**Politecnico di Torino**

**A.A. 2007/2008**



**Progetto di**

**Programmazione di Sistema**

***Sistema di monitoraggio remoto* Volume Logger**

di

**Alessandro Giambruno**

**Fabio Aulico**

***Prof.: Giovanni Malnati***

# Introduzione

Il sistema implementato si propone come soluzione alla consegna e alle richieste del progetto del corso. E’ stato, dunque, sviluppato un sistema per il monitoraggio remoto di attività di I/O su dischi presenti all’interno di elaboratori connessi tra loro formando una rete locale.

# Architettura

L’architettura progettata è aderente, almeno ad alto livello, ai specifici requisiti presenti nelle specifiche del progetto del corso.

## Architettura ad alto livello

In generale l’applicazione è composta da più moduli che interagiscono tra di loro al fine di assolvere alle varie funzionalità richieste.

**Minifilter Driver**

**Filter Driver**

**User application**

**(service)**

**HW**

**Remote user application**



*Local Network* locale



**Service interaction**

## Architettura a basso livello

I vari componenti sono stati realizzati integrando diverse tecnologie.



**Net**



*Local Network* locale

**GUI**

**Remote user application**

**L/S**

**User Application**



**Minifilter Driver**

**Dynamic Library DLL**

**Windows Service**

**Console Application**

**Service interaction**

### Minifilter

Il minifilter che abbiamo utilizzato per il nostro progetto è basato sul Minispy presente nel Windows Drivers Development Kit, in quanto questo assolveva a tutte la funzionalità richieste nelle specifiche. E’ stata effettuata un’operazione di scrematura a monte delle informazioni inviate dal Minispy (come ad esempio gli argomenti) , in quanto non espressamente richieste nel progetto in esame: in questo modo, è stato possibile ridurre la mole di dati che in alcuni casi, come quando si poneva in esame il drive principale (quello che ospita il sistema operativo), diventava decisamente onerosa da gestire. Si è in particolare tenuto conto, in questa operazione di scrematura, di mantenere una coerenza nelle struct con i sistemi IA64.

### User application

L’applicazione nello spazio utente è stata, invece, realizzata in C++ e la sua logica è stata incapsulata sotto forma di un servizio Win32. Inizialmente, si era pensato di racchiudere più funzionalità lato driver, ma le difficoltà riscontrate nel compilare librerie standard all’interno del DDK, hanno portato ad una soluzione alternativa del problema: è stato pensato pertanto di sviluppare una DLL, compilata sotto WinDDK, che fungesse da ponte tra il servizio WIN32, compilato sotto Visual Studio, ed il driver stesso. In questo modo non è stata pregiudicata la possibilità di comunicare con il minifilter, e non si è rinunciato alla semplicità di utilizzo di librerie esterne che il Visual Studio offre. Vengono elencate di seguito le funzioni che vengono esportate nella DLL:

* DllMain: Questa funzione è la funzione di entry della DLL. L’unica funzionalità è quella di ritornare un BOOL se questa è stata chiamata correttamente.
* Connect: Questa è la funzione che serve a connettersi al Minispy. Utilizza la funzione FilterConnectCommunicationPort , e ritorna l’HANDLE al Minifilter se questo viene trovato correttamente.
* AttachDevice: Questa funzione viene utilizzata per comunicare al Minispy che si vogliono intercettare le operazioni relative ad un certo drive. Vuole in ingresso una stringa nel formato “c:”, e ritorna un BOOL a seconda che l’operazione sia o no andata a buon fine.
* DetachDevice: Funzione duale della precedente. Serve a disconnettere un drive dall’intercettazione del Minispy e ritorna un BOOL a seconda della bontà dell’operazione.
* GetPacket: Questa è la funzione principale della DLL. Si occupa di ricevere in ingresso un buffer di caratteri, e di riempirlo secondo il formato che è spiegato più in dettaglio nella sezione successiva.

Come descritto precedentemente, il servizio di occupa di chiamare dinamicamente le funzioni della DLL, e di utilizzarle per ottenere i pacchetti dal Minifilter. All’avvio del servizio, viene effettuata automaticamente soltanto la connect al minifilter, pertanto nessun device viene intercettato. Per attaccare un device al Minispy, bisogna utilizzare pertanto l’applicazione di supporto che comunica con il servizio e tramite messaggi user-defined, indica quale operazione di attach o detach si vuole effettuare. Tale applicazione si presenta come segue:



Per semplicità, si è scelto di considerare soltanto i drive la cui lettera va dalla “a” alla “f”. Inoltre, non viene effettuato alcun controllo sulla correttezza dell’operazione effettuata: se si tenta di effettuare un’operazione di attach su un drive non presente oppure già attaccato al Minispy, o se si tenta di effettuare un’operazione di detach su un drive non attaccato, l’applicazione continua a funzionare normalmente e l’operazione richiesta viene semplicemente ignorata.

Il servizio, dopo aver ricevuto i pacchetti dal Minispy, si occupa di inviarli in Multicast alle applicazioni che sono in ascolto.

Per quanto riguarda la FSM relativa al servizio, bisogna effettuare una precisazione sull’operazione di pause e continue. Questa infatti viene vista come una pausa “logica”: non vengono chiusi handle né i socket, ad il thread principale del servizio non viene sospeso; ma semplicemente, il servizio resta appeso ad un semaforo che gli impedisce di continuare il suo lavoro di retrieve-and-send: l’operazione di continue serve appunto a mandare un evento ed a sbloccare il semaforo, permettendo al servizio di continuare il proprio lavoro.

### Il formato del record del log

|  |  |
| --- | --- |
| **SEQ\_NUM**  **(ulong, 4 byte)** | **TimeStamp**  **(char[], *HH:MM:SS:mmm*, 12 byte** |
| **FILE\_NAME**  **(char[], \0 terminated, <256 bytes)** | **PROCESS\_NAME**  **(char[], \0 terminated, <256 bytes)** |
| **ACTION**  **(char[], \0 terminated, <64 bytes)** | **ACTION\_DETAIL**  **(char[], \0 terminated, <64 bytes)** |

Viene qui descritta la struttura dati che viene scambiata tra il driver ed il servizio. Come si può vedere nella figura sopra indicata, il minispy innanzitutto invia dei numeri di sequenza ed un timestamp di quando l’operazione in esame è avvenuta: in questo modo è possibile avere una visione del numero di operazioni che effettivamente vengono intercettate dal driver ed inviate al servizio. Altra informazione che viene direttamente inviata dal driver è quella relativa al file che è stato modificato, che è rappresentato da una stringa terminata da uno /0. Per quanto riguarda il nome del processo, il formato indicato nell’immagine soprastante è relativa al pacchetto che viaggia in multicast dal servizio: in realtà, dal driver il servizio riceve non il nome del processo che sta effettuando l’operazione di I/O, ma il suo pid. Questo perché, come detto in precedenza, all’interno del DDK, utilizzare delle funzioni per il retrieve del nome di un processo a partire dal suo PID, è risultato essere un’operazione abbastanza difficile. Pertanto, il pacchetto che riceve il servizio non è lo stesso di quello che riceve l’applicazione: il servizio di occupa di trovare il nome del processo a partire dal pid (e lo può fare dato che Minispy e servizio risiedono sulla stessa macchina), e si occupa di inviare alle applicazioni in ascolto il pacchetto correttamente modificato. Infine, vengono inviate le informazioni sul tipo di operazione di I/O che è stata intercettata: è stato pensato di inserire non solo il major code, ma anche il relativo minor code, poiché in alcune operazioni, come quelle sulle directory, solo dal primo non era possibile evincere di quale operazione si trattasse.

|  |
| --- |
| **HOST\_NAME** |
| **SEQ\_NUM** |
| **TimeStamp** |
| **FILE\_NAME** |
| **PROCESS\_NAME** |
| **ACTION** |
| **ACTION\_DETAIL** |

### Remote application

L’applicazione remota è stata realizzata in C#. Consta principalmente di un’interfaccia utente quanto più possibile semplificata ed intuitiva, di un modulo dedicato al recupero dei pacchetti dalla rete locale e un modulo per la gestione dei dati acquisiti (salvataggio, caricamento, modifica, cancellazione).

L’interfaccia grafica è stata resa il più interattiva possibile e assicura dei tempi di risposta ottimali anche nel caso di processa mento di grandi moli di dati. L’interfaccia non perde di *responsiveness* nemmeno per quantità di dati dell’ordine del milione di record[[1]](#footnote-2). Questo comportamento è garantito dalla natura *multithreaded* dell’applicazione stessa. Infatti, in aggiunta al flusso di esecuzione per il *rendering* grafico dei *form* vi è un *thread* parallelo che si occupa di fare il *dispatch* dei pacchetti provenienti dalla rete. Un sommario diagramma di sequenza è mostrato in figura. L’oggetto che ci occupa della logica di recupero dalla rete (della classe DataRetrieval) viene istanziato all’avvio dell’applicazione. Soltanto dopo che l’utente ha premuto il pulsante “Start” (o per vie alternative attraverso menu o tasti *shortcut*) viene però creato il nuovo *thread* con il comportamento di del metodo StartRetrieval().



Il metodo implementa la logica di ricezione di dati ottenuti dalla rete ed il suo comportamento dipende da come sia impostata la proprietà IsRunning. Quest’ultima è utile per avviare/bloccare temporaneamente la ricezione dei dati senza terminare il *thread*. StartRetrieval() si comporta in questo modo non appena IsRunning è settato a true:

* Se StartRetrieval() è appeso alla *socket* in attesa di dati, vi resta e alla ricezione del primo pacchetto utile lo scarta e si appende al semaforo in attesa che IsRunning torni true e il semaforo venga segnalato.
* Se StartRetrieval() non è appeso alla *socket*, si appende direttamente al semaforo in attesa che "IsRunning" torni true e il semaforo venga segnalato.

Una volta ottenuto un pacchetto dalla rete il metodo si occupa di recuperare i tipi in base al protocollo scelto per la comunicazione dei dati. Infine, una volta recuperate le informazioni di interesse, aggiunge le informazioni di sua competenza (il nome dell’host, infatti, non è incluso esplicitamente nel pacchetto pervenuto) e viene utilizzato il riferimento al MainForm per chiamare il delegato di tipo AddRecordDelegate invocato tramite la funzione BeginInvoke(). Infatti il thread non è il proprietario del form e altrimenti non potrebbe effettuare l’aggiornamento grafico di quest’ultimo. La classe DataReceiver contiene un *flag* (proprietà HasBeenLaunched) non modificabile che traccia se StartRetrieval() è stato chiamato o meno. In questo modo si può discriminare, all'avvio del *thread* principale, se il *thread* è stato già lanciato o meno.

Il metodo Kill() invece si occupa di chiudere la *socket* di ricezione e conseguentemente di sbloccare il metodo StartRetrieval()e di permettere al *thread* di terminare correttamente. E’ stato, infatti, scelto espressamente di attendere la morte di tutti i thread prima della chiusura dell’applicazione e questa è stata considerata la migliore tra le scelte disponibili per la terminazione corretta del *thread,* evitando metodi insicuri come ad esempio l’Abort(). Se la chiusura della socie dovesse fallire per qualche motivo è comunque previsto un *timeout* (J\_TIMEOUT) oltre il quale l’applicazione termina in modo brusco (questa condizione limite non dovrebbe mai verificarsi). Il metodo Kill() va chiamato dopo il metodo StartRetrieval() e non è idempotente (se chiamato più volte tenta di chiudere più volte la *socket*).

Al termine del flusso di esecuzione descritto (dopo il metodo Kill()) l’oggetto è di nuovo pronto per una nuova iterazione, chiamando di nuovo StartRetrieval(), non è cioè necessario re istanziare l’oggetto.

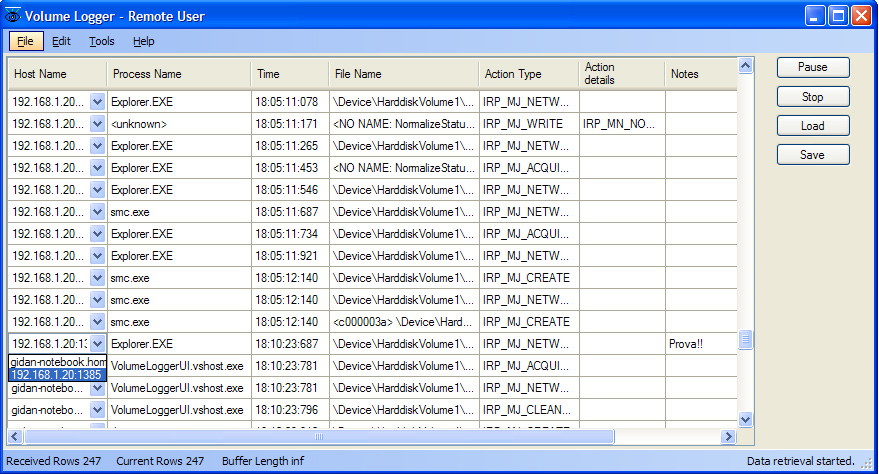
# Installazione dell’applicazione

La procedura di installazione non è particolarmente complessa ma richiede comunque l’integrazione delle diverse tecnologie e una corretta interazione tra loro. In particolare occorre principalmente seguire i seguenti passi:

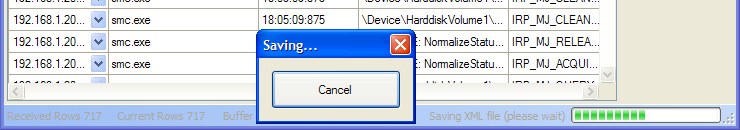
* Compilazione del driver .sys, semplificata tramite il file *batch* run.bat, ottimizzato per una esecuzione all’interno dell’ambiente DDK Windows XP Checked. Prima di eseguire il file run.bat occorre cambiare al suo interno la variabile locale TESINAPATH con il percorso effettivo della cartella contenente il progetto. Al suo interno il file run.bat esegue i seguenti passaggi:
  + cd %TESINAPATH% va alla directory del progetto
  + build –W building in ddk con visualizzazione di *warning*.
  + start minispy.inf installazione del driver nel sistema grazie al file .inf
  + cd user/objc\*/ i\* navigazione all’interno delle directory del progetto
  + net stop minispy ripristino eventuale del driver
  + net start minispy avvio del driver
* Compilazione della DLL che implementa le funzionalità di comunicazione con il minifilter
* Compilazione ed installazione dell’applicazione/servizio VLService. Una volta eseguiti con successo i passaggi precedenti occorre adesso compilare l’applicazione che si installerà come servizio di Windows.
  + Il servizio può essere compilato direttamente all’interno dell’ambiente Visual Studio, ma non va ovviamente lanciato. Il progetto che lo implementa è VLService.
  + Prima di installare e comunque prima di eseguire il servizio occorre affiancare la DLL per la comunicazione con il *minifilter* nella stessa cartella del file VLService.exe che modella il servizio di Windows. Infatti non essendo più all’interno dell’ambiente Visual Studio, Windows non troverebbe la DLL. Alternativamente si può porre la DLL all’interno di della cartella windows\system32.
  + Occorre usare i due progetti di Visual Studio di supporto per installare/disinstallare il servizio. In particolare il progetto di installazione richiede che sia inserito il path corretto dell’eseguibile che implementa il servizio di Windows. Quest’ultimo può essere modificando cambiando la macro SERVICE\_PATH e ricompilando il progetto. Entrambi i progetti si trovano all’interno del progetto di Visual Studio VLService.
  + Occorre lanciare il servizio tramite il comando opportuno all’interno della console services.msc di Windows.
* Compilazione ed esecuzione dell’applicazione remota. Questo è il passo relativamente più semplice in quanto il progetto può essere pensato come stand-alone per quanto riguarda dipendenze da altri oggetti/risorse esterne al progetto in C# di Visual Studio. Si può compilare ed eseguire direttamente all’interno del Visual Studio senza particolari accortezze.

# Descrizione delle caratteristiche e funzionamento

Una volta installato il sistema nelle macchine di interesse e dopo aver configurato il servizio attraverso l’apposita applicazione di supporto, se i passi precedenti hanno avuto successo e il *multicast* è permesso nella rete locale in cui si è collegati, è possibile immediatamente osservare che vi è una trasmissione di pacchetti dalle macchine dove sono installati driver e servizio e l’applicazione remota in ascolto.



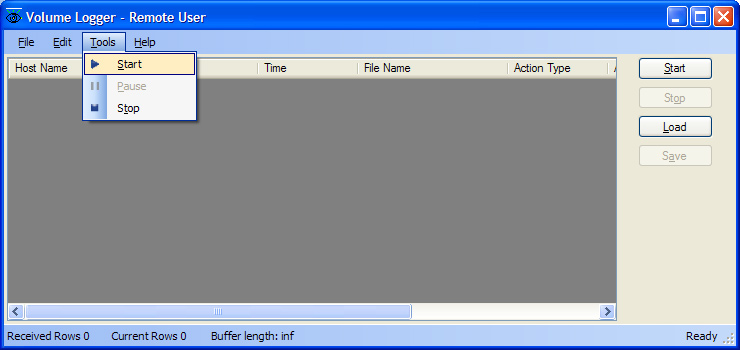
I dati vengono presentati in forma tabellare, sotto forma di *record* non modificabili (ad eccezione del campo Notes, nel quale l’utente può inserire qualsiasi informazione di suoi interesse). Se è possibile risolvere anche il nome dell’*host*, quest’ultimo viene visualizzato in aggiunta all’indirizzo IP: in questo caso è comunque possibile visualizzare l’indirizzo IP aprendo l’apposita casella combinata.



Nella barra di stato, infine, sono presenti alcune principali statistiche sui pacchetti ricevuti, informazioni sullo stato del programma e, in caso di caricamento/salvataggio, una barra di avanzamento ad indicare lo stato di tale azione.

## Comandi

Per lanciare la ricezione di dati si può utilizzare l’apposito *menù* o il bottone dedicato.



Per i principali comandi è stato previsto, infatti, un doppio controllo. E’ ininfluente agire su uno o sull’altro visto che le operazioni scatenate sono “centralizzate”.

I principali comandi sono i seguenti:

* **File**
  + **Close**

Chiude l’applicazione. Equivalente a chiudere il *form* tramite la **X** rossa. E’ richiesta conferma.

* + **Load**

Caricamento da file XML di *record* precedentemente salvati, controllando se il file è valido o corrotto.

* + **Save**

Salva i *record* ottenuti su un apposite file XML e include nella stessa directory la DTD che controlla la validità dei dati salvati.

* **Edit**
  + **Clear**

Elimina tutti i *record* collezionati. Se i *record* non sono salvati prima di questa operazione vengono persi.

* + **Options**

Accede alle opzioni principali dell’applicazione.

* **Tools**
  + **Start**

Inizio del *retrieval* dei dati.

* + **Pause**

Pausa del *retrieval* dei dati. La *socket* non viene disimpegnata.

* + **Stop**

Stop del *retrieval* dei dati. La socket viene disimpegnata in attesa di un nuovo avvio con *start*.

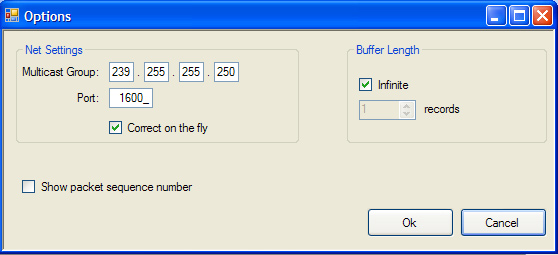
* **Help**
  + **About**

Apre una finestra informativa sull’applicazione.

## Opzioni dell’applicazione remota

E’ possibile modificare alcune opzioni di base dell’applicazione remota direttamente in *runtime* scegliendo l’apposito *item* del menù Edit.

In particolare è possibile modificare il gruppo multicast cui registrarsi, la porta cui mettersi in ascolto (una modifica a questi due parametri richiede il riavvio del processo di ricezione dei dati, quindi conseguentemente include uno *stop* automatico del *retriva* di dati dalla rete). Inoltre è possibile selezionare la lunghezza del buffer di ricezione. Lasciare questo valore ad Infinite esclude la possibile perdita di pacchetti ma allo stesso tempo comporta un carico di RAM che potrebbe risultare notevole in dipendenza al carico generato dalle macchine nella rete e alla durata dell’acquisizione. Settare il Buffer Length ad un valore finito lo rende un buffer circolare di ampiezza definita dall’utente.



E’ possibile, infine, decidere se mostrare o meno il numero di sequenza del pacchetto ricevuto. Non vi è garanzia che tale numero sia univoco per le diverse macchine e non vi è controllo di *overflow* su quest’ultimo. E’ comunque un utile strumento di *debug* per brevi campioni di pacchetti ricevuti.

L’opzione Correct on the fly effettua un controllo sui valori immessi dall’utente durante l’immissione stessa. Anche se non si seleziona questa casella il controllo viene effettuato una volta premuto il pulsante OK.

1. Testing effettuato su una macchina equipaggiata con un hardware adeguato. [↑](#footnote-ref-2)