Università degli Studi di Padova

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

psort: sistema di documentazione

RELATORE
Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE
Dott. Marco Bressan

TESI DI LAUREA DI
Pietro Vallese
Matr. N. 575874

Anno Accademico 2010/2011

Indice

Prefazione

Questo elaborato, che andiamo ad introdurre, ha come argomento principale la documentazione di un software, con delle applicazioni pratiche al progetto *psort*.

Ogni capitolo è di fatto autoconclusivo e separato dagli altri. Riassumiamo qui di seguito i vari contenuti, nell'eventualità che il lettore sia interessato solo ad alcuni di essi e voglia saltare il resto.

Nel capitolo 1 presentiamo un'analisi molto generale della documentazione e dei vari formati in cui essa può essere realizzata.

Nel capitolo 2, invece, presentiamo molto brevemente il progetto *psort*.

Nel capitolo 3 abbiamo condensato una piccola guida all'utilizzo del software Doxygen. È nostra opinione che la documentazione prodotta da questo programma possa costituire un manuale di riferimento molto utile per *psort* (e anche per altri software) nel caso in cui l'utente sia interessato più al suo utilizzo come libreria.

Nel capitolo 4 esponiamo sinteticamente cosa sia una pagina man, ed a seguire un breve compendio su come crearne una.

Nei capitolo 5 e 6 riportiamo per intero la pagina man che abbiamo realizzato per il programma *psort*.

Nel capitolo 7 discutiamo significato e struttura della *user guide*, allo scopo di fornire una base di informazione per i futuri sviluppatori di *psort*.

Infine nell'Appendice presentiamo i programmi usati per la stesura di questo elaborato, e che riteniamo possano essere utili per la composizione di documentazione esterna.

1 Introduzione alla documentazione

La documentazione del software è di fatto un testo che accompagna i programmi. Essa contiene sia spiegazioni di come il software operi, sia informazioni sul corretto uso di quest'ultimo, ed assume un diverso significato a seconda del ruolo dell'utente.

Da un punto di vista più teoretico, la documentazione è una parte importante dell'ingegneria del software ed essa si esegue con criteri differenti ed in diversi tempi dello sviluppo del software. Passiamo ora ad elencare le categorie principali di documentazione e le loro relative caratteristiche in sintesi:

Requirements documentation

sono enunciati che identificano gli attributi, le capacità, le caratteristiche, o la qualità di un sistema. Questo è il fondamento di ciò che è stato o sarà implementato.

Architecture/Design documentation

è una panoramica del software. Comprende le relazioni di un ambiente ed i principi di costruzione da utilizzare nella progettazione dei componenti del software.

Technical documentation

è la documentazione di codice, algoritmi, interfacce e API (Application Programming Interface).

End User documentation

sono i manuali per l'utente finale, per gli amministratori di sistema e per il personale di supporto.

Marketing documentation

è il materiale promozionale del prodotto software, oltre ad essere analisi della domanda del mercato e di come commercializzare il prodotto.

1.1 Requirements documentation

La Requirements documentation è la descrizione di quello che il software può già eseguire, o di ciò che in futuro dovrà operare. Chi la utilizza incomincia a delinearla prima dello sviluppo del programma, poiché essa viene utilizzata per definire i risultati ottenuti e gli obiettivi; per questo, di solito, i progettisti di software la usano come base di partenza del loro programma.

Comunque questo tipo di documentazione viene utilizzata da tutti i soggetti coinvolti nella produzione del software: utenti finali, clienti, product manager, project manager, marketing, architetti software, sviluppatori e tester, solo per citarne alcuni.

La Requirements documentation in realtà è spesso incompleta (o inesistente), e vogliamo riportare qui di seguito alcuni dei più frequenti motivi:

- la formattazione non comune, ad esempio possono essere scritti in linguaggio naturale, oppure in dettagliate formule matematiche o rappresentazioni grafiche, oppure da una combinazione di tutte.
- la difficoltà nel capire quello che per questa documentazione sarà necessario, e quanto può o deve essere lasciato ad un altro tipo di documentazione (ad esempio quella di architecture/design)
- la difficoltà a documentare considerando la varietà di persone che la dovrà leggere ed utilizzare

Il rovescio della medaglia è che, senza un'adeguata Requirements documentation, le variazioni del software, da eseguire in un secondo momento, diventano più difficili e quindi più soggette ad errori (di conseguenza la qualità del software diminuisce) e risultano costose in termini di tempo.

In ogni modo la necessità di questo tipo di documentazione è strettamente condizionata dall'aspettativa di vita del software, ed è tipicamente legata alla complessità ed all'impatto del prodotto. Ad esempio, se il software terminerà la propria utilità entro qualche mese, la Requirements documentation risulta quasi inutile. Invece, nel caso in cui il software è solo una prima versione che deve essere aggiornata, questo tipo di documentazione può essere molto utile per la gestione delle variazioni del software e per verificare che le modifiche introdotte non abbiamo causato problemi. Inoltre, se il software è molto complesso e sviluppato da molte persone, la Requirement documentation può essere utile per comunicare al meglio i progressi e gli obiettivi da raggiungere.

Per gestire la notevole complessità e la natura mutevole della Requirements documentation, vengono in aiuto i sistemi database-centric. Infatti, possiamo includere in questo tipo di documentazione un po' particolare, i report che gli sviluppatori utilizzano per aggiornare i colleghi attraverso software di controllo di versione come ad esempio *Git* o *SubVersion*.

1.2 Architecture/Design documentation

L'Architecture documetation è una particolare Design documentation che contiene ben poche informazioni specifiche per il codice. Questi documenti infatti non descrivono come programmare una routine specifica, o anche poiché la routine è stata implementata in una forma particolare rispetto ad un'altra, ma semplicemente stabilisce i requisiti generali che motivano l'esistenza di una routine. Un buona Architecture documentation è a corto di dettagli, ma è molto prolissa nelle spiegazioni. Potrebbe suggerire alcuni approcci per la progettazione di livello inferiore, ma lasciare le spiegazioni delle implementazioni ad altri documenti.

Un altro tipo di Design documentation è la documentazione di confronto, o studio commerciale. Essa assume spesso la forma di articoli o rapporti. La documentazione di confronto si concentra su un aspetto specifico del sistema e suggerisce approcci alternativi: questo vale per ogni livello del software (dall'interfaccia utente, al livello di architettura). Essa illustrerà qual è la situazione, analizzerà una o più alternative, ed enumererà i pro ed i contro di ciascuna di esse.

Un buon documento di studio commerciale dovrebbe esprimere la propria idea in modo chiaro (senza utilizzare un linguaggio artificioso), ma soprattutto dovrebbe risultare imparziale. Esso dovrebbe spiegare onestamente e chiaramente i costi di qualsiasi soluzione eventuale.

L'obiettivo di uno studio commerciale è quello di elaborare la soluzione migliore, piuttosto che spingere un particolare punto di vista. È del tutto plausibile affermare di non essere giunti ad una conclusione, o concludere che nessuna delle alternative sia sufficientemente migliore della linea base, da giustificare un cambiamento. Esso dovrebbe essere affrontato con un approccio scientifico, non come una tecnica di marketing.

1.3 Technical documentation

La maggior parte degli sviluppatori di software, quando parlano di documentazione del software, si riferisce solo a questo genere di documentazione, la quale coincide principalmente coi documenti che corredano il codice sorgente. È importante che la documentazione del codice sia abbastanza approfondita, ma non talmente dettagliata da renderne l'aggiornamento laborioso e complesso. Questo genere di documentazione viene usata principalmente dai programmatori e dai tester, ma anche dagli utenti finali, soprattutto se questi sfruttano il codice sorgente e non un compilato od un pacchetto.

Negli ultimi tempi si è rilevato un incremento dell'importanza di questo tipo di documentazione dovuto principalmente alla maggior diffusione, rispetto a prima, di competenze tecniche tra gli utenti finali di software.

Spesso strumenti, come Doxygen o Javadoc, vengono utilizzati per generare automaticamente la documentazione del codice, ovvero estraggono informazioni dal codice sorgente, come nome e parametri dei metodi e, soprattutto, i commenti del programmatore, per crearne manuali di riferimento, in forma di file di testo o di HTML. La documentazione del codice è spesso organizzata in uno stile tipo guida di riferimento, consentendo ad un programmatore di cercare rapidamente una funzione arbitraria od una classe.

L'idea di auto generare la documentazione risulta interessante e molto utile per i programmatori, per vari motivi. Ad esempio, poiché viene estratto dal codice sorgente stesso, il programmatore può utilizzare gli stessi strumenti con cui crea il codice, per produrre la documentazione, e scrivere quest'ultima facendo riferimenti, che risultano automatici, a passaggi del codice.

Naturalmente, il rovescio della medaglia è che i programmatori possono modificare solo questo tipo di documentazione, e dipende da loro aggiornarne l'uscita (per esempio, eseguendo un cron job per aggiornare i documenti di notte). Alcuni potrebbero, al contrario considerarlo vantaggioso.

L'informatico statunitense Donald Knuth ha insistito sul fatto che la documentazione può essere un processo di ripensamento molto difficile. E per questo egli sosteneva il concetto di scrivere la documentazione allo stesso tempo del codice e sullo stesso file, e successivamente estrarla con mezzi automatici, di cui stiamo discutendo. E per questo metodo ha coniato il termine di *literate programming.*

Da applicazioni pratiche della *literate programming,* in contesti di programmazione reale, si è giunti a definire l'*Elucidative programming.* Spesso, gli sviluppatori di software devono essere in grado di creare e di accedere ad informazioni che non possono essere

parte del file di origine. Il paradigma Elucidative propone che il codice sorgente e la documentazione siano salvati separatamente, ma entrambi contengano traccia di riferimenti incrociati. Un programma di nome *Kelp*, sfrutta questo paradigma, memorizza le annotazioni in file separati, e collega le informazioni al codice sorgente in modo dinamico.

1.4 User documentation

Se la documentazione del codice spiega al lettore come interpretare il codice sorgente, la User documentation descrive semplicemente come utilizzare il software. Nel caso in cui si stia parlando di una libreria software, la User documentation e la documentazione del codice possono efficacemente essere equivalenti, ma per un'applicazione generale spesso questo non è attuabile e le due documentazioni si distanziano considerevolmente.

In genere, la User documentation descrive ogni caratteristica del programma, ed assiste l'utente nel mettere in pratica l'utilizzo di queste caratteristiche. Una buona User documentation può anche arrivare al punto di fornire assistenza completa alla risoluzione dei problemi. È molto importante per questo documento non essere causa di confusione, questo implica principalmente coerenza, semplicità ed un aggiornamento costante. Questa documentazione non deve essere organizzata in una maniera particolare, ma è molto importante che possieda un indice completo.

La User documentation è considerata come un contratto che specifica ciò che il software dovrà eseguire. In genere, gli scrittori API dovrebbero essere i più preparati per scrivere una buona User documentation, in quanto ben consapevoli delle architetture software e delle tecniche di programmazione usate.

La User documentation può essere organizzata in tre modi principali, che passiamo ad elencare qui di seguito:

Tutorial:
l'approccio tutorial è considerato il più utile per i nuovi utenti, in quanto questi ultimi vengono guidati in ogni passo nella realizzazione di particolari esercizi esplicativi.

Tematico:
l'approccio tematico, con capitoli o sezioni dedicati a particolari aree di interesse, generalmente, viene apprezzato di più da un utente intermedio. Alcuni autori preferiscono trasmettere le loro idee attraverso un articolo che fornisca una conoscenza di base, e successivamente correlarlo di risoluzioni di problemi non appena questi verranno segnalati dalla popolazione degli utenti.

Listato o di riferimento:

L'ultimo tipo di principio organizzativo è quello in cui i comandi o funzioni sono semplicemente in ordine alfabetico o raggruppate in modo logico, spesso attraverso indici con riferimenti incrociati. Quest'ultimo approccio è di maggiore utilità per gli utenti avanzati che sanno esattamente che tipo di informazioni stanno cercando.

NOTA: Una lamentela molto comune da parte degli utenti in merito alla documentazione di un software, è che di solito solo uno di questi tre approcci viene realizzato per la quasi totale esclusione degli altri due.

Inoltre è comune, per ridurre la dimensione della documentazione per i personal computer, fornire solo voci di riferimento per la guida in linea. In aggiunta, spesso il lavoro di scrivere il tutoraggio viene lasciata ad editori privati, i quali a loro volta vengono guidati dallo sviluppatore del software.

1.5 Marketing documentation

Per molte applicazioni è necessario avere del materiale promozionale per incoraggiare osservatori occasionali a dedicare del tempo ad imparare l'utilizzo del prodotto.

Questa forma di documentazione ha tre scopi:

1. Entusiasmare il potenziale utente del prodotto ed infondere in loro il desiderio di approfondire il loro coinvolgimento.

2. Informarli su ciò che il prodotto fa esattamente, in modo che le loro aspettative siano in linea con quello che otterranno realmente.

3. Spiegare la posizione del proprio prodotto rispetto ad altre alternative.

Una buona tecnica di marketing è quella di fornire uno slogan, chiaro e facile da ricordare, che esemplifichi il punto di vista che vogliamo trasmettere, e sottolinea l'interoperabilità del programma con altro materiale fornito dal produttore.

2 *psort* ed il Sort Benchmark

2.1 *psort*

psort è un software per ordinamento di grandi moli di dati su memoria esterna sviluppato al laboratorio ACG, DEI - Università di Padova (i siti internet di riferimento sono reperibili in bibliografia [1]). Gli ambiti di applicazione di *psort* spaziano dall'ordinamento di record in grandi database all'ordinamento di grandi moli di dati scientifici (es. misure astronomiche). *psort* è attualmente il software più veloce al mondo nella sua categoria secondo il Sort Benchmark 2011, di cui ha vinto anche le edizioni 2008 e 2009, rimanendo imbattuto nel 2010.

2.1.1 *psort* in breve

psort è una libreria per l'ordinamento di grandi moli di dati. Nella sua versione standard, *psort* effettua l'ordinamento dell'input, visto come sequenza di record di dimensione arbitraria, in spezzoni (run) di taglia paragonabile alla dimensione della memoria principale (prima fase), seguito da uno o più merge delle run (seconda fase).

La maggior parte degli algoritmi utilizzati all'interno del software si basano sul "classico" principio del merge. È stata adottata questa logica di implementazione per fornire un comportamento deterministico al software: questo garantisce stabilità ed alte prestazioni anche con gli ingressi nei peggiori dei casi.

psort lavora su sistemi GNU/Linux e FreeBSD, e può utilizzare i dischi in maniera indipendente, o come un volume RAID singolo.

2.1.1.1 Prima fase

Nella prima fase, *psort* ordina il file di ingresso. Se il file è più piccolo della memoria principale, termina l'esecuzione del programma scrivendo l'output ordinato su disco. Altrimenti, scrive su disco alcune run ordinate, di circa la dimensione della memoria principale (o la dimensione specificata dall'utente).

Una parte della memoria RAM (la cui percentuale è impostabile dall'utente) viene occupata dai buffer di lettura e scrittura da e per il disco. A partire dal buffer di lettura, il programma divide ogni record in una chiave ed un carico. Prima di essere ordinate, le chiavi vengono pre-processate per ottenere un confronto aritmetico. Inoltre, se necessario, alle chiavi viene aggiunta un'estensione (in formato di parola binaria) contenente informazioni sulla locazione in memoria dei carichi associati alle chiavi stesse.

Successivamente, *psort* forma dei blocchi di chiavi ordinate utilizzando mergesort, per poi unire i blocchi utilizzando un k-way merge-tree. Quando il programma estrae una chiave dalla radice della struttura merge-tree, essa viene riportata allo stato originale di record, quindi si cancella l'estensione, si post-processa la chiave e la si riallaccia al carico. Dopo, il record viene direttamente spostato nel buffer di scrittura da cui si andrà a scrivere la run su disco.

La dimensione dei blocchi delle chiavi deve essere bilanciata accuratamente tra due criteri opposti. Il primo criterio cerca di diminuire il più possibile la dimensione in modo che i blocchi si inseriscano perfettamente nella cache L2, questo garantirebbe un solo accesso di lettura ed uno di scrittura nella fase di ordinamento. L'altro invece cerca di aumentare la dimensione per ridurre il numero di blocchi e quindi il numero di nodi esterni del merge-tree, questo criterio ha lo scopo di cercare di contenere interamente la struttura ad albero nella cache L2.

2.1.1.2 Seconda fase

In questa fase, un k-way merge-tree (la cui dimensione deve essere contenuta nella L2 cache), unisce le run già ordinate nella fase precedente. Nelle operazioni intermedie tra la lettura e la scrittura da disco, anche nello stage_two i record vengono separati in carichi e chiavi (le quali sono di nuovo pre-processate e probabilmente estese) .

Nello seconda fase la memoria principale viene suddivisa in strutture dati di supporto (ad esempio i buffer di lettura e scrittura), ed in strutture dati di tipo code ordinate, una per ogni run di ingresso ed un'altra che rappresenterà lo spazio a disposizione.

Queste ultime strutture sono composte da microbuffer, contenenti le chiavi che il k-way merge-tree deve ordinare. Ogni volta che l'albero libera un microbuffer, questo viene spostato nella coda rappresentante lo spazio libero. Tutto questo per un uso efficiente dello spazio a disposizione.

Quando i dati contenuti in una coda scendono sotto una determinata soglia (la quale può essere scelta come opzione dall'utente), lo stage_two andrà a leggere dalla run corrispondente dei nuovi microbuffer, togliendoli dalla coda dello spazio vuoto.

Per ridurre il numero di volte che il programma deve interromper l'ordinamento per recuperare nuovi dati, *psort* cerca di mantenere la lunghezza delle code in progressione aritmetica. Con le impostazioni di default una coda viene ricaricata alla dimensione massima ogni volta che scende sotto l'80% della dimensione minima della coda.

2.1.2 Colli di bottiglia

Senza scendere nei dettagli, l'hardware del sistema presenta fondamentalmente 4 colli di bottiglia per il funzionamento di *psort*, ed essi sono:

1. La velocità di lettura/scrittura del disco rigido
2. Il southbridge della scheda madre
3. La dimensione della memoria principale
4. La dimensione della memoria cache del processore

2.2 Sort Benchmark e PennySort

Il Sort Benchmark nasce per valutare ogni anno le prestazioni dei calcolatori, utilizzando processi di ordinamento, l'indirizzo del sito ufficiale è presente in bibliografia [3]. La competizione richiede l'utilizzo di software di ordinamento per valutare le performance dell'intero sistema del computer, perché l'ordinamento è un carico di lavoro semplice e bilanciato, che coinvolge accessi di memoria, I/O e CPU.

Il primo aprile del 1985 venne definito Datamation all'interno dell'articolo scientifico intitolato "A Measure of Transaction Processing Performance". Da esso prese origine la competizione denominata DatamationSort, la prima del Sort Benchmark. Essa consisteva nello stabilire il minor tempo possibile per ordinare un file di input, composto da un insieme in ordine casuale di un milione di record da 100 byte, contenenti ciascuno una chiave da 10 byte. Il tempo doveva includere l'avvio del programma, la creazione dei file temporanei e di output, e l'esecuzione dell'ordinamento.

Col passare degli anni, per motivi come l'aumento delle conoscenze informatiche, e delle prestazioni dell'hardware (legge di Moore), il risultato vincitore della gara diminuì fino a raggiungere poche manciate di secondi, portando di conseguenza i parametri del concorso a divenire obsoleti come misura di riferimento per le prestazioni.

Questo fatto ha portato la commissione del Sort Benchmark a ridefinire la competizione, imponendo ai partecipanti un limite fissato di tempo nell'esecuzione del software presentato, trascurando l'obiettivo di ordinare una quantità predeterminata di dati. Così, nel 1994 DatamationSort fu sostituito da MinuteSort, una competizione che richiede di ordinare quanti più dati possibile nell'arco di 60 secondi – senza alcuna restrizione hardware. Il formato dei record è identico al formato originale proposto in Datamation: 100 byte di caratteri stampabili, di cui i primi 10 costituiscono la chiave.

Successivamente venne introdotta anche PennySort, essa richiede di ordinare quanti più dati possibili in un budget di tempo inversamente proporzionale al costo del calcolatore utilizzato per ottenere il risultato presentato alla competizione.

Dal 2009 il Sort Benchmark presenta quattro sezioni: MinuteSort, PennySort, GraySort e JouleSort.

La sezione GraySort è stata introdotta per valutare la maggior velocità di ordinamento praticabile, espressa in TBs / minute, necessaria ad ordinare una grandissima quantità di dati (attualmente, minimo 100 TeraByte).

La sezione JouleSort, invece, per stimare la minor quantità di energia spesa per ordinare rispettivamente 10^8, 10^9 o 10^{10} record (rispettivamente 10 GB, 100 GB o 1 TB di file di input).

2.2.1 PennySort

Questa competizione valuta la maggior quantità di dati ordinati in un tempo deteminato dal costo del calcolatore, assemblato dai concorrenti e presentato alla commissione assieme al software.

Introduciamo brevemente come questo intervallo di tempo, viene calcolato. Il primo passo consiste nel misurare il costo ottimistico della macchina, ovvero, si considerano solo le spese iniziali dell'hardware, tralasciando le spese aggiuntive come l'elettricità, il software di terze persone (ad esempio il sistema operativo), il compenso delle persone che la gestiscono o l'aria condizionata per mantenerla operativa. Il valore risultante deve essere considerato in centesimi di dollaro statunitense (penny), ed i prezzi devono essere reperibili dalla commissione su siti internet ritenuti affidabili (nella sezione FAQ del sito ufficiale del Benchmark citato in bibliografia vi è qualche esempio).

Nel passaggio successivo, si deve considerare il tempo di vita utile di un computer, ma non essendo questo un parametro che si può determinare a priori con esattezza, la commissione ha stabilito che per tutti i concorrenti questo valore corrisponde a 3 anni, da considerarsi in secondi, ovvero 94 608 000 [s]. Infine, prendendo questo valore e dividendolo per il prezzo della macchina, calcolato prima, otteniamo il budget di tempo su cui si baserà la gara.

Dunque, ponendo la questione in altri termini, PennySort ha lo scopo di determinare la maggior quantità di dati che un calcolatore può ordinare in un intervallo di tempo che corrisponde ad un penny del suo valore.

Come si può facilmente osservare, PennySort presenta comunque delle difficoltà nelle comparazioni dei risultati dei vari partecipanti, derivate principalmente dalla componente commerciale. Per chi fosse interessato a conoscere maggiori dettagli sulle regole di acquisto di un sistema per poterlo presentare alla competizione, lo rimandiamo alla già citata sezione FAQ del sito ufficiale.

2.2.2 Indy e Daytona

Ognuna delle competizioni del Sort Benchmark si suddivide ulteriormente in due categorie, chiamate rispettivamente Daytona ed Indy. Gli obiettivi da raggiungere rimangono gli stessi, le due categorie si differenziano sul fatto che i software presentati in Indy possono essere specificatamente perfezionati per la gara, mentre nella categoria Daytona devono essere presentati software general purpose.

Quindi, nella categoria Indy vengono fornite delle condizioni (che, nella maggioranza dei casi al di fuori della gara, non possono essere assunte come verificate), sulle quali i progettisti possono basare delle ottimizzazioni per il loro software. In particolare:

- le dimensioni fissate dei record e delle chiavi (rispettivamente 100 e 10 byte)
- la densità di distribuzione uniforme delle chiavi di ingresso

Invece, nell'elenco seguente riportiamo quelle che riteniamo le più significative delle caratteristiche che i software devono possedere per essere presentati nella categoria Daytona:

- non mostrare considerevoli alterazioni delle prestazioni di ordinamento, con tipi di record e/o di chiavi differenti da quelli della categoria Indy

- l'algoritmo non deve dipendere troppo dall'assunzione di densità di distribuzione uniforme del file di input
- essere in grado di mantenere un funzionamento continuato per un'ora senza lanciare un errore di sistema

2.3 Riconoscimenti ufficiali

Riportiamo in figura 2.1 nella pagina seguente i riconoscimenti ottenuti negli anni da *psort* alla Sort Benchmark. Le tabelle suddividono i risultati in anno di edizione della gara PennySort, l'intervallo di tempo per un penny ottenuto col metodo spiegato precedentemente, il risultato nella rispettiva categoria con la quantità di dati ordinati nel budget di tempo, e l'ultima riga presenta alcuni dettagli del computer con cui si è sostenuta la competizione.

PennySort	2008
Budget di tempo	2˙408 secondi
Categoria Daytona	vinto ordinando 181 GB
Categoria Indy	vinto ordinando 190 GB
Sistema presentato	2.4 Ghz AMD Athlon 64, 2 GB RAM, 4x160GB SATA disks, Linux

PennySort	2009
Budget di tempo	2˙211 secondi
Categoria Daytona	vinto ordinando 223 GB
Categoria Indy	vinto a pari merito con OzSort (progetto dell'università di Melbourne) ordinando rispettivamente 248 e 246 GB
Sistema presentato	2.6 Ghz AMD Athlon LE 1640, 4 GB RAM, 5x160 GB 7200 RPM SATA, Linux

PennySort	2010
Psort non è stato presentato, ma il risultato ottenuto l'anno precedente è rimasto imbattuto nella categoria Daytona	

PennySort	2011
Budget di tempo	2˙096 secondi
Categoria Daytona	vinto ordinando 286 GB
Categoria Indy	vinto ordinando 334 GB
Sistema presentato	2.7 Ghz AMD Sempron, 4 GB RAM, 5x320 GB 7200 RPM Samsung SpinPoint F4 HD332GJ, Linux

Figura 2.1: Riconoscimenti di *psort*

3 Doxygen

3.1 Introduzione

Doxygen è un'applicazione per la generazione automatica della documentazione a partire dal codice sorgente di un generico software. È un progetto open source rilasciato sotto licenza GPL e scritto per la maggior parte da Dimitri van Heesch a partire dal 1997.

Il programma è un sistema multipiattaforma (Linux, MacOS, Windows , ecc.) ed opera con i linguaggi C++, C, Java, Objective-C, Python, IDL (versioni CORBA e Microsoft), Fortran, PHP, C#, e D.

Doxygen può essere utile in tre modi per coloro che lavorano ad un progetto software:

1. Può generare la documentazione per browser di rete (in HTML) e/o un manuale di riferimento off-line (in LaTeX) da un insieme di file sorgenti adeguatamente documentati. Inoltre, ci sono dei programmi che supportano Doxygen per generare file di output anche nei formati: RTF (MS-Word), PostScript, PDF con collegamenti ipertestuali, HTML compresso, e pagine man di Unix.

2. È possibile configurare Doxygen per estrarre la struttura del codice da file sorgente non documentati. Questo è molto utile per orientarsi rapidamente in distribuzioni molto estese di codice sorgente. È anche possibile visualizzare le relazioni tra i vari elementi mediante i grafici delle dipendenze, diagrammi dell'ereditarietà e diagrammi di collaborazione, che si possono generare tutti automaticamente.

3. Si può anche eccedere nell'uso di Doxygen come editor per creare documentazione normale (può essere preso come esempio il suo manuale).

La documentazione prodotta in formato HTML, si giova di un sistema di collegamenti ipertestuali molto curato che permette al lettore un'agevole navigazione della struttura dei file sorgenti. I documenti possono essere generati in diverse lingue, tra cui è compreso l'italiano.

Doxygen, come funzioni di base, appare simile a Javadoc o Qt-Doc, ma offre una vasta gamma di funzionalità aggiuntive, tra cui citiamo:

- È un sistema multipiattaforma (disponibile per Unix, Windows e MacOS X).
- È compatibile con Javadoc, Qt-Doc, e KDOC.
- Riconosce e genera automaticamente riferimenti incrociati.
- Converte le tag HTML in markup per LaTeX, RTF, o pagine man.
- Genera automaticamente i diagrammi delle classi.
- Include le formule matematiche in stile LaTeX .

Doxygen è il sistema di documentazione di gran lunga più utilizzato nei grandi progetti open source in C++. Riportiamo di seguito due esempi su tutti, con i link per la loro documentazione online dei sorgenti:

- RPM - http://www.rpm.org/rpmapi-4.1/
- The GNU Standard C++ Library - http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/index.html

Ci sono molti altri progetti elencati nella pagina "Projects that use Doxygen" nel sito officiale di Doxyen, che riportiamo in Bibliografia. Navigare tra la documentazione generata dei vari progetti è il modo migliore per capire attraverso degli esempi cosa effettivamente può fare questo programma.

Nella figura 3.2 a pagina 38 presentiamo un'immagine estratta dal manuale di Doxygen, la quale mostra la relazione tra gli strumenti di Doxygen ed il flusso di informazioni tra di loro (sembra complessa ma è solo perché vuole essere completa):

3.1.1 Perché utilizzare Doxygen?

3.1.1.1 Perché usare un sistema automatico di generazione della documentazione?

Alcune ragioni, le abbiamo già elencate nel capitolo 1, di seguito ne facciamo un piccolo riassunto:

- Rapidità di aggiornamento - La documentazione viene estratta direttamente dal codice sorgente, il che rende molto più facile mantenere aggiornata la documentazione online.
- "Prossimità spaziale" - La documentazione viene generata a partire dai commenti posti nel codice, ne consegue che non appena si modifica un elemento del sorgente, basta modificarne il commento relativo.

- Riutilizzo dei propri commenti - Supponendo che si ha già commentato il codice, basterà modificarne lievemente la formattazione ed il lavoro è già fatto.
- Formattazione automatica e riferimenti incrociati al codice con poche e semplici annotazioni.
- I commenti nel codice contengono meta-informazioni importanti - Le quali devono essere rese note a chiunque sia incaricato di mantenere il codice, che si tratti di voi stessi del futuro o di qualcun altro. Uno stile formale per questi commenti, abbinato con un parser, può rendere queste informazioni disponibili in formato personalizzato per i diversi destinatari : sia per la gestione dei progetti, che per i tester, e simili.

3.1.1.2 Perché Doxygen?

- È un programma gratuito - Quindi è perfetto per:
  - Coloro che dispongono di un budget limitato
  - Valutare se e come un sistema di documentazione automatica ti può aiutare
  - Individuare le funzioni che ci si aspetta quando si pensa di spendere per un prodotto migliore
- È un progetto OpenSource che mette a disposizione l'installer - Quindi risulta semplice da installare, soprattutto per chi opera in sistemi Windows e MacOS, ed avere il codice sorgente disponibile si presenta come un ulteriore bonus.
- È altamente configurabile - Il sistema è facilmente configurabile al fine di permettere all'utilizzatore di intervenire su tutti gli aspetti della documentazione prodotta.

3.2 L'utilizzo di Doxygen

Doxygen è un software basato su comandi da terminale. Chiamare Doxygen con l'opzione - help dalla riga di comando fornirà una breve descrizione dell'utilizzo del programma.

In generale, per realizzare un manuale per un progetto è necessario seguire questi passaggi:

1. Si documenta il codice sorgente con blocchi di commenti speciali.
2. Si genera un file di configurazione chiamando Doxygen con l'opzione-g:

```
doxygen -g <config_file>
```

3. Si modifica il file di configurazione in modo che corrisponda al progetto. Nel file di configurazione è possibile specificare il file di input e molte altre informazioni facoltative.
4. Si lascia generare la documentazione a Doxygen, in base alle impostazioni del file di configurazione, attraverso il comando:

```
doxygen <config_file>
```

I passi 2 e 3 possono essere sostituiti dall'utilizzo di Doxywizard, di cui parleremo nella sezione 3.3 a pagina 33.

3.2.1 Come scrivere i commenti nel codice sorgente

Il funzionamento di Doxygen richiede una particolare formattazione dei commenti inseriti nel codice sorgente. Per chi ha già familiarità con Javadoc o Qt-Doc, conosce già le basi della scrittura della documentazione Doxygen. Per chi non conosce entrambi i sistemi, l'idea di base è che si devono mettere dei commenti appositamente formattati immediatamente sopra tutto quello che si vuole documentare (come classe, struttura, metodo, campo, ecc.).

Comunque, le regole di formattazione sono chiaramente spiegate nel manuale. Riportiamo di seguito degli esempi.

Javadoc-style example:

```
/**
 * Method documentation.
 * @param x The parameter.
 * @return The return value.
 * @see anotherFunction()
 */

/** Single-line documentation. */
/// Single-line documentation.
```

Qt-style example:

```
/*!
  Method documentation.
 \param x The parameter.
 \return The return value.
 \sa anotherFunction()
*/

/*! Single-line documentation. */
//! Single-line documentation.
```

Inoltre, Doxygen permette anche di posizionare i commenti per la documentazione dopo un elemento, questo risulta utile per documentare rapidamente le enumerazioni, le strutture, e le variabili di membro:

```
int a; ///< Javadoc-style.
char b; //!< Qt-style.
```

Per i file C e C++, è possibile inserire la documentazione per gli elementi sia nell'header che nel file principale; Doxygen combinerà le dichiarazioni con il codice vero e proprio.

3.2.2 Alcuni Markup utili

Come si è potuto notare anche negli esempi dei commenti precedenti, Doxygen permette all'utente di indicare elementi particolari all'interno del codice, attraverso delle semplici tag. Riportiamo di seguito una tabella con quelle che abbiamo ritenuto tra le più significative; lo stile che abbiamo usato è quello di Qt, ma per la maggior parte delle tag sono equivalenti nello stile Javadoc, solo che iniziano con il simbolo @ piuttosto che con una backslash:

Markup	Description
\param var desc...	Descrizione di un parametro chiamato var, associato ad una funzione o metodo.
\return desc...	Descrizione del valore di ritorno di una funzione.
\sa elem	Per inserire un link "see also" ad elem, che può essere una funzione, una classe o qualsiasi altro elemento identificativo e documentato.
\author name	Per indicare l'autore di un elemento del codice.
\version ver	Per indicare la versione di un elemento del codice.
\todo desc...	Per lasciare una nota riguardo al lavoro non finito.
\warning desc...	Per lasciare un warning.
\b, \c, \e	Per imposta la prossima parola rispettivamente in bold, italic o courier.

\n	Per iniziare una nuova linea.
\mainpage	Per indicare che la sezione seguente dovrebbe comparire sulla pagina principale. È un buon modo per segnalare gli elementi più importanti, ad esempio le classi maggiori, etc.
\par, \par Title	Per iniziare un nuovo paragrafo (opzionalmente con un titolo di paragrafo), funziona anche all'interno di altri paragrafi (come \param).
Lists	Doxygen crea automaticamente un elenco se più righe iniziano con un segno meno nella stessa posizione. Gli elenchi numerati possono essere creati iniziando la riga con un segno meno e un cancelletto (-#). Vedere la documentazione Doxygen per ulteriori informazioni.
More..	Doxygen supporta molte altre tag di quelle citate, ma per esse rimandiamo alla sezione specifica del manuale di Doxygen.
HTML..	Doxygen supporta anche i tag HTML, che vengono convertiti in altre uscite più o meno correttamente. Però, anche per queste, vi rimandiamo alla documentazione originale.

3.2.3 Il file di configurazione

Doxygen utilizza un file di configurazione per determinare tutte le sue impostazioni. Ogni progetto deve avere un proprio file di configurazione. Dove con progetto intendiamo sia un singolo file sorgente, ma anche un'intera struttura ad albero di sorgenti che il programma deve analizzare ricorsivamente.

Per semplificare la creazione di un file di configurazione, con Doxygen è possibile creare un modello. Per fare questo si digita a riga di comando l'opzione:

```
doxygen -g <config-file>
```

il file generato automaticamente da Doxygen contiene dei parametri generici che l'utente può personalizzare o lasciare invariati. Questo file è un elenco di assegnazioni di opportuni valori a determinati parametri (TAG). Ogni tag è formata dalla coppia di informazioni

```
NOME_PARAMETRO = VALORE_PARAMETRO
```

analogamente a quanto avviene nei file di configurazione di numerosi altri prodotti open source e simile a quello di un Makefile. Riportiamo nella sezione 3.2.5 a pagina 31 un frammento di un esempio di file di configurazione.

Come è possibile notare in questo frammento, alcune delle impostazioni che l'utente stabilisce, attraverso il file di configurazione, sono:

- Il nome del progetto
- La directory di destinazione, dove il programma salverà la documentazione prodotta
- La lingua della documentazione
- La directory di origine dove si trovano i file sorgente
- L'estensione dei file di input da considerare come origine
- L'indicazione di ricorsività nella directory di origine
- L'indicazione di generazione del formato HTML
- Il nome della directory di destinazione per il formato HTML
- L'indicazione di generazione del formato LaTeX
- Il nome della directory di destinazione per il formato LaTeX

Come abbiamo già accennato, c'è la possibilità di generare e modificare il file di configurazione attraverso il programma Doxywizard, di cui parleremo nella sezione 3.3 a pagina 33.

3.2.4 Alcune Tag interessanti

Se si vuole utilizzare Doxygen per documentare un progetto scritto su più file sorgenti con struttura ad albero, si deve assegnare la directory principale (contenente tutti i codici) al tag INPUT, e aggiungere uno o più modelli di file al tag FILE_PATTERNS (ad es.: *. cpp *. h).

Per ottimizzare ulteriormente l'elenco dei file analizzato dal programma, si può utilizzare i tag EXCLUDE e EXCLUDE_PATTERNS. Ad esempio per omettere tutte le directory di prova di un albero dei sorgenti, si potrebbe scrivere:

```
EXCLUDE_PATTERNS = */test/*
```

Riportiamo nella tabella seguente ulteriori tag che reputiamo interessanti, per approfondire, vi rimandiamo alla sezione apposita del manuale di Doxygen.

Opzione	Descrizione
EXTRACT_ALL = YES	Genera la documentazione per tutti gli elementi, anche se non sono ancora stati commentati. Questa opzione è utile per avere un'idea di quale sia la struttura di un progetto già iniziato ma senza documentazione, e di come sarà il risultato documentato. NOTA: Fintanto che questa opzione sarà abilitata, il programma non genererà warning per gli elementi privi di commenti.
JAVADOC_AUTOBRIEF = YES	Quando questa opzione è abilitata, permette di avere in un unico blocco sia il breve commento che la descrizione dettagliata (nonostante il suo nome, funziona anche per sorgenti C++). La prima riga (fino al primo periodo) di un blocco di commento è usata come descrizione breve.
SOURCE_BROWSER = YES	Con questa opzione abilitata, il programma genera un elenco dei file sorgente con riferimenti incrociati nella documentazione degli elementi del codice.
GENERATE_HTML = YES	Genera la documentazione in HTML, inclusi i diagrammi di classe.
GENERATE_LATEX = YES	Genera la documentazione in LaTeX. Inoltre sarà generato un makefile in modo che si possa compilare in PostScript, PDF, e le versioni DVI correttamente.
WARN_FORMAT = $file($line): $text	Con questa opzione configurata, si può fare doppio clic sui messaggi di avviso di Doxygen nella finestra di uscita per passare alla riga di codice sorgente rilevante
GENERATE_TREEVIEW = YES	Per output HTML, genera un indice sidebar.

3.2.5 File di configurazione di base

```
# The PROJECT_NAME tag is a single word (or a sequence of words
# surrounded by quotes) that should identify the project.
        PROJECT_NAME     = MyProject
# The OUTPUT_DIRECTORY tag is used to specify the (relative or
# absolute) base path where the generated documentation will be
# put. If a relative path is entered, it will be relative to
# the location where doxygen was started. If left blank the
# current directory will be used.
        OUTPUT_DIRECTORY         =
# The OUTPUT_LANGUAGE tag is used to specify the language in
# which all documentation generated by doxygen is written.
        OUTPUT_LANGUAGE = English
# The INPUT tag documented source files. You may enter
# file names like "myfile.cpp" or directories like
# "/usr/src/myproject".
# Separate the files or directories with spaces.
        INPUT    =
# If the value of the INPUT tag contains directories, you can
# use the FILE_PATTERNS tag to specify one or more wildcard
# pattern (like *.cpp and *.h) to filter out the source-files
# in the directories. If left blank the following patterns are
# tested: *.c *.cc *.cxx *.cpp *.c++ *.java *.ii *.ixx *.ipp
# *.i++ *.inl *.h *.hh *.hxx *.hpp *.h++ *.idl *.odl *.cs
# *.php *.php3 *.inc *.m *.mm
        FILE_PATTERNS    = *.h *.hh *.idl
# The RECURSIVE tag can be used to turn specify whether or not
# subdirectories should be searched for input files as well.
# Possible values are YES and NO. If left blank NO is used.
        RECURSIVE        = YES
# If the GENERATE_HTML tag is set to YES (the default)
# Doxygen will generate HTML output.
        GENERATE_HTML    = YES
# The HTML_OUTPUT tag is used to specify where the HTML docs
# will be put. If a relative path is entered the value of
```

```
# OUTPUT_DIRECTORY will be put in front of it. If left blank
# -html- will be used as the default path.
        HTML_OUTPUT      = html
# If the GENERATE_LATEX tag is set to YES (the default) Doxygen
# will generate Latex output.
        GENERATE_LATEX  = NO
# The LATEX_OUTPUT tag is used to specify where the LaTeX docs
# will be put. If a relative path is entered the value of
# OUTPUT_DIRECTORY will be put in front of it. If left blank
# -latex- will be used as the default path.
        LATEX_OUTPUT     = latex
```

3.3 Doxywizard

Per chi non desidera modificare il file di configurazione con un editor di testo, è stato creato Doxywizard, il quale è una GUI front-end in grado di creare, leggere e scrivere file di configurazione Doxygen, consentendo di impostare le opzioni di configurazione attraverso finestre di dialogo.

Doxywizard è un programma già presente nel pacchetto Doxygen, ma bisogna, durante la compilazione (se avete compilato Doxygen dai sorgenti), configurare l'opzione *–with-doxywizard flag* . Inoltre, Doxywizard richiede Qt 2.x per l'esecuzione.

Quando si avvia Doxywizard viene visualizzata la finestra principale. L'aspetto attuale dipende dal sistema operativo utilizzato, ne riportiamo un esempio in figura 3.1 a pagina 36.

Il programma è strutturato in vari passi per la configurazione di Doxygen. Il primo passo è quello di scegliere una delle modalità di configurazione proposte:

Wizard Selezionare questo pulsante per configurare rapidamente le impostazioni più importanti e lasciare il resto delle opzioni ai valori predefiniti.

Expert Selezionare questo pulsante per accedere alla gamma completa di opzioni di configurazione.

Load Selezionare questo pulsante per caricare da disco un file di configurazione già esistente.

Si noti che è possibile utilizzare insieme le modalità wizard ed expert, ad esempio usando la prima per configurare velocemente le opzioni di base e la seconda per affinare solo le impostazioni avanzate che interessano.

Il passo successivo alla configurazione è il salvataggio del file di configurazione con le opzioni desiderate. Questa seconda fase è obbligatoria per permettere a Doxygen di lavorare con la configurazione scelta. Inoltre, pone l'ulteriore vantaggio di poter riutilizzare le stesse impostazioni in successive esecuzioni di doxygen, ma anche in ulteriori progetti.

Dopo il salvataggio delle impostazioni, si deve selezionare una cartella da cui eseguire Doxygen. Essa rappresenterà la radice dell'albero dei sorgenti. Inoltre, questa impostazione è necessaria per quelle opzioni di configurazione che utilizzano i path relativi.

Una volta che il file di configurazione viene salvato e la directory di lavoro è stata impostata, è possibile eseguire Doxygen in base alle impostazioni selezionate premendo

il pulsante "Start". Una volta che Doxygen ha cominciato ad eseguire, potete annullare l'operazione cliccando nuovamente lo stesso tasto.

I messaggi di output che Doxygen invia durante il funzionamento saranno mostrati in una finestra di log. Una volta che Doxygen finisce l'esecuzione, questo registro può essere salvato come file di testo.

3.3.1 Finestre di dialogo Wizard

Se si seleziona il pulsante Wizard al primo passo, appariranno una serie di schede da compilare, di cui andiamo ad enunciare alcuni punti:

- I campi della scheda progetto non necessitano di ulteriori spiegazioni di quelle presenti sulla schermata. Una volta che Doxygen ha terminato, i risultati si troveranno nella cartella di destinazione. Doxygen metterà ogni formato di output selezionato in una sotto-directory.

- La scheda modalità consente di selezionare come Doxygen andrà ad analizzare i sorgenti. Di default, il programma prende in considerazione solo le parti che sono state documentate.

- È possibile selezionare sia come Doxygen dovrà presentare i dati, sia quali formati di output andrà a produrre (per HTML e LaTeX ci sono opzioni aggiuntive). Queste configurazioni non influenzano il modo in cui Doxygen analizza il codice sorgente.

- Nella scheda dei diagrammi è possibile selezionare quali grafici generare tra la serie messa a disposizione dal programma. Per la maggior parte dei diagrammi Doxygen necessita degli strumenti del pacchetto GraphViz (se si utilizzano i pacchetti binari per MacOS X questo strumento è già incluso).

3.3.2 Finestre di dialogo Expert

La finestra di dialogo Expert ha un certo numero di schede, una per ogni sezione nel file di configurazione. Ogni scheda contiene un numero di campi da compilare, uno per ogni opzione di configurazione in quella sezione.

Il tipo di input widget dipende dal tipo di opzione di configurazione:

- Per ogni opzione booleana (quelle opzioni alle quali si scrive YES o NO nel file di configurazione) vi è un check-box.

- Per le voci per cui si ha a disposizione un insieme fisso di valori (come OUTPUT LANGUAGE), viene utilizzata una casella combinata.
- Per le voci che prendono un valore intero da un intervallo, viene utilizzata una casella di riepilogo.
- Per il tipo di opzioni che prendono come input una stringa, viene utilizzato uno spazio vuoto.
- Per le opzioni che ricevono una lista di stringhe, un campo di una linea di modifica è disponibile, con un tasto '+' per aggiungere questa stringa alla lista e un '-' per rimuovere la stringa selezionata dalla lista. C'è anche un tasto '*' che, se premuto, sostituisce l'elemento selezionato nella lista con la stringa immessa nel campo di modifica.
- Per le voci di file e cartelle, ci sono dei pulsanti speciali che attivano una finestra di selezione file.

Per ottenere ulteriori informazioni sul significato di una opzione, selezionare il pulsante "Help" in basso a destra della finestra e poi sulla voce. Apparirà un tooltip con informazioni aggiuntive.

3.3.2.1 Opzioni del menù

La GUI front-end dispone di un menù di cui vogliamo riportare un paio di elementi utili:

Open... Questa opzione ha la stessa funzione del pulsante Load nella finestra principale e permette di aprire un file di configurazione dal disco.

Save_as... Questo ha la stessa funzione del pulsante "Salva" nella finestra principale e può essere utilizzato per salvare le impostazioni di configurazione corrente su disco.

Recent_configurations Permette di caricare rapidamente una configurazione salvata di recente.

Set_as_default... Memorizza le impostazioni di configurazione corrente come predefinita da utilizzare al successivo avvio di Doxywizard. Richiederà di confermare l'operazione.

Reset... Ripristina le impostazioni di fabbrica come impostazioni predefinite da usare. Richiederà di confermare l'operazione.

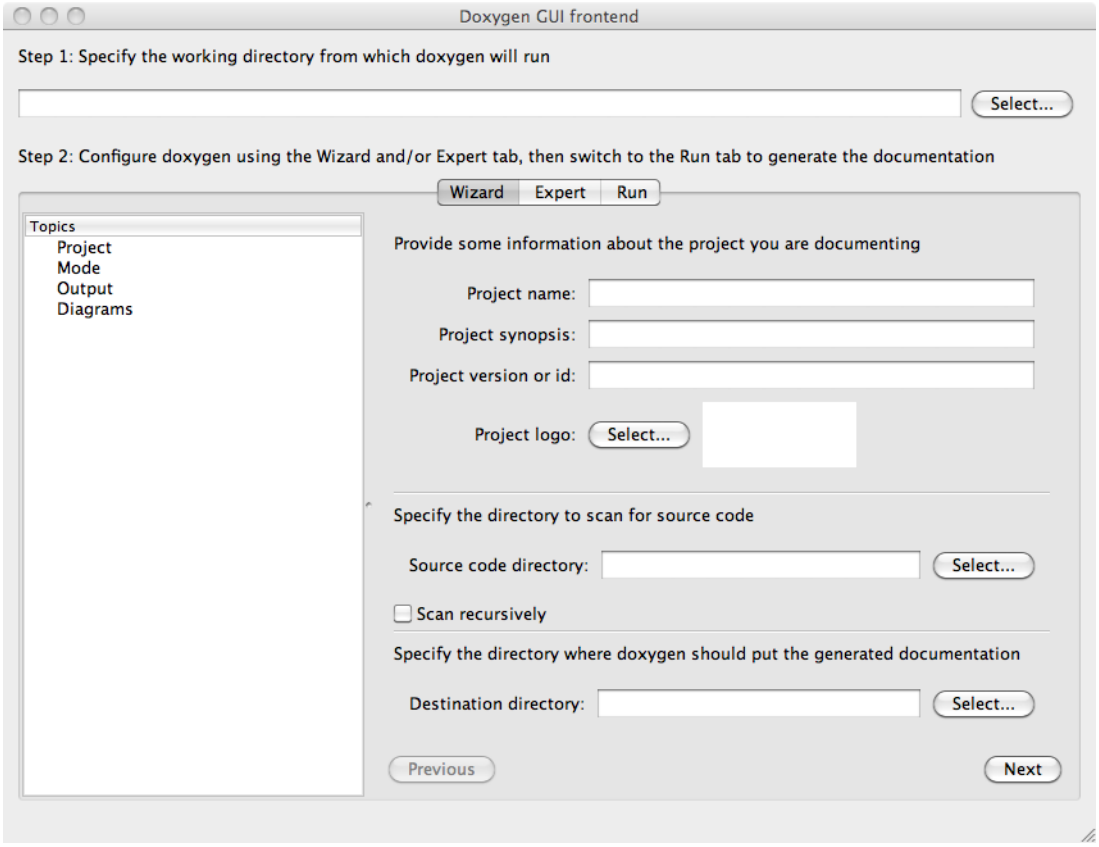


Figura 3.1: Immagine della pagina principale di Doxywizard della versione di Doxygen 1.7.5.1 per il sistema MacOS X

3.4 Documentare un progetto "in corso d'opera"

In chiusura di questo capitolo, vorremmo riportare alcuni consigli, che abbiamo trovato sull'articolo "10 Minutes to document your code" citato in bibliografia.

"Nel caso in cui, stiate lavorando in un grande progetto nel quale vorreste siano inseriti dei commenti al codice in un formato compatibile con Doxygen, ma l'idea di spulciare 10 000 righe di codice e documentarle vi fa rabbrividire. Ecco alcuni suggerimenti:

- Innanzitutto impostate il file di configurazione per il progetto, è un'operazione di breve durata.
- Documentate tutto quello che scrivete di nuovo.
- Aggiungete qualche commento alle funzioni che dovete modificare. È più facile aggiungere alcuni commenti in più sulle funzioni che si stanno rielaborando o aggiungendo. Inoltre, molte volte basta regolare commenti già esistenti o copiarli quasi completamente, da altre funzioni vicine e molto simili.
- Passate gli ultimi 5 minuti prima della pausa pranzo ad aggiungere commenti. Oppure: aggiungete sempre qualche commento alle funzioni prima di ogni aggiornamento del vostro gruppo di lavoro.

Inoltre, il miglior consiglio è quello di cominciare. Non appena si vedranno i primi risultati si sarà incoraggiati ad andare avanti."

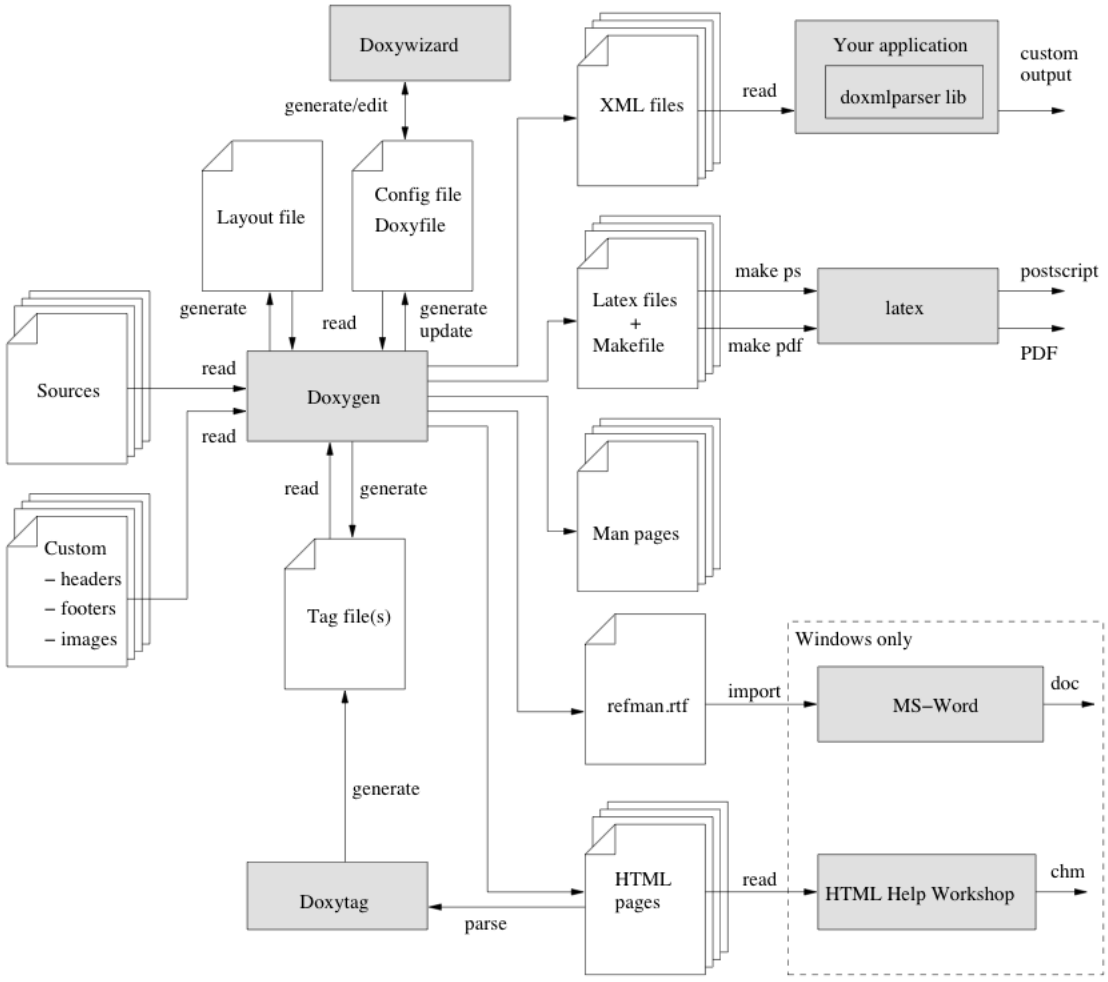


Figura 3.2: Il flusso delle informazioni di Doxygen

4 Man page

4.1 Introduzione

Le pagine man (*man-page*, abbreviazione di manual pages) fanno parte della vasta documentazione, per la gran parte preinstallata, che quasi tutti i sistemi operativi Unix e Unix-like forniscono.

Ogni pagina man è un documento indipendente, ma può comunque contenere riferimenti alle altre pagine. Essa rappresenta una guida esplicativa, che descrive in maniera sintetica un argomento, il quale può essere un comando, una funzione o un formato di un file, o altro ancora.

Le varie pagine man sono raggruppate in sezioni omogenee suddivise per tipo di argomento trattato (ad esempio, vi è una sezione per i comandi utente). Ma affronteremo in seguito questa suddivisione nello specifico.

Esse sono solitamente realizzate in lingua inglese, anche se, per alcuni sistemi, sono disponibili delle traduzioni.

4.1.1 Utilizzo

Per accedere ad una pagina del manuale si può digitare a riga di comando:

man nome_pagina

ad esempio, "man ftp".

Tradizionalmente il riferimento ad una pagina man è espresso usando la notazione "nome_pagina(sezione)", per esempio, ftp(1) o socket(2). Diverse pagine man possono apparire con lo stesso nome in differenti sezioni del manuale: questo succede quando una chiamata di sistema, un comando utente o un macro package hanno lo stesso nome e generano dunque un contrasto tra loro. Due esempi sono man(1) e man(7), oppure exit(1) ed exit(3).

La sintassi per accedere alle pagine man di una sezione del manuale, che non sia quella predefinita, varia tra le differenti implementazioni di man: nella gran parte dei sistemi

Unix e Unix-like, inclusi Linux ed i vari sistemi BSD, si usa indicare la sezione prima del nome della pagina. Ad esempio, la sintassi per leggere la pagina relativa a printf(3) è:

man 3 printf

NOTA: Esiste anche una pagina relativa al comando man, visibile tramite il comando

man man

4.1.2 Cenni storici

L'*UNIX Programmer's Manual* ("Manuale Unix per Programmatori") fu pubblicato per la prima volta il 3 novembre 1971. Ciò nonostante, le pagine man non furono disponibili in linea fino alla settima edizione di UNIX nel 1979.

A quel tempo, la disponibilità della documentazione in linea, attraverso il sistema delle pagine man, fu considerato un notevole passo avanti. Oggigiorno, ogni applicazione Unix a riga di comando è corredata virtualmente dalla propria pagina man, e la sua assenza è generalmente percepita come indice di bassa qualità del software.

Alcuni progetti, come Debian, si sono incaricati anche di scrivere pagine man per quei programmi e comandi che ne sono privi.

Il formato di default di una pagina man è il *troff*, o con il pacchetto di macro *man* (basato sull'aspetto), o con alcuni sistemi *mdoc* (basati sulla semantica). Questo permette di comporre una pagina man in PostScript, PDF e altri formati, per la lettura o la stampa. Il pacchetto man, nella maggior parte delle distribuzioni moderne di Linux, include il comando *man2html,* il quale permette agli utenti di sfogliare le pagine man attraverso un browser html.

Nel 2010, OpenBSD ha deprecato il *troff* per la formattazione delle pagine man in favore di *mandoc*, un compilatore/formattatore specializzato nelle pagine man con i supporti per output in PsotScripts, HTML, XHTML e per il terminale.

4.2 Accesso alle pagine man

Conoscere il meccanismo di accesso alle pagine man, è importante per poter dar loro correttamente il nome ed identificare la giusta posizione d'installazione. Ogni pagina man è da classificare in una sezione specifica, idenficata da un singolo carattere. Nella tabella qui di seguito riportiamo le sezioni più comuni utilizzate sotto Linux, insieme ai loro nomi di umana comprensione:

Sezione	Nome
1	Comandi dell'utente avviabili da chiunque
2	Chiamate di sistema, cioè le funzioni fornite dal kernel
3	Subroutine, ovvero le librerie di funzioni
4	Dispositivi, ossia i nomi dei file speciali nella directory /dev
5	Descrizioni dei formati dei file, ad esempio /etc/passwd
6	Giochi
7	Miscellanea, ad esempio i pacchetti di macro e le convenzioni
8	Strumenti per l'amministrazione di sistema che soltanto il root può eseguire
9	Un'altra sezione (specifica di Linux) per documentare le routine del kernel
n	(Deprecato) Nuova documentazione, da spostare in una sezione più adatta
o	(Deprecato) Documentazione obsoleta, da tenere qui solo per un breve periodo
l	(Deprecato) Documentazione locale, riguardante questo sistema in particolare

Tabella 4.1:

Il nome del file sorgente della pagina man (quello di input per il sistema di formattazione), è quello del comando, della funzione o del file, seguito da un punto e dal carattere identificativo della sezione. Dunque, identificata la sezione di appartenenza, (nel nostro caso *psort* rientra nella sezione 1) il nome del file sorgente della nostra pagina man risulterà: psort.1

A volte, ma ciò non riguarda il nostro caso, succede che il nome di un file risulti identico al nome di un comando, od anche ad una libreria di subroutine con lo stesso nome. La classificazione in sezioni è il metodo standard per risolvere queste ambiguità: la descrizione dell'ipotetico comando è trattata nel file 'psort.5', mentre l'ipotetica libreria è descritta in 'psort.3'. Per dovere di cronaca abbiamo riportato le sezioni non numeriche (*n*, *o*, e *l*), ma queste, conformemente al File System Standard, sono deprecate, quindi sconsigliate all'utilizzo.

Inoltre, può succedere di incontrare nomi di file sorgente con caratteri terminali aggiuntivi, come in 'xterm.1x' o 'wish.1tk'; in questo modo si vuol indicare che si tratta della documentazione rispettivamente per un programma di X Window o di un'applicazione Tk,2. Alcuni visualizzatori delle pagine man possono far uso di quest'informazione aggiuntiva. Per esempio, nella lista della documentazione messa a disposizione da xman compariranno le voci 'xterm(x)' e 'wish(tk)'.

Inoltre, si faccia attenzione a non usare nomi di programmi, funzioni o file già esistenti. Assicurandosi l'unicità del nome, si evita all'utente di eseguire un programma facendogli leggere la pagina man di un altra applicazione, e viceversa.

Dopo aver capito il procedimento di denominazione del file, il passo successivo riguarda la directory in cui installarlo (ad esempio, come inserirlo all'interno del makefile).

Su Linux ogni pagina di manuale si trova all'interno delle directory elencate nella variabile d'ambiente MANPATH. Gli strumenti adibiti al trattamento della documentazione utilizzano MANPATH nello stesso modo in cui una shell utilizza PATH per localizzare gli eseguibili. Effettivamente, MANPATH ha lo stesso formato di PATH: infatti, ognuna di queste contiene un elenco di directory separate da punti e virgole, se si eccettua il fatto che MANPATH non ammette campi vuoti e nomi di percorso relativi, utilizzando solo nomi assoluti.

Se, però, MANPATH non è stata impostata, verrà utilizzato un valore predefinito che conterrà almeno la directory /usr/man. Per velocizzare la ricerca e mantenere sintetico l'elenco delle directory, le directory indicate in MANPATH (chiamate "directory base") contengono una manciata di sottodirectory chiamate 'man<s>', dove <s> sta per il carattere che descrive la sezione introdotta nella tabella 4.1. Non tutte le sezioni saranno rappresentate da una sottodirectory semplicemente perché non c'è alcuna ragione di mantenere delle sottodirectory vuote. Affinché una pagina man della sezione <s> sia collocata con certezza nel posto giusto, basterà copiarla nella directory /usr/man/man<s>.

Un buon Makefile, però, permetterà all'utente di scegliere la propria directory base, per mezzo di una variabile di make, ad esempio MANDIR. Ad esempio, la maggior parte dei pacchetti GNU possono essere configurati con l'opzione –prefix=/qualsiasi/dir. I manuali verranno perciò installati all'interno della directory base /qualsiasi/dir.

4.3 Il formato di una pagina man

In questa sezione spieghiamo di quali argomenti principali è composta una pagina man. Si assuma come esempio per i contenuti la pagina man realizzata per *psort*, riportata nel capitolo 5 a pagina 47.

Le sezioni che solitamente compongono una pagina man sono:
NOME
SINTASSI
DESCRIZIONE
OPZIONI
FILE
AMBIENTE

DIAGNOSTICA
BUG
AUTORE
VEDERE ANCHE
ESEMPI

La sezione NOME (NAME)

...è la sola sezione obbligatoria. Questa sezione ha anche un formato standard che consiste in un elenco, di nomi di programmi o funzioni, separati da virgole, seguiti da un trattino e da una breve descrizione (di una sola riga di solito) della funzionalità che il programma (o la funzione, o il file) è in grado di fornire. Nel sorgente in formato groff, questa sezione deve apparire come di seguito:

```
.SH NAME psort \- yet another fast stable external sorter
```

La sequenza "\-" è importante: la barra rovesciata (o backslash) è necessaria a distinguere il trattino da quello della suddivisione in sillabe che può comparire sia nel nome del comando che nella descrizione disposta su una riga.

La sezione SINTASSI (SYNOPSIS)

...ha lo scopo di fornire una breve panoramica sulle opzioni messe a disposizione dal programma. Per quel che riguarda le funzioni, questa sezione elenca i corrispondenti file di *include* ed i prototipi, così da informare il programmatore sul tipo degli argomenti di input, sul loro numero e sul tipo dei dati restituiti.

La sezione DESCRIZIONE (DESCRIPTION)

...descrive in maniera eloquente il programma che si sta considerando. Essa si pone l'obiettivo di essere per gli altri programmatori ed utenti, fonte d'informazioni dettagliate ed affidabili. In questo contesto si può esporre l'utilizzo degli argomenti, il formato dei file e quali sono gli algoritmi delle operazioni critiche.

La sezione OPZIONI (OPTIONS)

...fornisce una descrizione di come ogni opzione influenza il comportamento del programma.

La sezione FILE

...elenca i file usati dal programma o dalla funzione. Per esempio, può elencare i file di configurazione, quelli di avvio e/o quelli su cui il programma interviene direttamente in uscita.

La sezione AMBIENTE (ENVIRONMENT)

...elenca tutte le variabili d'ambiente che influiscono sul programma o sulla funzione, descrivendo anche le modalità di quest'interazione. La maggior parte delle variabili contiene nomi di percorso, nomi di file o opzioni di default.

La sezione DIAGNOSTICA (DIAGNOSTICS)

...contiene una panoramica dei messaggi d'errore emessi più frequentemente dal programma descritto, e suggerimenti su come venirne a capo. Non c'è alcuna necessità di descrivere i messaggi d'errore di sistema o i segnali d'errore fatale che eventualmente dovessero comparire durante l'esecuzione, in quanto sono descritti in altri documenti e riguardano l'esecuzione di qualsiasi programma.

La sezione BUG

...informa l'utente delle limitazioni e dei problemi noti a cui non è ancora stato posto rimedio. Questa sezione può anche essere simpaticamente rinominata in TO DO.

La sezione AUTORE (AUTHOR)

...risulta molto utile nel caso in cui l'utente riscontrasse errori grossolani nella documentazione o nel comportamento del programma, e volesse inviare una segnalazione (o una lamentela) all'autore.

La sezione VEDERE ANCHE (SEE ALSO)

...è un elenco di pagine man correlate a quella che si è preso in considerazione, elencate in ordine alfabetico. Per convenzione, questa è l'ultima sezione.

Se le sezioni, a cui abbiamo appena accennato, fossero insufficienti a descrivere il contenuto che si intende fornire, l'autore è libero di aggiungerne ulteriori.

Nel capitolo 6 a pagina 63, riportiamo il file sorgente *psort.1*, a cui abbiamo abbreviato le parti descrittive delle sezioni, per occupare meno spazio.

4.4 Convenzioni sui font

Esistono numerosi pacchetti di macro progettati appositamente per scrivere le pagine di manuale; noi abbiamo optato per il pacchetto di macro tmac.an poiché è il più diffuso, ed a questo pacchetto faremo riferimento in questa sezione.

Il pacchetti si trovano nella directory delle macro di groff /usr/lib/groff/tmac. I nomi dei file sono tmac.<qualcosa>, dove <qualcosa> è l'argomento per l'opzione -m di groff. Groff utilizzerà tmac.<qualcosa> quando verrà data l'opzione '-m <qualcosa>'. Spesso

viene omesso lo spazio tra '-m' e '<qualcosa>', cosicché potremo scrivere 'groff -man' per formattare le pagine man con il pacchetto di macro tmac.an.

Per la redazione della pagina man di *psort* abbiamo utilizzato le macro che accettano argomenti, invece di utilizzare gli operatori diretti dei font (come \fB, \fP, etc); questo per evitare un errore comune: dimenticare il cambio del font alla fine della parola, estendendo erroneamente il grassetto o il corsivo fino al successivo cambio di font.

Le macro di tmac.an forniscono i seguenti formati tipografici:

.B	grassetto (bold)
.BI	grassetto alternato al corsivo (italic)
.BR	grassetto alternato al carattere Roman
.I	corsivo
.IB	corsivo alternato al grassetto
.IR	corsivo alternato al Roman
.RB	Roman alternato al grassetto
.RI	Roman alternato al corsivo
.SM	small: piccolo (di dimensioni pari a 9/10 di quella normale)
.SB	piccolo e grassetto (non piccolo alternato al grassetto)

X alternato con Y significa che gli argomenti di ordine dispari vengono resi nel formato X, mentre quelli pari sono in formato Y.

Ad esempio:

```
.BI "Arg 1 è grassetto, " "Arg 2 è corsivo, " "questo è grassetto, " "e questo corsivo."
```

Le doppie virgolette sono necessarie per includere gli spazi in un argomento; senza di esse non si avrebbe modo di far apparire gli spazi tra i diversi formati tipografici. A tutti gli effetti, quando in un passaggio tra un formato tipografico e l'altro si vuole evitare la comparsa di spazi bianchi, sono necessarie solo le macro con formato tipografico alternato. Ciò vale per la maggior parte dei casi.

Riportiamo ora di seguito un estratto della documentazione man(7), che riporta le regole standard dei diversi formati tipografici:

Benché nel mondo UNIX esistano molte convenzioni arbitrarie per le pagine man, l'esistenza di svariate centinaia di pagine man specifiche per Linux definiscono i nostri standard: nel caso delle funzioni, gli argomenti sono sempre indicati in corsivo, anche nella sezione SINTASSI, mentre il resto della funzione è indicata in grassetto:

.BI "miafunzione(int " argc ", char **" argv);

I nomi dei file sono sempre in corsivo, tranne che nella sezione SINTASSI, in cui i file inclusi sono in grassetto. Quindi andrebbero utilizzati i formati
.I /usr/include/stdio.h
e
.B #include <stdio.h>
Le macro speciali, solitamente indicate in maiuscolo, sono in grassetto:
.B MAXINT
Quando si elenca una serie di codici d'errore, questi vanno resi in grassetto. Di solito, per questo elenco si utilizza la macro .TP (il paragrafo con i tag appesi) come segue:
.TP
.B EBADF
.I fd non è un descrittore di file valido.
.TP
.B EINVAL
.I fd è inaffidabile in lettura

Ogni riferimento ad un'altra pagina man (o al soggetto della pagina man corrente) è in grassetto. Se presente, il numero di sezione del manuale viene reso in carattere roman, senza spazi:
.BR man (7)

Gli acronimi vengono resi meglio usando i caratteri tipografici piccoli, per cui raccomando di fare come negli esempi che seguono
.SM UNIX
.SM ASCII
.SM TAB
.SM NFS
.SM LALR(1)

5 Pagina man di *psort*

5.1 DESCRIPTION

5.1.1 INTRODUCTION

Psort is a fast stable external sorter which manages large data volumes. It comes in two possible ways of use (both available under GPL): the first is in the form of a static-linked, ready-to-use executable, which can order a disk file, viewed as a sequence of r-byte records, according to a key of k-bytes that identifies into each record from the i-th byte. The second possibility of use is as a C library, which allows the user to define their own key comparison functions, as well as to implement new functions of key pre-processing and/or post-processing, all to increase efficiency of the comparisons.

5.1.2 HOW PSORT WORKS

Psort sorts fixed-length records according to an arbitrary infix. It is designed to work on datasets much larger than PC RAMs (typically, from a few dozen to some hundreds GBs) and therefore operates on external memory (i.e. from/to files). Psort exploits a classic two-pass sorting algorithm, but if the file fits entirely in the main memory, only the first pass is used. In the first pass (stage_one), the original file is read and split into runs slightly smaller than the main memory size; each run is sorted and written to disk in an intermediate file. In the second pass (stage_two), runs are merged into a single sorted file.This file must be different from the intermediate file, but it can be the original file.

5.1.2.1 HOW STAGE_ONE WORKS

In stage one, psort sorts and writes to disk the output file (if small enough) or several sorted runs; the size of a run is typically comparable to the size of the main memory (but can be specified by the user). Read/write buffers should be sufficiently large to amortize the disk head movement and rotation cost and yet sufficiently small to leave enough space to sort large (and therefore few) runs - which is critical to reach high speed in the

second pass (see below). The size of the buffers can be specified manually by the user; otherwise psort allocates to them 10% of the total available memory, with a minimum of 16MB per buffer. From the read buffer, data is separated between keys and payloads. Keys are preprocessed so that they can then be compared using integer arithmetic. Since psort needs to keep track of the record associated with each key, some informations are stored in an additional (if required) machine word which extends the key. Blocks of keys and payloads are then sorted using MergeSort (the initial pass actually sorts sequences of 4 items with SelectionSort). Block size can be specified by the user; otherwise, psort computes the size of a block (in terms of items) as the square root (rounded up to the nearest power of two) of the number of items in a full run. Ideally, blocks should (approximately) be the size of the L2 processor cache, so that this phase of the first pass only requires, for each item, a read and a write access to the main memory buffer, and yet blocks are as large (and thus as few) as possible. Blocks are then merged using a k-way merge-tree into a single sorted run that is directly written to the output buffer (with records being restored to their original form with the key embedded into the payload). Ideally, the data in the k-way merge-tree should also fit entirely in cache; the default choice of block size attempts to balance it with the size of the merge-tree.

5.1.2.2 HOW STAGE_TWO WORKS

In the stage_two, psort maintains a sort queue of blocks for each run. As in stage_one, keys are separated from payloads and (possibly) extended. The keys are then merge using a k-way merge-tree and written into a write buffer - from there to disk. Of course, you can enter only a small part of each run in main memory at the same time. This means that, unlike the first pass involving long sequences of reading and writing, the second pass is still composed by long sequences of writing, but by short of reading. This, in turn, means more time lost positioning the disk head, and a lower effective disk bandwidth. Runs are read (again using asynchronous, direct-mapped I/O) into a read buffer of contiguously allocated userspace memory; from there they are moved into sort buffers, one for each run. Psort uses sort buffers whose size varies dynamically to store the data from runs in the first pass. Each buffer consists of a series of microbuffers. When records are inserted into the sort output sequence, they are removed from microbuffer. As soon as a microbuffer is emptied, it can be reused for other purposes, reducing the size of the buffer. When the amount of data in a buffer falls below a certain threshold (chosen as an option by the user) the buffer is "refilled" from the appropriate run. For performance, psort tries to keep the queue lengths in arithmetic progression, so it can fill the empty queue at a constant rate. For this reason, psort allows you to specify the

geometry of space buffer as a parameter g between 0 and 1, where 1 + g specifies the ratio of the maximum buffer size and buffer size B/n in the presence of a static buffer. The maximum and minimum length of the queues are, respectively, (1 + g) times and (1 - g) times the average size of the queue, where g is called "geometric factor". So, for each run is always guaranteed a minimum buffer size (1 - g) B/n. The default value of g is 0.5. Of default a queue is filled to the maximum queue size every time it drops below 80% of the minimum queue size.

5.1.2.3 DATA TYPES

Psort uses fundamentally one data type to perform every key manipulation; this is the ureg_t (stands for 'unsigned register type') data type defined in 'global.h', which actually is a quadword (64 bits = 8 bytes). Note also that psort often measures various quantities (key lengths, key offsets, etc.) using sizeof(ureg_t) as the basic unity; take this into account when tuning (see Section "HOW TO USE PSORT" below).

5.1.2.4 KEY EXTENSION

Key extension allows psort to store information about where the payload associated to a key is located in memory. Psort uses the last 20 bits of the extended key for the position of a record into a block, the previous 18 bits for the block id, and (only in stage two) the previous 10 bits for the run id, therefore needing 48 bits = 6 bytes of extension (rounding to the nearest multiple of sizeof(ureg_t) from above).

5.1.2.5 KEY HANDLING

In order to provide efficiency and flexibility, psort allows to *preprocess* keys, *compare* keys, and then *postprocess* keys. For example (this is implemented by the predefined lexicographical sort) you can efficiently compare strings by first swapping bytes in each machine word (preprocessing), performing an arithmetic comparison (comparing), and then swapping the bytes back (postprocessing). See Section "TUNING PSORT" to understand how to write your own functions.

5.1.3 TUNING PSORT

To tune psort, there are three points you can operate on: parameters, code, and compilation. All the parameters are explained in the OPTION SECTION see below. You should also understand hardware and file system issues so that they do not become the bottleneck (see the end of this Section).

5.1.3.1 TUNING CODE

If the default comparison functions (lexicographical and numerical) do not fit your needs, you can implement your own preprocessing, comparison and postprocessing functions in "inlines.cpp" and adapt the corresponding macro definitions in "global.h". As an example, see the functions implemented in "inlines.cpp". The following snippet of code describes the interface and the semantic you should respect:

```
inline void key_preprocess( void *raw_key, ureg_t *processed_key )
{ This function should copy the first key_length bytes of raw_key to
  processed_key and process it. }
inline ureg_t key_compare( ureg_t *key1, ureg_t *key2 )
{ This function should return 1 if key1 is greater than key2, and return 0
  else. }
inline void key_postprocess( void *raw_key, ureg_t *processed_key )
{ This function should postprocess processed_key and copy its first
  key_length bytes to raw_key. }
```

If you don't need real pre/post-processing (e.g. because your keys are to be compared as numbers), write pre/post-processing functions which only copy (e.g. with 'memcpy()') raw_key to processed_key and vice versa. Don't overwrite the existing functions, but implement your own and then redefine the macros in "global.h", for example:

```
#define KEY_COMPARE_CUSTOM my_compare_function
#define KEY_PREPROCESS_CUSTOM my_preprocess_function
#define KEY_POSTPROCESS_CUSTOM my_postprocess_function
```

Once done, recompile psort and use it specifying –order=custom.

5.1.3.2 TUNING THE COMPILATION PROCESS

CMakeLists.txt uses many flags that affect performance. The arguments passed to the compiler via the CMAKE_C_FLAGS and CMAKE_CXX_FLAGS variables are among the most important, so set the arguments of "-mtune" and "-march" to match your machine's architecture and set the optimization via the "-O" option to at least 3 to guarantee high CPU/RAM performance. If you have a CPU/RAM bottleneck, enable direct I/O by passing "-DDIRECT_IO=on" to cmake on the command line. You should

also enable the support for huge pages; please refer to your OS documentation to enable it in the kernel.

5.1.3.3 HARDWARE

To optimize psort's performance, you should also understand the limitations of the hardware you are using.

There are four fundamental hardware bottlenecks.

1. The hard drives. New hard drives in 2008 have typical peak transfer rates of 50-150MB/s; transfer rate is lower (up to 20-50% lower) closer to the inner rim of the disk (the last used when you write onto an empty disk). Also, this rate can be achieved only if reading *large* sequential chunks of data (a few MBs at least). You can read less than that, but the time to do so will not drop significantly below 10ms or so even if you are reading a single word. Fortunately, one can boost the performance of a single drive by setting up a RAID 0. In practice, this means YOU NEED ABOUT 1GB OF RAM TO SORT 100GBs OF DATA EFFICIENTLY. Beyond that, each doubling of the RAM size allows you to quadruple the amount of data you are sorting efficiently (e.g. you'd need about 4GBs for 1.5TBs of data). If you are planning to sort much larger datasets (seems hard to do with today's drives), you might consider doing a triple pass sort. Software RAID under Linux works well and drains only a negligible amount of CPU resources (of the order of 1%-4%). Setting stripe size for the raid is also important; ideally, it should be as large as possible while remaining significantly below the size of a chunk. We found 32k-128k stripes to work extremely well under most circumstances.

2. The motherboard. Without getting into too many technicalities, in 2008 good SATA 2 motherboards typically support up to 400-600MB/s from RAIDed drives.

3. The main memory. Keep in mind that data is transferred from and to the memory in multiples of a cache line (typically 64 bytes to 256 bytes), so that if you are reading little chunks of data (e.g. 1 double at a time) and they are not contiguous, you will see a *huge* drop compared to the theoretical sustained bandwidth. Also, note that in order to write a cache line that is not in the cache, the system will "implicitly" first read it and then write it, so in some sense you pay the price twice. Taking all this into account psort ends up making what effectively amounts to 10-15 "bytes of access" for each "byte of data" during the first phase of the sort. E.g. if you are reading and writing from disk at 150MB/s (total bandwidth of 300MB/s), you need a peak sustained bandwidth of at least 2000-3000 MB/s for memory not to become the bottleneck (because of conflicts and less-than-perfect cache and RAM access overlap in the current version of psort, you probably want about twice as much). *Important*: this assumes your cache is large

enough. TRY TO MAKE SURE YOU HAVE AT LEAST 1MB OF CACHE (PER CORE) FOR EVERY 1-2GB OF RAM. Less than that and you'll see many more cache misses and many more RAM accesses.

4. The processor cache. Without getting too technical again, what you really want from your processor cache is *size*. Speed should be adequate for most purposes, and larger caches tend to also have good associativity. Stay away from cheapish Celerons or Semprons, and you should be fine. Generally speaking, in this regard single processors are better than multicores in terms of cache*per*core*. As an example, we had no trouble with a 1MB L2 Cache Athlon 2.4Ghz (LE1620) using 2GB of RAM to sort 100 byte records on the basis of the first 10 bytes (had they been 10 byte records, we'd probably have wanted a slightly larger cache - no larger than 2MBs even when supporting 4GB of RAM).

Note that the processor itself is not, generally, a bottleneck (if you see a high "processor utilization" with e.g. top or ps, chances are that it's really a high RAM utilization, that registers in the same way).

5.1.3.4 FILE SYSTEM

Psort has been tested on several Linux 32 and 64 bit systems, with different filesystems. Filesystem can have a serious impact on performance. We recommend xfs or jfs (a statistically insignificant 2-3% slower). Linux's native Ext3fs and Reiser are not optimized for streaming- we found it 30% slower than either xfs or jfs. Stay away from Microsoft stuff.

Note that while psort does not support more than one logical volume, it is designed assuming that you will combine several (typically 2-8, but there's no hard limit) drives into a single RAID 0 volume. This simplifies usage and should have a minimal effect on performance compared to "true" parallel disk usage for most parameters.

5.2 OPTIONS

5.2.1 BASIC USAGE

5.2.1.1 GLOBAL OPTIONS

–input-file=file

Specify input file path

–num-records=N

Specify the number of records to be sorted from the input file

–record-length=n

Specify the length, in bytes, of the input file record.

NOTE: psort does not interpret any part of the input as a delimiter – e.g., carriage returns, end of lines, spacing, etc. are considered as part of the records they belong to. Consequently a file of 1000 bytes, with record-length equal to 10 bytes, it will contain exactly 100 records.

–key-length=k

Specify the length, in bytes, of the input file record keys

–key-offset=f

Specify the offset, in bytes, of the input file record keys. The offset corresponds to the number of bytes before the key inside the record. Obviously, the input of this option can't be greater than the difference between the record length and the key length.

–output-file=file

Specify output file path, defaults to the input file

–order=ord

Specify the sort order; valid values are 'lex', 'num', 'custom', or rather lexicographical, numerical and custom. The numerical order is given by the direct comparison of the keys where they are interpreted as unsigned integers. The lexicographical order pre-processes the keys so to get again a numerical comparison process (in the end, it is obtained an order according to the ASCII code), and finally post-processes the keys to return them to their original state. Lexicographical is the default option. The 'custom' option requires the user to write a custom comparator routine – see above for details.

–verbose-level=L

This option allows the user to set the level of informations that the program write to standard output (0 = no output, 1 = basic output, >1 = debug output). The information

written on standard output, when verbose-level option is set on > 1, is not specified and depends on the version, it is primarly meant for debugging purposes.

–run-file-prefix=pref

Specify the path and the prefix for the run files (e.g. '/tmp/run-')

–stage-one

Execute only stage_one (e.g. for benchmarking/debugging purposes). Default psort executes only the stage_one, if the size of the input file is smaller than the size of the available memory (see below, option –mem), or executes both stages. When psort executes only stage_one, but the input file is larger than the main memory, psort produces several intermediate runs, so if the user wants a sorted output, he has to execute also stage_two. The main purpose of this option is to enable the user to run tests only on stage_one of psort.

–stage-two

Execute only stage_two (default both - see the option above). To run psort with this enabled option, the user needs to have some runs and have to specify their number (see below the option –num-runs), their path and prefix. The main purpose of this option is to enable the user to run tests only on stage_two of psort.

5.2.2 TUNING

The following section contains options that the user can use to improve the program performance. To avoid psort malfunctions, we advise the user to understand the meaning of each option before use it. The author of the manual will specify what follows the options has been written with the aim to provide suggestions about how the user can modify the execution of psort, through indications to the program structures corresponding the options. Indeed, the given explanations were not written with the aim to provide solutions, because the only way to find the best result for the user is verifing through testing.

5.2.2.1 GLOBAL OPTIONS

–mem=amount[bBkKmMgG]

This option limits the amount of the main memory used by psort to the input specified value. The memory amount must be specified as a number, optionally followed by a suffix (the accepted ones are reported in the brackets next the option) that indicates how this value should be interpreted (e.g. 1 b,B = 1 bits; 2 k,K = 2 10^3 bytes; 3 m,M = 3 10^6

bytes; 4 g, G = 4 10^9 bytes) . If this option is disabled, psort uses all the memory made available by the OS.

–run-size=R

This option allows the user to specify the size of the output run of stage_one. The actual size of a run will be computed as the maximum between this value and the available memory (either reported by the OS or specified by the –mem option, see above) minus the size of the auxiliary data structure (I/O buffers + indexes + . . .).

–io-space=fraction

This option allows the user to specify the fraction of main memory (available for psort, see –mem option) that the program allocates for the I/O buffers. The value of this option indicates a fraction of the memory, thus it receives as input a floating point number between 0 and 1; by default this option is set to 0,1. This option is more quick to set up and it gives a more rapid view on the amount of the memory used to support structures, but it is less important than the values of options that configure exactly the space and the number of buffers (–s1-read-buffers; –s1-write-buffers; –s1-read-buffer-size; –s1-write-buffer-size; and the corresponding in stage_two, see below), so if these options are enabled by the user, psort not consider the option –io-space.

–no-detach

If this option is enabled, psort doesn't execute the detachment of the key from the payloads before sorting the records. This option is convenient, in terms of performance, when the record length is less than or equal to the length of the extended key, which is the minimum multiple of 8 bytes greater than the key-length bytes plus the 6 extension bytes.

=> record-length $\leq$ extended_key-length = min(8 n) $\geq$ (key-length(bytes) + 6 bytes)

This option is still EXPERIMENTAL and may result in unpredictable behaviour.

–runs-over-input

This option, when enabled, compels psort to overwrite the input file with the runs in output from stage_one.

–fake-write

This option, when enabled, compels psort to write to /dev/null. It was implemented with debugging/benchmark purpose.

–num-runs=r|t|t|t

This option receives as input an integer, which is interpreted by the program as the number of runs that stage_two will process. From this, it is obvious that the number must be greater than 1. In addition, this option becomes mandatory if only the stage_two is executed (see above the option –stage-two).

We notify to the user that, to explain following options to the best, we will give more practical informations relating the two stages, than the theoretical way of the section DESCRIPTION above.

5.2.2.2 STAGE ONE OPTIONS

The stage_one functions run in loop, until they exhaust the input file. The loop body consists of methods for asynchronous I/O and for the operations on record (processing/sorting). The use of asynchronous I/O allows the machine to perform the I/O procedures and the record operations in parallel, so that (in theory) the execution time of each iteration of the cycle results the maximum time between I/O time and the time of operations:

time(execution) = max[time(I/O) ; time(record_op)]

To obtain this purpose, seems obvious the need to have more of a read buffer (the same is true for the write buffers), so you must have –s1-read-buffers ≥ 2 and –s1-write-buffers ≥ 2. In our opinion, there are no large differences in the psort performance between the values 3 or 4, for both options, and we do not consider very logical a further increase of the values.

–s1-read-buffers=arg\|t\|t
Number of read buffers

–s1-write-buffers=arg\|t
Number of write buffers

To obtain efficient I/O operations, it has to choose a suitable size of the buffer read and write. To understand why, we will try to explain how the buffer size affects the reading time of data from disk. We call D the amount of data to read from disk (it corresponds to the buffer size per the record length - see the option record-length), and B_{disk} the bandwidth of the disk, so the total time to transfer the D data corresponds to the value [$(D/B_{disk}) + t_{seek}$] where t_{seek} is a latency period dependent on the physical support. From this formula we obtain that the effective bandwidth of the disk corresponds to

B_{eff}= D / [(D/B_{disk}) + t_{seek}] = [multiplying above and below by (B_{disk}/D)] = B_{disk} / [1 + (t_{seek} B_{disk}) / D]

from which we can see that higher the number of data, more increases the proximity between the effective bandwidth and the bandwidth of the disk (D→∞ => $B_{eff} \approx B_{disk}$). Of course you can extend this reasoning to the option –s1-write-buffer-size. Warning: the values of input to pass to the two options correspond to the number of records that the buffers have to contain.

–s1-read-buffer-size=arg\|t

Size of the read buffers, in records

–s1-write-buffer-size=arg\|t

Size of write buffers, in records

From an ideal point of view, between the operations of stage_one, the most important point for the efficiency is the size of the k-way merge tree for the sorting of extended keys. It should correspond to the L2 cache size, in order to get for every key a single access for reading and for writing to main memory. To ensure this, the –s1-records-per-block option, which we consider now , must be chosen so that the following condition is met:

K * 2 * (extended-key-length) ≤ L2-cache-size

where K is the number of ways of the tree and depends by:

K = (run-size) / (records-per-block)

As already mentioned, the extended length of a key corresponds to the minimum multiple of 8 bytes greater than the key-length bytes plus 6 extension bytes. The extension bytes contain the memory address of the block and the position in the block of the record from which comes the key. This is to know why it puts a limit on the maximum value that the option we are considering can accept, even if this limit is very high (over a million).

–s1-records-per-block=arg\|t

Number of records per block

5.2.2.3 STAGE TWO OPTIONS

For the descriptions of the following four options, we refer to the corresponding explanations of the options that are in the section of stage_one, because there are no big differences.

–s2-read-buffer-size=arg\|t

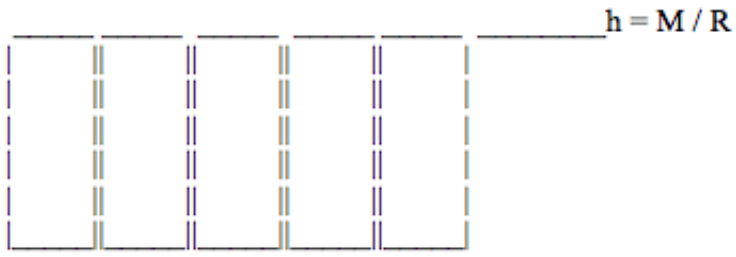
Size of the read buffers, in records

–s2-read-buffers=arg\|t\|t
Number of read buffers

–s2-write-buffer-size=arg\|t
Size of write buffers, in records

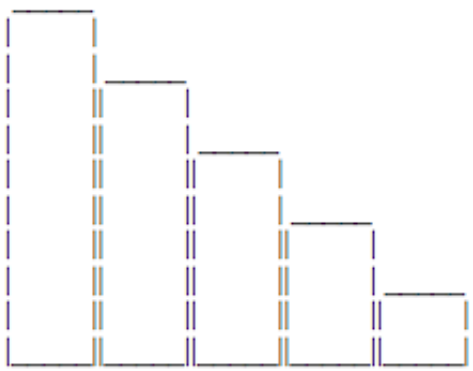
–s2-write-buffers=arg\|t
Number of write buffers

For the explanation of the following options, we'll help us with some graphical interpretations, through simple examples made with the ASCII art, but we still ask the user to imagine what we are stating. First, we denote: with M the size of the available main memory for psort (see the –mem option), with R the number of runs to be merged in stage_two and finally, the value h = M/R which corresponds to the area of available memory for each buffer of each run (thus you must have h $\geq B_{IO}$ where B_{IO} is the maximum size that the buffer can reach - see the option –s2-read-buffer-size). So, if we consider each buffer as a rectangle whose height is h (and whose base is the key length), the user can think the entire memory as a rectangle whose area corresponds to M, and which is formed by the combination of all rectangles devoted to the respective buffer.

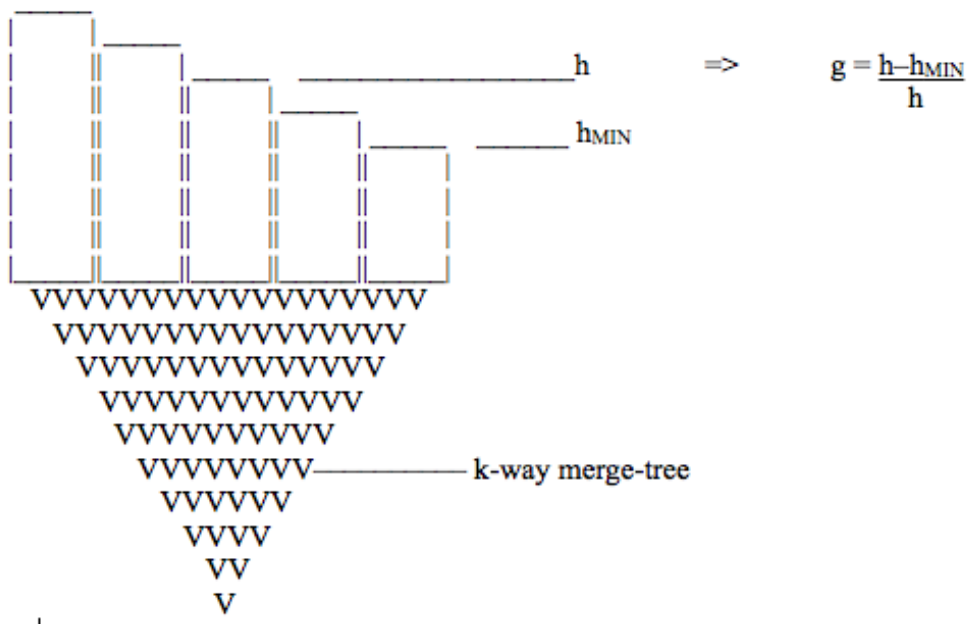


This we have just listed is the simplest version of the division of the memory, but not the most efficient. In fact, the recharging of the buffer is to bring back its height to h when the buffer ends the contained elements, doing so it will block the ordering tree (which takes the keys from the buffer). Moreover, assuming that the records within the buffers are distributed fairly evenly, the userwill find that the buffers will be emptied at the same speed and, therefore, will be nearly empty at the same time. There will be a phase in which the buffers will be reloaded almost sequentially (by blocking the tree for a long time) and others in which practically will work only the ordering tree. So we consider an alternative, and we imagine the memory no longer a rectangle but as a right triangle whose two cathetuses are respectively the base, equal to that of the rectangle, and the height of the triangle, which instead is twice the one of the rectangle (to get the

same area). In reality, however, the triangle is formed by the rectangles representing the buffers, except that here they are willing to decreasing height.



So (always assuming an uniform distribution) when the lowest buffer has reached the 0, it will be refilled to the maximum heigth corresponding to 2h. Nor this version is the most efficiently, because also in this case the refill operation requires to stop the ordering tree, since one of its root has no elements. The two cases that we have just explained are the two limiting cases, corresponding to g = 0 for the rectangle case and to g = 1 for the triangle case, where g is the geometrical factor about which we mentioned in the section HOW STAGE_TWO WORKS (see above) which corresponds the option –s2-geometry below. It corresponds to the ratio g = (h - h_{MIN})/h where h_{MIN} is the minimum height, or the threshold, which must reach the buffer before being refilled to the maximum. By default, the geometric factor is set to 0,5.



–s2-geometry=g|t|t

Geometric factor for the size of run queues

–s2-read-threshold=t|t

This option allows the user to set the minimum treshold value of the buffer before it is refilled to the maximum size, or rather the value h_{MIN} = h (1 – g) (see the previous option). The numeric value, accepted by the option, is interpreted as the megabytes lacking to the stall of the ordering tree; default is set to 1. It has been observed experimentally, that this is particularly influential on the behaviour of the program, and we explain this by observing the two extremes of operation: it can not be too high (very close to the maximum size) otherwise the reading operations are done too often and with little data, it can not even be too low (close to zero) otherwise the reading operations are made too late and so stop the ordering tree.

As explained above (see HOW STAGE_TWO WORKS) buffers are composed of microblocks, for a more efficient reuse of the avalaible space. In fact, every time that a microblock is emptied, it is added to a queue of empty microblocks, which will then be removed to refill the next buffer that reaches the minimum treshold.

–s2-records-per-block=arg|t

Number of records per block

This option allows the user to choose how many records will compose the microblock. Ideally the usershould aim to make the microblocks as smaller as possible, so to waste

less space; but more space will be used for much larger tables, for a longer queue and there is a longer overhead time.

–s2-allocated-blocks=qty|t

This option compels psort to stop at number of microblocks that the user sets. We call M the space of main memory available for psort (see the option –mem) and b the size of each block, which corresponds to (–s2-records-per-blocks * –record-length); the actual number of microblocks is calculated as the minimum of the ratio M/b and the value setby the user –s2-allocated-blocks.

real-blocks-number = min[M/b , –s2-allocated-blocks]

Ideally, the final result should be about 10 times the number of the runs.

5.3 DIAGNOSTICS

The following diagnostic messages can be sent to stderr:

"Invalid or unspecified record length!

Try ‘psort –help’ for more information."

You have entered an incorrect value for the record length, verify that the data you entered corresponds to the length of the records of your input file.

"Unable to open input file!

Try ‘psort –help’ for more information."

psort can not open the input file, check the path entered or file integrity.

"Too few blocks allocated for stage one!

Try ‘psort –help’ for more information."

You have entered an incorrect value for the option - s1-records-per-block, see the section stage_one options of this man page for more details.

"Too many blocks allocated for stage one! Try increasing the block size or decreasing the memory.

See ‘psort –help’ for more information."

You have entered an incorrect value for the option - s1-records-per-block, see the section stage_one options of this man page for more details.

"–s2-geometry requires a parameter between 0 and 1!

See ‘psort –help’ for more information."

You have entered an incorrect value for the geometric factor, see the section stage_two options of this man page for more details.

PORTABILITY

psort works under GNU/Linux and FreeBSD.

AUTHOR

Pietro Vallese & Marco Bressan - <psort@dei.unipd.it>

6 Estratto di psort.1

```
.TH PSORT 1 "APRILE 2011" Linux "User Manual"
.SH NAME
 psort \- yet another fast stable external sorter
.SH SYNOPSIS
.B psort --input-file=file --record-length=n [
.I global_options
.B ] [
.I stage_one_options
.B ] [
.I stage_two_options
.B ]
.SH DESCRIPTION

.SH      INTRODUCTION
        Psort is a fast stable external sorter which manages
 large data volumes. It comes in two possible ways of use
 (both avaible under GPL):
        The first is in the form of a static-linked, ready-to-use
 executable, which can order a disk file, viewed as a
 sequence of r byte records, according to a key of k-bytes
 that identify into each record from the i-th byte.
        The second possibility of use is as a C library, which
 allows the user to define their own key comparison
 functions, as well as to implement new functions of key
 pre-processing and/or post-processing, all to increase
 efficiency comparisons.

.SH      HOW PSORT WORKS
 Psort sorts fixed-length records according [...]
```

```
.SH      HOW STAGE_ONE WORKS
 In stage one, psort sorts and writes to disk [...]

.SH      HOW STAGE_TWO WORKS
 In the stage_two, psort maintains a sort [...]

.SH      DATA TYPES
 Psort uses fundamentally one data type to [...]

.SH      KEY EXTENSION
 Key extension allows psort to store [...]

.SH      KEY HANDLING
 In order to provide efficiency and [...]

.SH TUNING PSORT
 To tune psort, there are [...]

.SH      TUNING CODE
 If the default comparison functions [...]


.SH      TUNING THE COMPILATION PROCESS
 CMakeLists.txt uses many flags that affect [...]

.SH HARDWARE
 To optimize psort's performance, [...]

.SH      FILE SYSTEM
 Psort has been tested on [...]


.SH OPTIONS
.SH      BASIC USAGE
.SH              GLOBAL OPTIONS
```

```
.IP --input-file=file
 Specify input file path
.IP --num-records=N
 Specify the number of records to be sorted from the input file
.IP --record-length=n
Specify the length, in bytes, of the input file record.
[...]

.SH      TUNING
 The following section contains options that the [...]
.SH              GLOBAL OPTIONS
.IP --mem=amount[bBkKmMgG]
 This option limits the amount of the main [...]
.IP --run-size=R
 This option allows the user to specify the [...]
.IP --io-space=fraction
 This option allows the user to specify the
[...]

.SH              STAGE ONE OPTIONS
 The stage_one functions [...]
.IP --s1-read-buffers=arg\t\t
 Number of read buffers
.IP --s1-write-buffers=arg\t
 Number of write buffers

[...]

.SH              STAGE TWO OPTIONS
 For the descriptions of the [...]
.IP --s2-read-buffer-size=arg\t
 Size of the read buffers, in records
.IP --s2-read-buffers=arg\t\t
 Number of read buffers

[...]
```

```
.IP --s2-geometry=g\t\t
 Geometric factor for the size of run queues
.IP --s2-read-threshold=t\t
 This option allows the user to set [...]

[...]

.SH DIAGNOSTICS

 The following diagnostic messages can be sent to stderr:

 "Invalid or unspecified record length!
 Try `psort --help' for more information."

              You have entered an incorrect value for the
              record length, verify that the data you entered
              corresponds to the length of the records of your
              input file.

[...]

.SH PORTABILITY
 psort works under GNU/Linux and FreeBSD.

.SH AUTHOR
 Pietro Vallese & Marco Bressan \- <psort@dei.unipd.it>
```

7 User Guide

Una user guide è un documento di comunicazione tecnica, il cui contenuto riguarda principalmente le procedure di utilizzo di un determinato software. Quindi, tale guida fornisce all'utente del software precise indicazioni per una corretta e migliore fruizione del prodotto. Possiamo sinteticamente spiegare la sua funzione, dicendo che si pone come obiettivo quello di dare una risposta completa e concisa alla domanda "come devo fare...?". Le caratteristiche di un prodotto software, però, vengono illustrate più dettagliatamente nel manuale di riferimento. Esso, infatti, contiene le descrizioni puntuali e specifiche di ogni finestra di dialogo, di ogni campo, di ogni scheda e di ogni pulsante. Come in precedenza, possiamo semplificare il concetto, affermando che un manuale di riferimento deve rispondere alla domanda "che cosa è x?".

Molto spesso la user guide ed il manuale di riferimento vengono forniti in coppia. E questo per fare in modo che le spiegazioni che compaiono nella guida utente, risultino sintetizzate e di più facile comprensione. Facendo così, però, ne consegue la necessità di integrare le informazioni, introducendo dei riferimenti incrociati al manuale di riferimento. In questo modo, i due documenti si compensano a vicenda: ognuno completa le informazioni contenute nell'altro.

La maggior parte delle guide utente facilita la comprensione del proprio contenuto, affiancando alla guida scritta delle immagini esplicative. Tale sinergia semplifica notevolmente la lettura delle guide a beneficio dell'utente. Nel caso di applicazioni informatiche, per favorire le stesse agevolazioni, si è soliti includere gli screenshot delle interfacce grafiche del programma, oppure i diagrammi delle funzioni.

7.1 Contenuti di una guida utente

Una guida utente, generalmente, si compone delle seguenti sezioni:

- Una pagina di copertina
- Una pagina destinata a presentare il titolo
- Una pagina riservata ai diritti sul copyright del prodotto software

- Una prefazione contenente i dettagli dei relativi documenti e le informazioni riguardanti le modalità di navigazione nel manuale
- Una pagina dei contenuti
- Una guida che fornisce almeno una conoscenza di base dell'utilizzo delle funzioni principali del sistema
- Una sezione dedicata alla risoluzione dei problemi (in particolare, possibili errori od imprevisti che possono eventualmente verificarsi); collegata al proprio problema, viene avanzata la proposta di risoluzione
- Una sezione FAQ (Frequently Asked Questions)
- Una pagina rivolta a prestare ulteriore aiuto all'utente in difficoltà, e contenente i contatti utili
- Un glossario e, per i documenti di grandi dimensioni, un indice

7.2 Analisi del pubblico

Prima di iniziare a scrivere una guida, è bene prestare attenzione al tipo di target a cui ci si rivolge. Infatti, la guida si rivela di scarso valore se non è in grado di rispondere alle domande poste dagli utenti. Per questo, è necessario creare delle categorie di pubblico, e, di conseguenza, delle relative regole di compilazione della guida.

Ad esempio, una modalità di classificazione del target si compie attraverso l'analisi del ruolo dell'utente. Come riportiamo di seguito:

- Data entry impiegato
- Supervisore
- Amministratore di sistema
- Service desk operator

Questo metodo di classificazione può risultare utile per:

- creare una documentazione differente a seconda delle esigenze di ogni tipo di utente
- intuire il livello di abilità degli utenti (ma ciò non è sempre verificabile, infatti, ad esempio, un amministratore di sistema solitamente è un esperto, ma un impiegato data entry può essere sia un esperto che un principiante)

- comprendere quale linguaggio utilizzare; infatti, il linguaggio utilizzato dipende dai destinatari a cui si rivolge, e può essere sia un gergo tecnico e criptico ai più, che linguaggio accurato ed esplicativo.

Infine, è importante ricordare di non omettere passaggi logici fondamentali per i meno esperti. A volte risulta necessario inserire delle ovvietà, con lo scrupolo di pensare che chi legge la guida potrebbe non essere a piena conoscenza dell'argomento. Tuttavia, non bisogna nemmeno eccedere nell'altro verso, dando informazioni che non sono necessarie. Se una frase è sufficiente a spiegare il contenuto, è bene non includere una pagina di schermate .

8 APPENDICE - Software utilizzati

Per redigere questo documento, siamo ricorsi all'utilizzo dei seguenti programmi:

8.1 LyX - The Document Processor

Licenza

GNU General Public License, version 2, June 1991. Copyright © 1989-1991 Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA

Descrizione

LyX è un software open source con interfaccia grafica per elaborare testi. È particolarmente indicato per redigere documenti scientifici contenenti formule o tabelle, grazie all'implementazione basata su LaTeX. Il programma è disponibile per i sistemi operativi Unix, GNU/Linux, Win32 e MacOS X.

Utilizzo

Abbiamo utilizzato questo software per la compilazione e formattazione del presente elaborato. Inoltre, è nostra opinione consigliarlo per la scrittura della documentazione esterna di un software, come ad esempio l'user guide.

8.2 Xcode

Licenza

Descrizione

Xcode è un Integrated Development Environment (IDE) sviluppato da Apple per agevolare lo sviluppo di software per MacOS X e iOS. La sua interfaccia unificata permette di passare agevolmente dal comporre il codice sorgente, al debugging, ed al progettare il design dell'interfaccia dell'utente, il tutto all'interno della stessa finestra.

Utilizzo

Abbiamo utilizzato questo software come editor di testo per file di codice sorgente e per la compilazione del file "psort.1".

8.3 Pages

Licenza

Descrizione

Pages è un Word Processor intuitivo e facile da usare, capace di elaborare testi ed impaginare documenti. L'applicazione è stata sviluppata da Apple per le piattaforme MacOS X e iOS, e fa parte della suite di software iWork.

Utilizzo

Abbiamo utilizzato le componenti grafiche di questo software per elaborare la maggior parte delle figure presenti in questo elaborato.

Ringraziamenti

Durante la stesura di questo lavoro, che segna il primo traguardo della nostra carriera universitaria, abbiamo avuto il supporto ed il sostegno di molte persone, che ora passiamo a ringraziare qui di seguito:

Professor Enoch PESERICO

Innanzitutto, vogliamo ringraziare il professore che ha cortesemente accettato di ricoprire il ruolo di nostro relatore, e per la disponibilità dimostrataci.

Dottor Marco BRESSAN

Vogliamo ringraziare il nostro correlatore per averci supportato, e sopportato, nello sviluppo e correzione del presente elaborato.

Marco SERPELLONI e colleghi

Ringraziamo sentitamente i nostri colleghi che ci hanno aiutato durante le fatiche di questo percorso accademico.

Maria VALLESE

Ringraziamo nostra sorella, perchè anche dopo lunghe e molto spesso inutili battibeccate, alla fine c'è sempre nei momenti di bisogno di bisogno. In particolare per il supporto linguistico fornitoci nella correzioni grammaticali e sintattiche di questa tesina.

Flavia e Giulio VALLESE

Ringraziamo sinceramente i nostri genitori per il sostegno fornitoci, sia economico che morale. Senza il loro incoraggiamento e la loro perseveranza, probabilmente, oggi non saremmo qui.

Lorenzo VALLESE

Vogliamo ringraziare anche nostro fratello, perchè un fratello maggiore è un fratello maggiore.

Fabio CARPENE, Matteo RAINERI, Matteo LASEN ed amici tutti

A tutti i nostri amici, semplicemente grazie.

APPLE, Inc.

Infine, vogliamo simbolicamente ringraziare, citandola in conclusione del nostro lavoro, l'azienda che è stata l'ispirazione che mi ha spinto a scegliere questa facoltà. Inoltre, ha "sostenuto" tutto il nostro cammino accademico con i suoi prodotti, sempre eccellenti ed all'altezza di ogni situazione.

Bibliografia

[1] Sito ufficiale del Dipartimento di Ingegneria dell'Informazione - Università di Padova:
`http://www.dei.unipd.it/`

[2] Sito ufficiale del Sort Benchmark
`http://sortbenchmark.org/`

[3] Bertasi P., Bressan M. e Peserico E., 2008. *psort, yet another fast stable external sorting software*. Disponibile su
`http://sortbenchmark.org/psort.pdf` [Data di accesso 28/09/2011]

[4] Bertasi P., Bressan M. e Peserico E., 2009. *psort 2009*. Disponibile su `http://sortbenchmark.org/psort2009.pdf`
[Data di accesso 28/09/2011]

[5] Bertasi P., Bogo F., Bressan M. e Peserico E., 2011. *psort 20*11. Disponibile su
`http://sortbenchmark.org/psort_2011.pdf`
[Data di accesso 28/09/2011]

[6] Sito ufficiale del progetto Doxygen
`http://www.stack.nl/~dimitri/doxygen/index.html`

[7] Van Heesch D., 2011. *Doxygen manual 1.7.5*.1. Disponibile su
`http://www.stack.nl/~dimitri/doxygen/download.html#latestman`
[Data di accesso 28/09/2011]

[8] Imamura M., 2002. *Using Doxygen.* Disponibile su
`http://old.lugatgt.org/articles/doxygen/` [Data di accesso 28/09/2011]
o nella sezione articles del sito ufficiale di Doxygen

[9] Peterchen, 2003. *10 Minutes to document your code.* Disponibile su
`http://www.codeproject.com/KB/tips/doxysetup.aspx`
[Data di accesso 28/09/2011]
o nella sezione articles del sito ufficiale di Doxygen

[10] Schweikhardt J., 2002. *Linux Man Page Howto,* traduzione di Teatini F. Disponibile su
`http://www.pluto.it/files/ildp/HOWTO/Man-Page/index.html`
[Data di accesso 28/09/2011]

[11] Sito ufficiale di Lyx
`http://www.lyx.org`

[12] Sito ufficiale di Pages
`http://www.apple.com/it/iwork/pages/`

[13] Sito ufficiale di XCode
`http://developer.apple.com/technologies/tools/`

[14] AAVV, 2011. *Software Documentation*, Wikipedia, the free encyclopedia. Disponibile su
`http://en.wikipedia.org/wiki/Software_documentation`
[Data di accesso 28/09/2011]

[15] AAVV, 2011. *Doxygen*, Wikipedia, the free encyclopedia.
Disponibile su `http://it.wikipedia.org/wiki/Doxygen`
[Data di accesso 28/09/2011]

[16] AAVV, 2011. *Man page*, Wikipedia, the free encyclopedia.
Disponibile su `http://en.wikipedia.org/wiki/Man-page`
[Data di accesso 28/09/2011]

[17] AAVV, 2011. *User guide*, Wikipedia, the free encyclopedia.
Disponibile su`http://en.wikipedia.org/wiki/User_guide`
[Data di accesso 28/09/2011]