dal programma con errore

- *check_bounds:*

  se un elemento di un array non rientra nei limiti previsti esce dal programma con errore

menu.s:

- *menu:*

  output del prompt
  input dei caratteri in *choice_buffer*
  se *choice_buffer* corrisponde a "1\n" o "2\n" restituisce rispettivamente 0 o 1
  esce dal programma per ogni altro caso.

order.s:

- *order:*
  ordina l'array in base all'algoritmo utilizzato

  se EDF:
  chiama *rotate_by* per avere il byte della priorità in %al e chiama *sort* e *reverse*
  chiama *rotate_by* per avere il byte della scadenza in %al e chiama *sort*
  chiama *rotate_by* per ripristinare la normale posizione degli elementi

  se HPF:
  chiama *rotate_by* per avere il byte della scadenza in %al e chiama *sort* e *reverse*
  chiama *rotate_by* per avere il byte della priorità in %al e chiama *sort* e *reverse*
  chiama *rotate_by* per ripristinare la normale posizione degli elementi

- *sort:*
  esegue bubble sort su un array di interi usando %al come chiave

- *reverse:*
  inverte gli elementi di un array

- *rotate_by:*
  esegue una rotazione di %cl bit per ogni intero in un array

print.s:

- *print_report:*
  carica nel *buffer* il testo usando combinazioni di *int_to_string* e *copy_bytes*
  testo da scrivere in stdout o, se specificato, nel file di output

- *int_to_str:*
  converte il numero in %ax in stringa e lo carica nel *buffer*

- *copy_bytes:*
  copia la stringa all'indirizzo in %eax nel *buffer*

Informazioni:

La lettura del file avviene in un buffer di 132 caratteri. Il motivo di questo numero è che essendo 10 la massima quantità di prodotti, ed essendo 13 la massima quantità di caratteri in una riga
(eg. “127,10,100,5\n”), si può assumere 130 come massima quantità di caratteri in un file. abbiamo aumentato arbitrariamente la lunghezza del buffer di 2 byte per poter capire quando il file è più lungo del dovuto.

Il metodo che utilizziamo per ordinare l’array è particolare in quanto utilizza gli stessi principi usati dall’algoritmo radix-sort, cioè di ordinare prima dalla cifra meno significativa e poi da quella più significativa, l’unica necessità è che l’algoritmo di ordinamento base utilizzato sia stabile. noi abbiamo usato bubble-sort per semplicità di implementazione ma qualsiasi algoritmo stabile può essere valido.

Utilizziamo bubble-sort per ordinare in ordine crescente l’array di prodotti, utilizzando due passaggi, e invertendo l’array se necessario. Avremmo potuto implementare una funzione che ordina in ordine decrescente ma sarebbe stato solo un copia e incolla e abbiamo preferito fare altrimenti ai fini del progetto.

Per selezionare con quale byte ordinare l’array utilizziamo dei bitshift che vengono applicati a ogni elemento dell’array.

Nella funzione che converte un numero in stringa e lo aggiunge al buffer utilizziamo il registro %ebx come mini buffer per contenere
i caratteri estratti per poi invertirli. La grandezza del registro è sufficiente in quanto il numero massimo che mai si potrà stampare è dovuto dal calcolo della penale, che non può superare

10 prodotti * 5 priorità * 100 ritardo = 5000.

La cui stringa può essere contenuta in 4 byte.

Il programma funziona sia con un parametri che con due parametri. Dove il primo specifica il file di input e il secondo, opzionale, quello di output.

Test:

Abbiamo testato il programma con diversi file di input per verificare la sua correttezza e la gestione degli errori, con opportuni stress test.

Gli errori che possono presentarsi sono i seguenti:

- Troppi parametri dati da linea di comando
- Nessun file specificato in linea di comando
- Quantità di numeri nel file non multipla di 4
- Caratteri sconosciuti nel file
- Quantità di numeri nel file superiore a 40
- Limiti dei dati non rispettati

Infine abbiamo verificato la correttezza dei calcoli con i seguenti file.

EDF.txt:

```
1,6,10,1
2,5,11,2
```

Both.txt:

```
9,1,100,4
4,1,100,4
6,1,100,1
7,1,100,2
127,10,100,5
8,1,100,3
2,1,100,2
1,1,100,1
5,1,100,5
3,1,100,3
```

None.txt:

```
1,10,7,4
2,6,8,5
3,10,20,3
4,10,25,2
```