filestat

Release 1.0.0

Francesco Pio Stelluti, Francesco Coppola

12 giu 2019

Indice

All'interno delle seguenti pagine sarà possibile trovare la documentazione generata per il progetto **filestat**, utility di sistema in grado di monitorare informazioni su file e directory, realizzato per il corso di *Laboratorio di Sistemi Operativi* dell'anno 2018/2019.

Lo sviluppo di tale codice è da attribuire interamente agli studenti **Francesco Pio Stelluti** e **Francesco Coppola**.

CAPITOLO 1

Introduzione

«Software application that is able to monitor a group of file taking information about them»

L'utility di sistema realizzata prende il nome di **filestat** e consente il monitoraggio avanzato di file e direcotry all'interno del sitema, forenendo dei report e delle statistiche riguardo a quest'utlimi.

1.1 Sinossi del programma e avvio

La sinossi del programma è:

```
filestat [options] [input] [output]
```

Dove:

- `input` è il file di input dove vengono definiti i parametri di esecuzione del programma, se omesso viene usato il file `filestat.in`;
- `output` è il file di output dove vengono collezionati i dati raccolti, se omesso viene usato il file `filestat.db`. Le informazioni presenti nel file di output vengono *aggiornate* ad ogni esecuzione del programma **(e non soprascritte)**.

Le possibili opzioni sono:

```
--verbose|-v
--stat|-s
--report|-r
--history|-h <filepah>
--user|-u <userId>
--group|-g <groupId>
--length|-l <min>:<max>
--noscan
```

La descrizione è la seguente:

- `--verbose|-v`: durante l'esecuzione il programma mostra a video le informazioni sui file elaborati, ed i dati raccolti;
- `--stat|-s`: vengono mostrate sullo standard output le seguenti statistiche:
  - numero di file monitorati;
  - numero di link;
  - numero di directory;
  - dimensione totale;
  - dimensione media;
  - dimensione massima;
  - dimensione minima (in byte).
- `--report|-r`: al termine dell'esecuzione vengono mostrati sullo standard output le informazioni riguardanti numero di file elaborati, tempo di elaborazione, dimensione massima del file;
- `--history|-h <filepah>`: stampa sullo standard output la cronologia delle informazioni riguardanti il file `<filepah>`;
- `--user|-u <userId>`: stampa sullo standard output le informazioni di tutti i file di proprietà di `<userId>`
- `--group|-g <groupId>`: stampa sullo standard output le informazioni di tutti i file di proprietà di `<groupId>`
- `--length|-l <min>:<max>`: stampa sullo schermo le informazioni di tutti i file di dimensione (in byte) compresa tra `<min>` e `<max>` (`:<max>` indica ogni file di dimensione al più `<max>`, `<min>:` e `<min>` indicano ogni file di dimensione almeno `<min>`)
- `--noscan`: se presente questa opzione non viene effettuata la raccolta dei dati, ma vengono presentati solo le informazioni presenti del file di output.

1.2 Formato del file di input

I parametri di esecuzione di un programma vengono definiti in un file di testo costituito da una sequenza di righe della seguente forma:

```
<path> [r] [l]
```

Dove `r` indica che occorre leggere ricorsivamente i file nelle directory sottostanti (applicando le stesse opzioni) mentre `l` indica che i link devono essere trattati come file/directory regolari, in questo caso le informazioni collezionate fanno riferimento al file riferito dal link e non a link stesso.

**Per poter specificare pathname che presentano spazi al loro interno è necessario far precedere tali spazi dal carattere **

1.3 Formato del file di output

I dati raccolti vengono salvati usando il seguente formato:

```
# <path1>
<data1> <uid1> <gid1> <dim1> <perm1> <acc1> <change1> <mod1> <nlink1>
...
<data_n> <uid_n> <gid_n> <dim_n> <perm_n> <acc_n> <change_n> <mod_n> <nlink_n>
###
# <path2>
<data1> <uid1> <gid1> <dim1> <perm1> <acc1> <change1> <mod1> <nlink1>
...
<data_n> <uid_n> <gid_n> <dim_n> <perm_n> <acc_n> <change_n> <mod_n> <nlink_n>
###
...
# <pathm>
<data1> <uid1> <gid1> <dim1> <perm1> <acc1> <change1> <mod1> <nlink1>
...
<data_n> <uid_n> <gid_n> <dim_n> <perm_n> <acc_n> <change_n> <mod_n> <nlink_n>
###
###
```

Le informazioni associate al file/directory `<path>` iniziano con la riga:

```
# <path>
```

Successivamente si trovano una sequenza di righe (una per ogni analisi svolta) della forma:

```
<data> <uid> <gid> <dim> <perm> <acc> <change> <mod> <nlink>
```

Dove:

```
<data> indica ora-data in cui sono recuperate le informazioni;
<uid> è l'id dell'utente proprietario del file;
<gid> è l'id del gruppo del file;
<perm> è la stringa con i diritti di accesso al file;
<acc> data dell'ultimo accesso;
<change> data dell'ultimo cambiamento;
<mod> data dell'ultima modifica dei permessi;
<nlink> numero di link verso il file.
```

Le informazioni terminano con la riga:

```
###
```

Il file di output termina con una riga:

```
###
```

CAPITOLO 2

Realizzazione del progetto

La realizzazione del codice prodotto ha seguito uno standard **preciso** ed **efficente** che ha reso lo sviluppo di quest'ultimo **flessibile** ed **elegante** ai fini di aver un'utility di sistema *altamente performante* grazie alle potenzialità offerte dal `C` stesso.

2.1 Struttura e architettura del codice sviluppato

La struttura del progetto si articola fondamentalmente su 5 file sorgente di estensione `.c`, a cui seguono altrettanti file di estensione `.h`, in cui vengono dichiari i metodi da *estendere*:

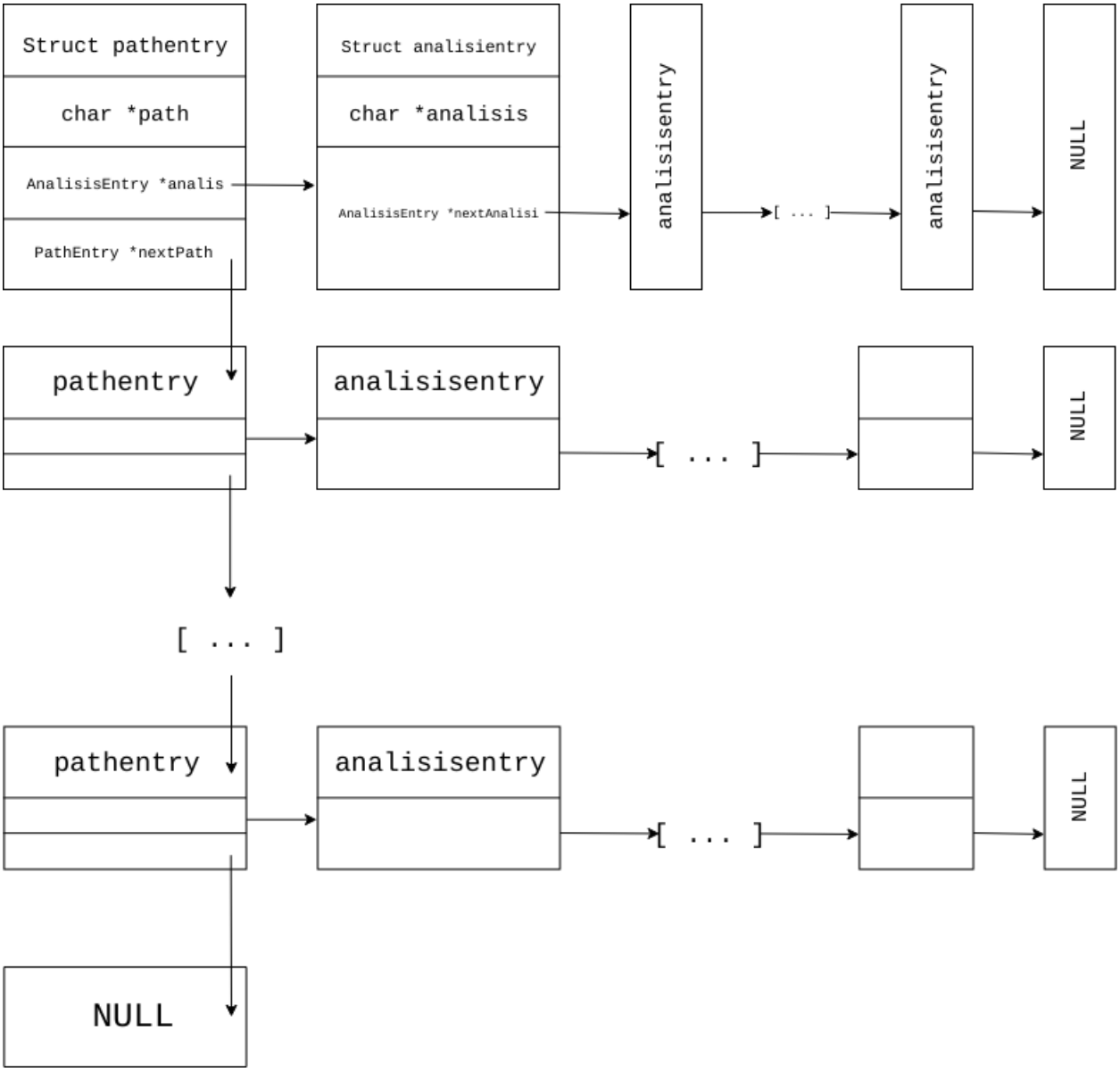
- `main.c`: contiene il codice di avvio del progetto. Consente il **parse** delle opzioni e la corretta apertura dei file di `input` e dei file di `output`.
- `datastructure.c`: contiene il codice necessario alla gestione della **struttura dati** impiegata nel progetto per la collezione dei dati relativi ai **file monitorati**.
- `scan.c`: contiene il codice necessario all'inizializzazione della **struttura** dati tramite la lettura delle informazioni specificate tramite i file di input e output.
- `inputscan.c`: contiene il codice finalizzato all'analisi del file di `input` e dei file i cui **pathname** sono specificati nel file di `input`. Aggiorna di conseguenza la **struttura dati** con le informazioni relative ai **file monitorati**.
- `output.c`: contiene il codice finalizzato all'analisi del file di `output` per poter inserire le informazioni contenute al suo interno nella **struttura dati** specificata in precedenza.

Per l'analisi dei singoli file sorgenti si rimanda alle sezioni dedicate alla spiegazione dei singoli metodi specificati al loro interno.

Le librerie impiegate all'interno del codice sono:

- `Libreria standard di C`: sono stati usati gli header `<stdio.h>`, `<string.h>`, `<stdlib.h>`, `<errno.h>`, `<time.h>`.
- `Libreria POSIX C`: sono stati usati gli header `<unistd.h>`, `<sys/stat.h>`, `<limits.h>`, `<pwd.h>`, `<grp.h>`, `<dirent.h>`, `<getopt.h>`.

2.2 Struttura dati implementata



La struttura dati alla base del funzionamento del progetto è stata definita tramite gli struct `pathentry` e `analisisentry`, entrambi definiti in `datastructure.h` e associati ai tipi `PathEntry` e `AnalisisEntry` definiti tramite il costrutto `typedef`. Lo struct `pathentry` è indirizzato alla definizione di una lista in cui ogni elemento contiene una stringa, associata ad un pathname, il puntatore ad un elemento `AnalisisEntry` ed il puntatore ad un elemento `PathEntry`, l'elemento successivo all'interno della lista. Analogamente lo struct `analisisentry` è puntato alla definizione di una lista i cui elementi contentono una stringa associata alle informazioni relative all'analisi di un file ed il puntatore ad un elemento `AnalisisEntry`, elemento successivo nella lista.

Elementi `PathEntry` e `AnalisisEntry` che non contengono informazioni sono associati al puntatore **NULL**. Le funzionalità incluse all'interno del file `datastructure.h` permettono di ottenere puntatori ad elementi vuoti di `PathEntry` e `AnalisisEntry`, di aggiungere ad una lista di `PathEntry` nuovi elementi tramite il passaggio di stringhe contenenti pathname e le informazioni derivate dall'analisi dei file associati a tali pathname, di verificare che una lista di `PathEntry` o di `AnalisisEntry` risulti vuota, di ottenere gli elementi successivi all'interno di una

lista `PathEntry` o `AnalisisEntry` dato il puntatore ad un elemento delle due liste e di ottenere il riferimento al primo elemento della lista di `AnalisisEntry` associata ad un dato elemento di una lista di `PathEntry`.

Abbiamo impiegato due istanze della struttura dati definita da `PathEntry`, una per collezionare le informazioni presenti all'interno del file di output, utili ad esempio per la gestione dell'opzione `-h/--history`, ed una per raccogliere le informazioni derivate dall'analisi dei file i cui pathname sono stati definiti all'interno del file di input. Una volta effettuato il corretto utilizzo delle due strutture dati si procede quindi al loro **merge** per la successiva scrittura delle nuove informazioni raccolte sul file di output specificato.

Per ulteriori informazioni si rimanda alla sezione dedicata a `datastructure.c`, in cui sono presenti le definizioni dei metodi dichiarati in `datastructure.h` e in `datastructure.c`.

Essendo la struttura dati alla base del progetto basata su due implementazioni di una lista la sua complessità risulta essere:

- `O(P)` quando è necessario aggiungere o recuperare le informazioni da uno specifico elemento `PathEntry`.
- `O(P x A)` quando è necessario aggiungere o recuperare le informazioni da uno specifico elemento `AnalisisEntry`.

Dove `P` rappresenta il numero di elementi `PathEntry` presenti nella struttura dati e `A` il numero massimo di elementi `AnalisisEntry` associati ad un `PathEntry`.

2.3 Implementazione delle funzionalità richieste

L'esecuzione del programma porta ad un aggiornamento complessivo delle informazioni contenute all'interno del file output, aggiungendo `pathname` se non già presenti e analisi delle informazioni relative ai file referenziati dai `pathname` presenti e aggiunti. Al termine dell'esecuzione il file di `output` risulterà pertanto aggiornato e non interamente sovrascritto. I pathname aggiunti al file di output gestito dal programma sono tutti assoluti per permettere la portabilità di tale file.

Per la gestione delle decisioni dell'utente circa l'uso di file di `input` e di `output` che non siano quelli di default si è deciso di assicurare le funzionalità del programma solo in presenza o in assenza dei `pathname` associati ad **entrambi** i file. L'inclusione di un singolo `pathname` all'interno della sinossi di avvio del programma porterà all'avvio del programma con l'impiego dei file di `input` e di `output` di default.

L'uso dell'opzione `-v/--verbose` porterà alla stampa sullo **standard output** di informazioni circa i file analizzati, quali il loro `pathname` relativo (che diventa assoluto nel caso il file analizzato sia un file referenziato da un link), l'eventuale natura di link o di directory e la corretta riuscita dell'operazione di analisi delle informazioni.

L'implementazione delle opzioni `-s/--stat` e `-r/--report` risulta la medesima e consiste nella stampa sullo **standard output** delle informazioni richieste alla fine dell'elaborazione generale. L'uso dell'opzione `-h/--history`, seguita dal pathname del file di cui si vuole ottenere la cronologia, porterà alla stampa sullo **standard output** delle informazioni relative al file associato a tale pathname. Se il pathname non è presente all'interno del file di output gestito dal programma verrà effettuata una notifica sullo **standard output** di tale mancanza. Non sono ovviamente incluse le informazioni aggiunte tramite l'esecuzione del programma in corso.

L'implementazione di `-u/--user` e `-g/--group` consiste in un **filtro** effettivo su quelli che sono i file da monitorare e di cui aggiungere informazioni nel file di `output`. L'inclusione di un file all'interno dell'operazione di analisi effettuata dal programma, *in presenza di tali opzioni*, porta alla stampa sullo **standard output** del pathname assoluto del file incluso nell'analisi e delle relative informazioni collezionate. Il medesimo discorso si applica anche all'implementazione di `-l/--length`.

Infine, per l'implementazione di `--noscan` si è deciso di effettuare comunque l'operazione di popolamento della struttura dati con le informazioni derivate dal file di output evitando ogni tipo di operazione di analisi su ulteriori file, come quelli specificati dai pathname presenti nel file di input con conseguente popolamento della struttura dati dedicata, e portando alla stampa sullo **standard output** delle informazioni presenti all'interno della struttura dati riferita al file di output al termine delle operazioni del programma.

2.4 Makefile

> Il make è un'utility, sviluppata sui sistemi operativi della famiglia UNIX, ma disponibile su un'ampia gamma di sistemi, che automatizza il processo di creazione di file che dipendono da altri file, risolvendo le dipendenze e invocando programmi esterni per il lavoro necessario.

Tale utility nel nostro caso è stata utilizzata per la compilazione di **codice sorgente** in **codice oggetto**, unendo e poi linkando il codice oggetto in un programma eseguibile chiamato `filestat`.

Essa usa file chiamati `makefile` per determinare il grado delle dipendenze per un particolare output, e gli script necessari per la compilazione da passare alla shell.

I *task* che mette a disposizione sono i seguenti:

- `make filestat`: converte il codice sorgente realizzato, *con le librerie a lui annesse*, in un codice oggetto eseguibile lanciando il comando `./filestat`
- `make clean`: elimina il contenuto delle directory indicate al suo interno per ottenere sempre un ambiente di lavoro pulito e privo di file obsoleti
- `make test`: generazione della cartella principale `folder_testing` in grado di dare all'utente **la possibilità** di testare il corretto funzionamento dell'utility `filestat`

2.5 Test relativi al corretto funzionamento

Per avere una stima rispetto al corretto funzionamento del codice sono stati effettuati, in primo luogo, dei test molto *spartani* mediante i comandi `ls -l`, `du -sh file_path` e utilizzando l'"*explorer* di sistema fornito dall'OS.

Quest'ultimi ci restituivano infatti le informazioni **corrette** rispetto ai dati che analizzavamo, e in maniera banale, li confrontavamo con quelli che l'utility produceva. Una volta confermato il corretto funzionamento dell'utility si è deciso quindi di produrre una script per `bash` che fosse in grado di generare in maniera del tutto casuale file, link e directory, che a loro volta contenevano altrettanti elementi, per testare in maniera definitiva l'utility stessa e dimostrare in maniera oggettiva il suo funzionamento.

Da questa premessa nasce infatti `folder_testing.sh`.

Lo script in questione, disponibile all'interno della main direcotry del progetto, attinge a risorse di sistema localizzate in `/dev/urandom` per produrre dei contenuti di natura **random** relativi ai nomi dei file e delle directory e per popolare il loro contenuto.

L'esecuzione di tale script quindi genera una nuova direcotry `folder_testing` al cui interno sarà possibile trovare i file - *e le direcotry* - nati da tale generazione.

Per avviare tale processo sarà necessario lanciare il comando:

```
make test
```

Infatti all'interno del **Makefile** di cui si è parlato nella sezione relativa a codesto argomento è possibile reperire tale informazione.

È interessante poi vedere come l'implementazione e il lancio di tale script produca subito un risultato tangibile che attesti il numero di file e directory generate, così come il numero di link presenti e in particolar modo la somma complessiva del peso di tali file.

Di seguito è possibile apprezzare la bontà e la comodità di tale `script`:

```
./folder_testing
├── [        4096]  ICcJo
│   ├── [        4096]  2ymehX
│   │   └── [        4096]  QP
│   │       ├── [        4026]  8qR1g46s.bin
│   │       ├── [       16086]  bdkx0.bin
│   │       └── [        3837]  ezK6fiUW3dAR.bin
│   ├── [        4096]  87
│   │   ├── [        4096]  QP
│   │   ├── [        2109]  tdzY.bin
│   │   └── [       16310]  YpdiX.bin
│   └── [        4096]  JmqQ
│       └── [        4096]  QP
│           ├── [        3652]  Prdg0.bin
│           └── [         861]  yMutQoBsI.bin
├── [        4096]  Si1
│   ├── [        4096]  Aw
│   │   ├── [        4096]  0BxaN
│   │   │   ├── [        6707]  nq.bin
│   │   │   └── [       11253]  pWIsm.bin
│   │   ├── [        4096]  aG
│   │   │   └── [        9555]  ZbioiWDOw.bin
│   │   ├── [        4096]  h3n1
│   │   │   └── [        2694]  8m.bin
│   │   ├── [        4096]  rFes
│   │   │   ├── [       19273]  RRP.bin
│   │   │   └── [        8035]  YpaCzp.bin
│   │   └── [         677]  QT0Dwlb3.bin
│   ├── [        4096]  fkXD
│   │   ├── [        4096]  0BxaN
│   │   │   └── [        2503]  aIqEChA.bin
│   │   ├── [        4096]  aG
│   │   │   └── [        1507]  G5MF0.bin
│   │   ├── [        4096]  h3n1
│   │   │   └── [        3017]  1YF3kYej9P.bin
│   │   ├── [        4096]  rFes
│   │   ├── [        6573]  07b.bin
│   │   └── [        3764]  qBbF.bin
│   ├── [        4096]  l jPwu3
│   │   ├── [        4096]  0BxaN
│   │   │   ├── [        9030]  h5FDXIsn.bin
│   │   │   └── [        2658]  XomM4.bin
│   │   ├── [        4096]  aG
│   │   │   ├── [        2980]  QUWYJY.bin
│   │   │   └── [       10209]  RgziPE7jj.bin
│   │   ├── [        4096]  h3n1
│   │   │   └── [        3405]  xKFZ6j.bin
│   │   ├── [        4096]  rFes
│   │   │   ├── [       11185]  17s.bin
│   │   │   └── [        6144]  uJW1U.bin
│   │   ├── [        8663]  Df8rs.bin
│   │   └── [       12494]  Vqo8R.bin
│   ├── [        4096]  M F
│   │   ├── [        4096]  0BxaN
│   │   │   └── [       14186]  rCt35X.bin
│   │   ├── [        4096]  aG
│   │   │   └── [         359]  j0Bl7jx.bin
```

(continues on next page)

(continua dalla pagina precedente)

```
│   │   ├── [       4096]  h3n1
│   │   │   ├── [      17973]  SE4XX8ExlK.bin
│   │   │   └── [       4964]  sfiq0Q.bin
│   │   └── [       4096]  rFes
│   │       └── [      13436]  P9hWuujV.bin
│   ├── [       1663]  G6G0n8W7.bin
│   └── [      10078]  RjpHS.bin
├── [          7]  link_0 -> Mru.bin
└── [       5795]  Mru.bin

28 directories, 37 files
Dimensione totale dei file: 376452      ./folder_testing
```

Dopo aver lanciato tale comando infatti basterà modificare il percorso da analizzare all'interno del file di input fornito per poi confrontarle con quelle restituire dall'utility prodotta.

CAPITOLO 3

Sphinx e le sue potenzialità

L'intera documentazione generata della quale si sta usufruendo è frutto dell'unione tra **Sphinx** e **hawkmoth**, due strumenti dedicati alla generazione di testi a partire da del mero e puro codice.

3.1 Strumenti con cui è stata realizzata

Solitamente quando si affronta il tema della generazione di una documentazione per un progetto di natura C si incappa in una serie di difficoltà dettate dall'assenza di uno standard noto per la stesura dei commenti relativi ai metodi e alle variabili presenti nel codice stesso.

Per soccombere a tale problematiche si è deciso di usufruire di un'estensione per **Sphinx**[1] che fosse in grado di convertire uno standard noto in una sintassi facilmente interpolabile da Sphinx stesso per ottenere una documentazione *elegante* e *facilmente usufruibile*.

Tale estensione è chiamata **hawkmoth**, software realizzato in **Pyhton** il quale si basa su un progetto **Open Source** disponibile su GitHub.

3.2 Autogenerazione della sintassi convertita da C a Sphinx

Una volta installata la dipendenza mediante `pip`, attraverso il seguente comando:

```
$ sudo pip install hawkmoth
```

Sarà possibile integrarla nel file `conf.py` e adattare la propria documentazione nel codice sorgente seguendo un noto tipo di standard.

Quest'ultimi sono molteplici e si possono facilmente reperire all'interno delle specifiche del progetto online.

[1] *Software Open Source per l'autogenerazione di documentazioni a partire da un codice sorgente generico*

Vi è per esempio la possibilità di scrivere commenti in `JavaDoc`[2] per un codice che in realtà col Java a ben poco in comune.

[2] *Javadoc è un applicativo incluso nel Java Development Kit della Sun Microsystems, utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java*

CAPITOLO 4

Documentazione

Nei seguenti paragrafi sarà possibile accedere alla **documentazione** prodotta per l'utility di sistema **filestat**.

4.1 main.c

OptInfo **options**
: Struct in cui viene eseguito lo storage delle possibili opzioni.

FILE * **file_input**
: Puntatori di natura FILE in cui vengono settati i file di I/O che verranno utilizzati. Quest'ultimi potranno essere scelti dall'utente o impostati di default.

int **main** (int *argc*, char ** *argv*)
: Main necessario all'avvio del programma.

void **parsePaths** (int *argc*, char ** *argv*)
: Apertura dei file di input e di output basandosi sulle scelte dell'utente.

  Parametri

  - **argc** – parametro passato come argomento a main
  - **argv** – parametro passato come argomento a main

void **printOpt** ()
: Metodo necessario al debug per la visualizzazione dello struct delle opzioni.

int **parseOpt** (int *argc*, char ** *argv*)
: Viene effettuata la lettura delle opzioni inserite da linea di comando.

  Parametri

  - **argc** – parametro passato come argomento a main
  - **argv** – parametro passato come argomento a main

  Ritorna 0 in caso di errore, non-zero in caso di successo

int **getLengthArg**(char * *arg*)
Metodo ausiliario all'impostazione del flag `-l/--length`.

Parametri

- **arg** – puntatore ad un array di caratteri contenente i valori delle lunghezze

Ritorna 1 in caso di successo

int **getHistoryPath**(char * *arg*)
Metodo che restituisce la cronologia dei path analizzati grazie all'opzione `-h/--history`.

Parametri

- **arg** – puntatore ad un array di caratteri contenente il path che si desidera analizzare

Ritorna 1 in caso di successo

4.2 scan.c

ScanInfo **stats**
Struct di riferimento per le informazioni complessive raccolte sui file, quali:

- numero di file monitorati;
- numero di link incontrati;
- numero di directory incontrate;
- dimensione totale dei file;
- dimensione media dei file;
- dimensione massima dei file;
- dimensione minima (in byte) dei file.

int **startScan**(FILE * *input*, FILE * *output*)
Avvio delle operazioni di scan e di recupero delle informazioni richieste dal programma.

Parametri

- **input** – puntatore al file di input trattato nel programma
- **output** – puntatore al file di output trattato nel programma

Ritorna 1 in caso di successo

void **increaseMonitorati**()
Incremento numero di file monitorati e di cui è stata aggiunta la relativa analisi.

void **increaseLink**()
Incremento numero di link incontrati.

void **increaseDirectory**()
Incremento numero di directory.

void **updateDimMedia**()
Aggiornamento valore dimensione media dei file monitorati.

void **updateDimMax**(int *data*)
Aggiornamento valore dimensione massima dei file monitorati.

Parametri

- **data** – valore della dimensione che si desidera aggiungere

void **updateDimMin**(int *data*)
Aggiornamento valore dimensione minima dei file monitorati.

Parametri

- **data** – valore della dimensione che si desidera aggiungere

void **increaseDimTotale**(int *data*)
Aggiornamento valore dimensione totale dei file monitorati.

Parametri

- **data** – valore della dimensione che si desidera aggiungere

void **cleanFile**(FILE * *target*)
Reset completo di un file.

Parametri

- **target** – puntatore al file si cui si desidera effettuare i reset

void **printOnOutput**(FILE * *target*, PathEntry * *entry*)
Gestione completa della stampa delle informazioni contenute in una struttura PathEntry su un file di output.

Parametri

- **target** – puntatore al file di output su cui si vogliono inserire le informazioni
- **entry** – puntatore alla struttura dati da cui si vogliono leggere le informazioni da stampare

void **updateStats**(long *size*)
Aggiornamento completo di tutte le informazioni riguardo i file che sono stati monitorati.

Parametri

- **size** – valore della dimensione che si desidera aggiungere

void **printStats**()
Metodo che stampa a video il report completo delle informazioni acquisite.

void **printHistory**(PathEntry * *entry*, char * *path*)
Stampa sullo standard output delle informazioni relative alle analisi passate effettuate su un file.

Parametri

- **entry** – puntatore alla struttura dati da cui leggere le informazioni
- **path** – puntatore all'array di caratteri contenente il associato al file di cui si vuole conoscere la cronologia

void **printOnFile**(PathEntry * *pathentry*, FILE * *file*)
Stampa completa delle informazioni presenti in una struttura dati PathEntry su di un file.

Parametri

- **pathentry** – puntatore all'entry associato al file di cui si vogliono stampare le informazioni sullo standard output
- **file** – puntatore al il file su cui si vogliono inserire le informazioni

void **freePath**(PathEntry * *entry*)
Gestione completa delle operazioni di rilascio delle risorse da operare su una struttura dati PathEntry.

Parametri

- **entry** – puntatore struttura dati su cui effetturare l'operazione completa di rilascio delle risorse

void **freeAnalisis** (AnalisisEntry * *entry*)
Gestione completa delle operazioni di rilascio delle risorse da operare su una struttura dati AnalisisEntry.

Parametri

- **entry** – puntatore alla struttura dati su cui effetturare l'operazione completa di rilascio delle risorse

PathEntry * **merge** (PathEntry * *out*, PathEntry * *in*)
Operazione di merge tra due strutture dati PathEntry.

Parametri

- **out** – puntatore a struttura PathEntry in cui collezionare le informazioni relative al merge
- **in** – puntatore a struttura PathEntry da cui prendere le informazioni da impiegare dal merge

Ritorna puntatore a struttura PathEntry in cui sono collezionate le informazioni risultate dal merge

4.3 inputscan.c

PathEntry * **readInputFile** (FILE * *input*, PathEntry * *entry*)
Gestione completa delle analisi delle informazioni presenti sul file di input trattato dal programma.

Parametri

- **input** – puntatore al file di input da cui leggere le informazioni sulle analisi passate
- **entry** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate grazie al file di input

Ritorna puntatore alla struttura dati PathEntry aggiornata

PathEntry * **inputLineAnalisis** (char * *riga*, PathEntry * *entry*)
Metodo che legge interamente le linee presenti all'interno del file di input.

Parametri

- **riga** – puntatore all'array di caratteri contenente la riga del file desiderata di cui si vuole effettuare la lettura
- **entry** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate grazie all'analisi della riga

Ritorna puntatore alla struttura dati PathEntry aggiornata

PathEntry * **scanFilePath** (char * *path*, int *isR*, int *isL*, PathEntry * *entry*)
Analisi completa di un singolo file.

Parametri

- **path** – puntatore all'array di caratteri contenente path del file da analizzare
- **isR** – flag per l'analisi ricorsiva del file in caso sia una directory
- **isL** – flag utile in caso il file analizzato sia un link, di cui vengono analizzate le informazioni senza far riferimento al file referenziato
- **entry** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate grazie all'analisi del file

Ritorna puntatore alla struttura dati PathEntry aggiornata

PathEntry * **directoryAnalisis** (struct stat * *dirStat*, PathEntry * *entry*, int *isR*, int *isL*, char * *path*)
Analisi delle entry presenti all'interno di un file directory.

Parametri

- **dirStat** – puntatore all'istanza di struct stat in cui sono contenute le informazioni del file da analizzare
- **entry** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate grazie all'analisi del file
- **isR** – flag da passare all'analisi dei file referenziati dalle entry del file directory argomento
- **isL** – flag da passare all'analisi dei file referenziati dalle entry del file directory argomento
- **path** – puntatore all'array di caratteri contenente il pathname associato a dirStat

Ritorna puntatore alla struttura dati PathEntry aggiornata

PathEntry * **addFileAnalisis** (struct stat * *currentStat*, char * *path*, PathEntry * *entry*)
Aggiunta delle informazioni associate all'analisi di un file ad una struttura dati PathEntry.

Parametri

- **currentStat** – puntatore all'istanza di struct stat in cui sono contenute le informazioni del file da analizzare
- **path** – puntatore all'array di caratteri associato al file da analizzare
- **entry** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate grazie all'analisi del file

Ritorna puntatore alla struttura dati PathEntry aggiornata

int **checkLength** (struct stat * *file*)
Verifica che un file rispetti le condizioni espresse con il flag `-l/--length`.

Parametri

- **file** – puntatore al file di cui verificare le condizioni

Ritorna 0 in caso di esito negativo della verifica, 1 in caso di esito positivo della verifica

int **checkUID** (struct stat * *file*)
Verifica che un file rispetti le condizioni espresse con il flag `-u/--user`.

Parametri

- **file** – puntatore file di cui verificare le condizioni

Ritorna 0 in caso di esito negativo della verifica, 1 in caso di esito positivo della verifica

int **checkGID** (struct stat * *file*)
Verifica che un file rispetti le condizioni espresse con il flag `-g/--group`.

Parametri

- **file** – puntatore al file di cui verificare le condizioni

Ritorna 0 in caso di esito negativo della verifica, 1 in caso di esito positivo della verifica

int **checkOptions** (struct stat * *file*)
Verifica che un file rispetti le condizioni espresse con i flag `-l/--length`, `-u/--user` e `-g/--group`.

Parametri

- **file** – puntatore al file di cui verificare le condizioni

Ritorna 0 in caso di esito negativo della verifica, 1 in caso di esito positivo della verifica

char * **findLastOf** (char * *str*, char *c*)
Permette di ottenere il puntatore all'ultima occorrenza di un carattere presente in una stringa. Metodo ausiliario a `getLinkAbsPath(char*)`.

Parametri

- **str** – puntetore all'array di caratteri di cui effettuare l'analisi
- **c** – carattere di cui si vuole ottenere il puntatore all'ultima occorrenza

Ritorna puntatore all'ultima occorrenza all'interno della stringa passata come argomento del carattere passato come argomento

char * **getLinkAbsPath** (char * *path*)
Metodo necessario al get del pathname assoluto di un link.

Parametri

- **path** – puntatore all'array di caratteri contentente il filename del link di cui si vuole ottenere il percorso assoluto

Ritorna la stringa contenente il percorso assoluto del link passato come parametro locale

4.4 outputscan.c

PathEntry * **readOutputFile** (FILE * *output*, PathEntry * *data*)
Gestione completa delle analisi delle informazioni presenti sul file di output trattato dal programma.

Parametri

- **output** – puntatore al file di output da cui leggere le informazioni sulle analisi passate
- **data** – puntatore alla struttura dati PathEntry in cui inserire le informazioni recuperate dal file di output

Ritorna puntatore alla struttura dati PathEntry aggiornata

4.5 datastructure.c

AnalisisEntry * **emptyAnalisis** ()
Ottenimento di un elemento vuoto della struttura dati AnalisisEntry.

Ritorna puntatore ad un'istanza vuota di AnalisisEntry

PathEntry * **emptyPath** ()
Ottenimento di un elemento vuoto della struttura dati PathEntry.

Ritorna puntatore ad un'istanza vuota di PathEntry

AnalisisEntry * **createNewAnalisis** (char * *an*)
Creazione di un nuovo elemento della struttura dati AnalisisEntry.

Parametri

- **an** – puntatore all'array di caratteri contenente la stringa contentente l'analisi da inserire all'interno dell'elemento

Ritorna puntatore ad un nuovo elemento di AnalisisEntry

PathEntry * **createNewPath** (char * *pt*)
Creazione di un nuovo elemento della struttura dati PathEntry.

Parametri

- **pt** – puntatore all'array di caratteri contenente il path da inserire all'interno dell'elemento

Ritorna puntatore ad un nuovo elemento di PathEntry

int **isAnalisisEmpty** (AnalisisEntry * *entry*)
Verifica che un puntatore ad AnalisisEntry corrisponda ad un elemento vuoto.

Parametri

- **entry** – puntatore all'elemento AnalisisENtry da analizzare

Ritorna 0 in caso di errore, non-zero in caso di successo

int **isPathEmpty** (PathEntry * *entry*)
Verifica che un puntatore ad PathEntry corrisponda ad un elemento vuoto.

Parametri

- **entry** – puntatore all'elemento PathEntry da analizzare

Ritorna 0 in caso di errore, non-zero in caso di successo

PathEntry * **addPath** (PathEntry * *entry*, char * *newpath*)
Aggiunta di un nuovo path all'interno di una struttura dati PathEntry. Metodo ausiliario ad addPathAndAnalisis(PathEntry*, char*, char*).

Parametri

- **entry** – puntatore alla struttura dati PathEntry da aggiornare
- **newpath** – puntatore all'array di caratteri contenente il path da inserire

Ritorna puntatore alla struttura PathEntry aggiornata

AnalisisEntry * **addAnalisisByAnalisisEntry** (AnalisisEntry * *entry*, char * *newanalisis*)
Aggiunta di una nuova analisi all'interno di una struttura AnalisisEntry. Metodo ausiliario ad addAnalisis(PathEntry*, char*, char*).

Parametri

- **entry** – puntatore alla struttura dati AnalisisEntry da aggiornare
- **newanalisis** – puntatore all'array di caratteri contenente l'analisi da inserire

Ritorna puntatore alla struttura AnalisisEntry aggiornata

PathEntry * **addAnalisis** (PathEntry * *pathentry*, char * *path*, char * *newanalisis*)
Aggiunta di una nuova analisi all'interno di una struttura dati PathEntry dato il path a cui corrisponde. Metodo ausiliario ad addPathAndAnalisis(PathEntry*, char*, char*).

Parametri

- **pathentry** – puntatore alla struttura dati PathEntry da aggiornare
- **path** – puntatore all'array di caratteri contenente il path associato all'analisi da inserire
- **newanalisis** – ountatore alla stringa contenente l'analisi da inserire

Ritorna puntatore alla struttura PathEntry aggiornata

PathEntry * **addPathAndAnalisis** (PathEntry * *entry*, char * *path*, char * *analisis*)
Aggiunta di una nuova analisi all'interno di una struttura dati PathEntry dato il path a cui corrisponde.

Parametri

- **pathentry** – puntatore alla struttura dati PathEntry da aggiornare
- **path** – puntatore all'array di caratteri contenente il path associato all'analisi da inserire
- **newanalisis** – stringa contenente l'analisi da inserire

Ritorna puntatore alla struttura PathEntry aggiornata

PathEntry * **getNextPath** (PathEntry * *entry*)
Metodo che restituisce l'elemento contentente il path successivo a quello contenuto in un elemento di PathEntry.

Parametri

- **entry** – puntatore al PathEntry di cui si vuole ottenere il successivo

Ritorna puntatore alla struttura PathEntry richiesta

AnalisisEntry * **getFirstAnalisis** (PathEntry * *entry*)
Metodo che restituisce la prima analisi all'interno della struttura associata al path passato come argomento.

Parametri

- **entry** – puntatore ll PathEntry di cui si vuole analizzare la prima analisi associata

Ritorna puntatore alla struttura AnalisisEntry richiesta

AnalisisEntry * **getNextAnalisis** (AnalisisEntry * *entry*)
Metodo che restituisce l'elemento contentente l'analisi successiva a quella contenuto in un elemento di AnalisisEntry.

Parametri

- **entry** – puntatore all'elemento AnalisisEntry di cui si vuole ottenere il successivo

Ritorna puntatore all'elemento AnalisisEntry richiesta

PathEntry * **getPathEntry** (PathEntry * *entry*, char * *path*)
Metodo per ottenere il puntatore all'elemento della struttura dati PathEntry contenente un determinato path.

Parametri

- **entry** – puntatore alla struttura dati in cui effettuare la ricerca
- **char** – stringa contenente il pathname di cui si vuole ottenere il relativo elemento di PathEntry

Ritorna puntatore alla struttura PathEntry richiesta

int **containsPath** (PathEntry * *entry*, char * *path*)
Verifica che una struttura dati PathEntry contenga al suo interno un elemento con un path specificato.

Parametri

- **entry** – puntatore alla struttura dati in cui effettuare la ricerca
- **char** – stringa contenente il pathname di cui si vuole verificare la presenza in un elemento PathEntry

Ritorna 1 se il path è presente in un elemento PathEntry, 0 altrimenti

Indice