Politecnico di Torino

POLITECNICO DI TORINO

Programmazione di Sistema

Report Progetto 2.1: Backup di emergenza

Marika Nappi, Fabiano Vaglio, Francesco Passiatore, Francesca Saglimbeni

Sommario

Questo documento descrive lo sviluppo di un'applicazione scritta in linguaggio Rust, pensata per eseguire il backup dei dati contenuti in una cartella scelta dall'utente tramite comandi accessibili con il mouse. L'applicazione è progettata per funzionare in background, riducendo al minimo l'utilizzo delle risorse della CPU e offrendo diverse opzioni di backup. Nel report vengono illustrate le fasi dello sviluppo, le funzionalità implementate e i risultati raggiunti.

Indice

1 Descrizione del progetto

Abbiamo sviluppato un programma che, all'avvio del PC, avvia automaticamente un'operazione di backup. L'unica interazione richiesta è la selezione iniziale delle directory, dopodiché il processo si svolge in modo completamente autonomo, senza necessità di monitor, sfruttando esclusivamente i movimenti del mouse. Il processo di utilizzo si articola nelle seguenti fasi:

1. **Selezione delle directory e del tipo di backup:** Al momento dell'avvio del PC, verrà visualizzata una finestra che chiederà di specificare il tipo di backup da eseguire: un backup dell'intera cartella o di alcuni file con estensioni specifiche (.txt, .TOML). Inoltre, sarà necessario indicare la directory di origine e la directory di destinazione per il salvataggio. Una volta completata questa configurazione, inizierà il monitoraggio dei movimenti del mouse.

2. **Monitoraggio del mouse:** Quando il monitoraggio del mouse è attivo, il sistema sarà in grado di riconoscere la forma di un rettangolo tracciato lungo i bordi dello schermo. Al termine della forma, verrà emesso un segnale sonoro per confermare il riconoscimento.

3. **Seconda forma e conferma:** Successivamente, sarà necessario disegnare una seconda forma, un segno meno al centro dello schermo. Anche in questo caso, al termine della forma, verrà emesso un segnale sonoro di conferma. L'introduzione di due forme serve a garantire che il backup non venga avviato accidentalmente.

4. **Avvio del backup:** Dopo aver completato i due segni, il backup partirà automaticamente, concludendosi con un segnale sonoro che indicherà l'esito del processo insieme ad un pop-up.

Poiché il programma è progettato per operare senza monitor, ad eccezione dell'interfaccia iniziale, tutti i comandi saranno accompagnati da un segnale sonoro che ne conferma il riconoscimento o il completamento.

2 Sviluppo Codice

Lo sviluppo del codice è stato suddiviso in due gruppi di lavoro:

1. Gestione del *backup*;
2. *Tracciamento* del mouse.

Dopo aver definito queste due parti principali, ci si è concentrati sulla gestione dei suoni e sulla creazione di una GUI per selezionare le cartelle di input e output del backup (ossia la cartella di origine e quella di destinazione). Successivamente, è stata implementata una funzionalità per avviare il sistema automaticamente all'accensione e per eseguirlo in background. Infine, è stato effettuato l'adattamento del sistema ai vari sistemi operativi (macOS, Linux e Windows).

2.1 Implementazione della GUI

È stata creata un'interfaccia grafica la cui logica è stata sviluppata con la libreria `eframe`. L'utente può configurare se eseguire un backup dell'intera cartella o solo di determinati tipi di file, selezionando anche i percorsi di input e output. Queste preferenze vengono salvate in un file `config.toml`. La struct `ConfigApp` definisce i parametri che verranno utilizzati per il backup, come la scelta tra un backup dell'intera cartella o di file specifici con una determinata estensione, insieme ai percorsi delle cartelle sorgente e di destinazione.

Quando l'utente clicca su "Salva configurazione", la GUI si chiude e i parametri vengono scritti nel file `config.toml`. Questi parametri verranno poi letti dalla logica del backup (descritta nella sezione 2.2) per eseguire il backup effettivo. La funzione `run_gui()` è quella che avvia la GUI come finestra separata, consentendo una gestione indipendente dell'interfaccia utente.

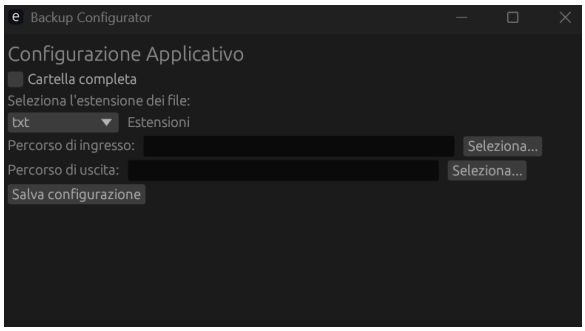

Figure 1: Interfaccia grafica - Backup file .txt

Figure 2: Interfaccia grafica - Backup intera cartella

Un aspetto importante da sottolineare è che la logica della GUI è stata implementata come processo figlio del processo principale, per separare le operazioni di configurazione dell'interfaccia dal flusso principale dell'applicazione, migliorando così l'organizzazione del codice e la gestione parallela delle operazioni, soprattutto nei sistemi operativi come Linux e macOS.

2.2 Tracciamento

Il sistema di tracciamento è stato progettato per garantire un rilevamento preciso del movimento del mouse, utilizzando librerie Rust specializzate nella gestione delle interazioni hardware, come `rdev`. L'implementazione si basa su un'architettura multithread che suddivide i compiti di monitoraggio in due thread separati, ottimizzando così il processo di rilevamento.

Per garantire l'efficacia e la sincronizzazione delle operazioni, il lavoro di tracciamento è stato suddiviso in due thread principali:

1. **Thread di monitoraggio del rettangolo**: il primo thread è responsabile del rilevamento del rettangolo tracciato dal mouse, nonché della gestione del passaggio tra il tracciamento del rettangolo e quello del segno meno.

2. **Thread di monitoraggio del segno meno**: il secondo thread si occupa esclusivamente del monitoraggio del movimento orizzontale del mouse, necessario per identificare un segno meno.

Per garantire la corretta sincronizzazione tra i due thread, è stato utilizzato un oggetto `Arc<AtomicBool>`, che consente di mantenere un flag condiviso tra i thread. Inoltre, la comunicazione tra i thread è avvenuta tramite un *channel*: il thread principale (`monitor_movement`) invia segnali tramite il `sender`, mentre il secondo thread (`monitor_minus_sign`) ascolta questi segnali attraverso il `receiver`.

2.2.1 Rilevamento rettangolo

Una volta gestita la parte multithread, il tracciamento del rettangolo è stato concepito con specifiche condizioni di validità per garantire l'affidabilità del sistema. Il movimento del mouse è monitorato in modo che possa iniziare da qualsiasi angolo del desktop, offrendo flessibilità all'utente. Tuttavia, per essere considerato valido, il tracciamento deve rispettare le seguenti condizioni:

1. **Chiusura completa del rettangolo**: non basta toccare tutti e quattro gli angoli dello schermo; il movimento deve essere continuo lungo i bordi, garantendo che il rettangolo sia chiuso correttamente.

2. **Precisione del percorso**: qualsiasi deviazione significativa al di fuori dei bordi, o un movimento errato, interrompe il processo di tracciamento. Il sistema, difatti, è dotato di un meccanismo di reset automatico che azzera il tracker se le condizioni non vengono soddisfatte, attendendo un nuovo tentativo fino a quando non viene completato un rettangolo valido.

Una volta che il rettangolo è stato tracciato correttamente, il programma emette una segnalazione sonora per confermare il successo del tracciamento e avvia automaticamente il monitoraggio del segno meno.

2.2.2 Rilevamento del segno meno

La funzione `detect_minus_sign` è progettata per rilevare i movimenti orizzontali del mouse, utili per identificare un segno meno. Questo tracciamento può avvenire in qualsiasi area dello schermo, purché il movimento sia orizzontale e percorra almeno il 20% della larghezza dello schermo. Inoltre, viene applicata una tolleranza per il movimento verticale al fine di garantire l'accuratezza del rilevamento. Al termine del tracciamento del segno meno, viene emessa una segnalazione sonora che indica che il backup è stato completato con successo e appare una'altra finestra che indica che il backup è stato completato.

2.2.3 Finestra Di conferma

La finestra di conferma viene mostrata immediatamente dopo il completamento del rettangolo, notificando all'utente che il monitoraggio è stato effettuato con successo. Contestualmente, viene avviato un timeout di 10 secondi, durante i quali il sistema attende il rilevamento del segno meno come conferma per procedere con il completamento del backup. Se il segno meno non viene rilevato entro il tempo previsto, il processo viene resettato e il sistema riprende il monitoraggio del rettangolo.

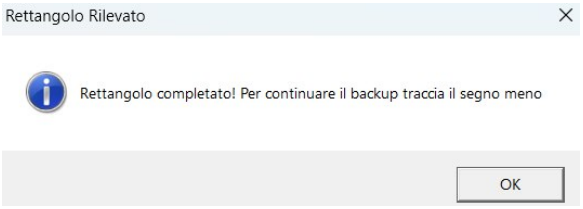

Figure 3: Finestra di conferma dopo il rettangolo

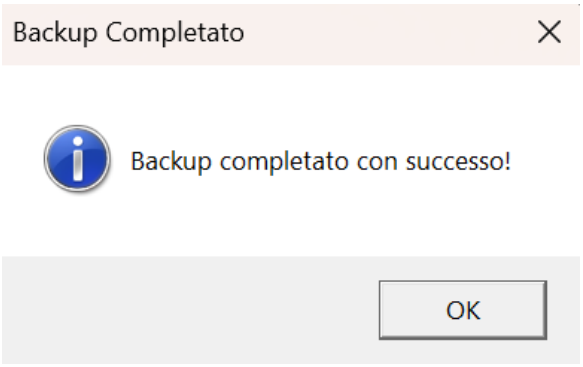

Figure 4: Finestra di conferma finale a backup effettuato

2.3 Logica del Backup

Nel file `backup.rs` è implementata la logica principale che gestisce l'intero processo di backup, definendo in modo dettagliato le operazioni necessarie per garantire il corretto funzionamento dell'applicativo.

2.3.1 Funzionalità Principali

Come detto nella sezione 2.1 sulla GUI l'utente può scegliere tra due diverse implementazioni del backup:

- **Backup Completo della Cartella**: copia l'intero contenuto di una directory sorgente, incluse eventuali sottocartelle, preservandone la struttura.
- **Backup per Tipo di File**: permette di selezionare solo i file con una determinata estensione (ad esempio, `.txt` o `.jpg`), offrendo maggiore flessibilità quando non è necessario copiare tutti i dati.

Una volta configurata l'operazione, il programma verifica che i percorsi di sorgente e destinazione siano validi, e inizia il processo di copia. Per garantire velocità ed efficienza, la copia dei file viene gestita in parallelo utilizzando la libreria `Rayon`, particolarmente utile in caso di grandi quantità di dati.

Al termine del backup, il programma registra alcune informazioni chiave in un file di log chiamato `backup_log.txt`, dove vengono riportati:

- La dimensione totale dei file copiati (in byte).
- Il tempo impiegato per completare l'operazione.

Questo consente di mantenere una traccia delle attività svolte, utile sia per analisi che per eventuali verifiche. Importante è ricordare che questo file creato (`backup_log.txt`) sarà salvato nella directory di destinazione scelta dall'utente. Su questo file sono annotati la grandezza del file di cui si è effettuato e il tempo in termini di CPU impiegato per il backup.

2.3.2 Notifiche e Gestione degli Errori

L'applicativo è progettato per essere user-friendly, grazie anche al sistema di notifiche implementato. In caso di successo, l'utente riceve un messaggio visivo:

- Su Windows e macOS, viene mostrata una finestra di dialogo con un messaggio di conferma.
- Su Linux, il programma utilizza `Zenity` per notificare il completamento.

Oltre al messaggio visivo, vengono riprodotti suoni di notifica per indicare se il backup è andato a buon fine o se si è verificato un errore. In caso di problemi, come percorsi non validi o inesistenti, il programma interrompe l'operazione e informa l'utente, evitando di proseguire con dati incompleti o errati.

2.3.3 Architettura Tecnica

L'applicativo è strutturato in modo modulare, con funzioni specifiche per ciascuna operazione:

- La funzione principale, `perform_backup`, si occupa di orchestrare tutto il processo, dalla scelta del tipo di backup alla gestione dei percorsi, fino alla registrazione del log finale.
- La copia dei file viene gestita dalla funzione `copy_directory_parallel`, che ottimizza il processo sfruttando la parallelizzazione.
- Altre funzioni, come `calculate_total_size` e `log_backup_info`, calcolano la dimensione totale dei dati copiati e registrano i dettagli in un file di log.

L'intero processo è pensato per offrire un'esperienza semplice e fluida. L'utente può selezionare rapidamente i parametri di backup desiderati, ricevere notifiche chiare al termine dell'operazione e accedere a un log dettagliato delle attività.

In definitiva, questo applicativo si distingue per l'efficienza e l'affidabilità, rendendolo uno strumento ideale sia per utenti che necessitano di una soluzione rapida e senza complicazioni, sia per chi ha esigenze più specifiche di gestione dei dati.

2.4 Gestione e Riproduzione dei Suoni

Il codice presentato gestisce la riproduzione di suoni all'interno di un'applicazione Rust, utilizzando due librerie principali: `rodio` e `cpal`. Le funzioni incluse sono progettate per riprodurre suoni in risposta a eventi specifici, come il completamento di un backup (suono di successo) o la sua interruzione (suono di errore), ma anche per generare toni audio in tempo reale.

2.4.1 Funzione `play_sound_backup_ok`

La funzione `play_sound_backup_ok` è responsabile della riproduzione di un suono che segnala il completamento con successo di un backup. Inizia recuperando la directory del progetto tramite la funzione `get_project_directory`, che restituisce il percorso del progetto. Successivamente, viene costruito il percorso completo del file audio `successoBackup.wav`, il quale si trova nella directory del progetto.

Il flusso audio viene gestito utilizzando la libreria `rodio`, che fornisce un'interfaccia semplice per lavorare con file audio. In particolare, viene creato un flusso di output predefinito con `OutputStream::try_default()`. Una volta che il flusso è configurato, viene creato un oggetto `sink`, che consente di inviare i dati audio al flusso di output. Il file WAV viene aperto e decodificato tramite `Decoder::new_wav`, e successivamente il suono viene riprodotto aggiungendo la fonte audio al sink. La funzione `sink.sleep_until_end()` è utilizzata per bloccare l'esecuzione del programma fino a quando la riproduzione del suono non è terminata, garantendo che la funzione non ritorni finché il suono non è stato completamente riprodotto.

2.4.2 Funzione `play_sound_backup_error`

La funzione `play_sound_backup_error` segue un processo simile alla funzione precedente, ma con una differenza importante: riproduce un suono di errore in caso di fallimento durante il backup. In questo caso, il file audio `erroreBackup.wav` viene riprodotto invece del suono di successo. La logica che gestisce la lettura del file e la riproduzione del suono è identica a quella della funzione precedente, con l'unica differenza rappresentata dal file audio caricato.

2.4.3 Funzione `play_sound_sign`

La funzione `play_sound_sign` ha un obiettivo diverso rispetto alle due precedenti: genera e riproduce un tono audio in tempo reale, utilizzando la libreria `cpal`. Quest'ultima consente di lavorare con dispositivi audio a basso livello, offrendo un controllo diretto sulla generazione del suono.

In questa funzione, un tono sinusoidale viene generato con una frequenza di 550 Hz e una durata di un secondo. La configurazione del dispositivo di output viene eseguita tramite `cpal`, che recupera il dispositivo audio predefinito e la sua configurazione di uscita. Il flusso audio viene creato utilizzando il metodo `build_output_stream`, e la generazione del suono avviene all'interno di una funzione di callback. Ogni campione di suono viene calcolato in tempo reale in base alla frequenza e inviato al dispositivo di output. Il contatore `sample_clock` traccia la posizione nella forma d'onda, aggiornando ogni campione e producendo il tono richiesto.
Infine, la funzione `std::thread::sleep` viene utilizzata per mantenere il suono per un secondo, dopo il quale la funzione termina.

2.5 Registrazione del Consumo di CPU ogni 2 Minuti

La funzione `log_cpu_usage` è progettata per monitorare l'utilizzo della CPU di un processo specifico e registrare i dati raccolti in un file di log. Questo processo avviene su un periodo di tempo definito e include diversi passaggi che garantiscono una raccolta accurata e una registrazione persistente dei dati.

La funzione inizia con l'inizializzazione di un'istanza della struttura `System` dalla libreria `sysinfo`. Questa struttura fornisce informazioni sul sistema, tra cui lo stato dei processi in esecuzione. Successivamente, viene recuperato il PID (Process ID) del processo corrente tramite la funzione `std::process::id()`, che restituisce un identificatore univoco per il processo in esecuzione. Questo PID viene stampato sulla console, fornendo un'indicazione del processo che verrà monitorato.

La funzione prevede due intervalli temporali principali: l'intervallo di logging, che è di 120 secondi (2 minuti), e l'intervallo di campionamento, fissato a 5 secondi. L'intervallo di logging definisce la durata totale durante la quale verranno raccolti i dati sull'utilizzo della CPU, mentre l'intervallo di campionamento indica quanto frequentemente i dati vengono raccolti durante questo periodo.

2.5.1 Raccolta dei Dati e Calcolo della Media dell'Uso della CPU

Il cuore del processo è un ciclo che dura per tutta la durata dell'intervallo di logging. All'interno di questo ciclo, viene avviato un secondo ciclo che raccoglie i dati sull'utilizzo della CPU ogni 5 secondi. Per fare ciò, viene chiamato il metodo `system.refresh_processes`, che aggiorna le informazioni sui processi in esecuzione, limitandosi al processo identificato dal PID specificato. Viene quindi controllato se il processo esiste ancora e, se presente, viene recuperato il valore di utilizzo della CPU tramite il metodo `cpu_usage()`.

Questi dati vengono sommati ad una variabile che tiene traccia del totale dell'utilizzo della CPU durante l'intervallo di logging. Ogni 5 secondi, il ciclo ripete la raccolta, aumentando così il campione di dati. Dopo il termine dell'intervallo di logging (120 secondi), la funzione calcola la media dell'utilizzo della CPU, normalizzandola in base al numero di core della CPU, poiché il valore di utilizzo della CPU è fornito separatamente per ciascun core.

Una volta calcolata la media, i risultati vengono scritti nel file di log. Se il campionamento ha prodotto dati validi, vengono registrati il PID del processo, l'utilizzo medio della CPU e il numero di campioni raccolti. In caso contrario, se non sono stati raccolti dati sufficienti, viene scritto nel file un messaggio che indica l'assenza di dati validi per il calcolo della media.

Anche in caso di errore durante la scrittura dei dati nel file di log, viene visualizzato un messaggio di errore sulla console, e la funzione tenta di continuare la sua esecuzione. Ogni volta che vengono scritti dei dati nel file, viene effettuato un "flush" del file per forzare la scrittura immediata dei dati sul disco, garantendo che le informazioni non vengano perse.

La funzione `log_cpu_usage` è un esempio di come monitorare l'utilizzo della CPU di un processo e registrare i dati in un file di log per l'analisi successiva. Attraverso l'uso di una combinazione di intervalli di tempo per il campionamento e il logging, il programma è in grado di raccogliere e registrare informazioni utili sull'uso delle risorse di sistema.

2.6 Configurazione dell'Avvio Automatico

Il codice presentato ha come obiettivo principale configurare l'avvio automatico dell'applicazione di backup al momento dell'accensione del sistema, permettendo all'applicazione di avviarsi senza che l'utente debba fare nulla manualmente. La funzione `configure_autorun` è responsabile di questo compito e agisce in modo differenziato a seconda del sistema operativo in uso (Windows, Linux o macOS).

2.6.1 Avvio automatico su Windows

Quando l'applicazione viene eseguita su Windows, la funzione sfrutta la libreria `winreg` per accedere al registro di sistema e modificare le impostazioni relative ai programmi che si avviano automaticamente. Il registro di sistema di Windows contiene una sezione specifica, `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`, dove sono memorizzati i percorsi degli eseguibili da lanciare all'avvio del sistema.
La funzione apre questa sezione del registro e cerca una voce chiamata "Backup", la quale dovrebbe contenere il percorso dell'eseguibile dell'applicazione. Se tale voce è già presente e contiene il percorso giusto (cioè quello dell'eseguibile corrente), significa che l'applicazione è già configurata per l'avvio automatico e quindi la funzione stampa un messaggio che lo indica.
Se invece non esiste una voce per l'avvio automatico, la funzione ne crea una con il nome "Backup" e imposta come valore il percorso completo dell'eseguibile dell'applicazione (avvolto tra virgolette per gestire eventuali spazi nel percorso). Questo permette all'applicazione di essere avviata automaticamente ogni volta che l'utente accende il computer, senza ulteriori interventi manuali.

2.7 Avvio automatico su Linux

Nel caso di Linux, la funzione adotta un approccio diverso. Linux non ha un registro di sistema come Windows, ma utilizza una struttura di file di configurazione per gestire l'avvio automatico delle applicazioni. In particolare, la directory `/.config/autostart` è il posto in cui vengono inseriti i file di configurazione per le applicazioni che devono essere avviate automaticamente all'avvio del sistema.
La funzione crea questa directory, se non esiste già, e al suo interno genera un file con estensione `.desktop` che definisce l'applicazione da avviare. All'interno di questo file vengono inseriti i dettagli dell'applicazione, come il tipo di programma, il nome e, soprattutto, il comando per avviare l'eseguibile dell'applicazione. Il file è strutturato in un formato che il sistema Linux può interpretare, consentendo all'applicazione di essere eseguita al boot.

2.8 Avvio automatico su macOS

Anche macOS ha un meccanismo proprio per gestire l'avvio automatico delle applicazioni. In questo caso, la funzione crea un file di configurazione in formato `.plist` nella directory `/Library/LaunchAgents`. Questo file è utilizzato dal sistema operativo per definire agenti di lancio che vengono eseguiti automaticamente quando il sistema si avvia.
Il file `.plist` creato dalla funzione contiene informazioni come il nome dell'applicazione (nel nostro caso "com.example.backup"), il percorso dell'eseguibile da avviare e una direttiva che indica che l'applicazione deve essere eseguita automaticamente all'avvio (`RunAtLoad`). Questo formato XML è specifico di macOS e consente di configurare il sistema per eseguire l'applicazione ogni volta che l'utente accende il computer.

3 Conclusioni

In conclusione, l'applicativo sviluppato in Rust offre una soluzione di backup versatile e automatizzata, con supporto per Windows, Linux e macOS. La configurazione dell'avvio automatico assicura che il programma si avvii ad ogni accensione del sistema, senza intervento dell'utente. L'applicazione utilizza il multi-threading per ottimizzare le risorse di sistema e gestisce in modo sicuro i percorsi di input e output. Grazie all'interfaccia grafica separata e alla possibilità di attivare il backup tramite comandi del mouse, il programma è facilmente utilizzabile, garantendo un'esperienza utente semplice ed efficiente.