Backup Application- Group 39

Manuale progettista

Raffaele Pane - S305485
Veronica Mattei - S310707
Jacopo Spaccatrosi - S285891

Indice

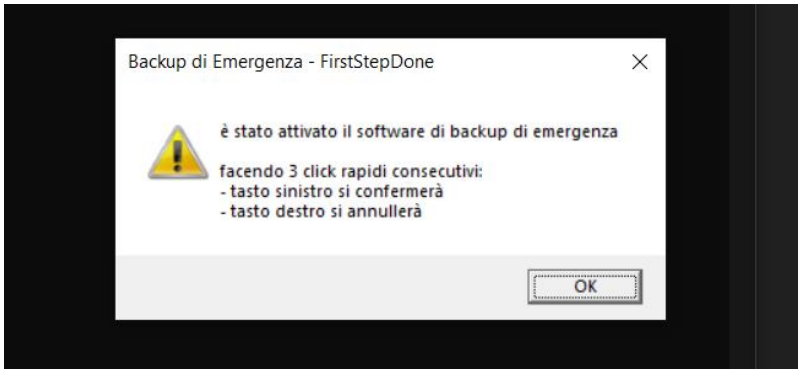
Panoramica

Questo progetto è un'applicazione Rust progettata per facilitare i backup quando lo schermo non è accessibile. L'utente può avviare un backup su un'unità esterna (come, ad esempio, una chiavetta USB) utilizzando un pattern convenzionale, che avrà una prima fase di attivazione e poi una seconda di conferma/cancellazione. Una volta confermato, verrà avviato il backup vero e proprio, dal path sorgente a quello di destinazione specificati nel file di configurazione.

Nell'applicativo sono presenti due possibili pattern per l'attivazione e la successiva selezione: un primo basato sul solo movimento del mouse ed un secondo che utilizza una combinazione di tasti e, successivamente, click del mouse.

All'inizio il programma resterà in attesa, controllando se viene effettuato il comando di attivazione.
Una volta usato tale comando, sarà possibile usare un secondo comando per confermare o annullare.
Nel caso si annulli, il programma riparte dalla fase pre-attivazione, mettendosi in attesa del comando di attivazione.
Nel caso si confermi, invece, il programma inizierà il backup vero e proprio. Una volta conclusosi, emetterà un segnale acustico accompagnato da un alert. Dopo 30 secondi, il programma termina.

Ogni step viene accompagnato da ulteriori feedback audiovisivi: un alert esplicativo ed un segnale acustico. In questo modo, anche in caso di fallimenti dei componenti hardware si sarà in grado di capire se si sta procedendo nelle varie fasi.



L'applicazione è stata pensata dal primo momento per essere multipiattaforma ed utilizza librerie compatibili con tutti i principali sistemi operativi: Windows, Linux, MacOS.
Nonostante questo, non avendo a disposizione dispositivi Apple, è stato possibile testare e rendere completamente funzionanti solo le versioni di Windows e Linux.

Struttura del Progetto

Il progetto è organizzato nei seguenti moduli principali:

- **src/**
    - **backup.rs**: Gestisce le operazioni di backup
    - **config.rs**: Gestisce il recupero e l'utilizzo del file di configurazione
    - **notification_popup.rs**: Gestisce la logica dei popup
    - **buttons_and_clicks_pattern_recognizer.rs**: Riconosce e gestisce il pattern formato dalla combinazione tasti più click
    - **beeper.rs**: Emette segnali acustici
    - **logger.rs**: Permette il logging sull'utilizzo della CPU e sui dettagli del backup
    - **main.rs**: Flusso principale del progetto
    - **pattern_recognizer.rs**: Riconosce e gestisce i pattern basati sul solo movimento del mouse

- **resources/**
    - **positive-beep.wav** e **negative-beep.wav**: Suoni di beep
- **log/**

    Cartella dove verranno salvati i log sul consumo della CPU

- **config.yaml**: File di configurazione del progetto
- **cargo.toml**: File di configurazione del progetto Rust, include le dipendenze

Dipendenze

Le principali dipendenze del progetto sono elencate nel file *Cargo.toml*:

- **sysinfo**: Per ottenere informazioni sul sistema
- **chrono**: Per la gestione delle date e degli orari
- **rodio**: Per la gestione dell'audio
- **serde** e **serde_yaml**: Per la serializzazione e deserializzazione dei dati di configurazione
- **tokio**: Per la gestione asincrona delle operazioni di I/O
- **libc**: Per le chiamate di sistema
- **native-dialog**: Per la gestione delle finestre di dialogo native
- **rdev**: Per la gestione degli eventi del mouse e della tastiera
- **emath**: Per le operazioni matematiche
- **winapi**: Per le chiamate di sistema su Windows
- **notify**: Per la gestione delle notifiche del file system
- **notify-rust**: Per creare notifiche di sistema su Linux/MacOs
- **async-recursion**: Per eseguire funzioni ricorsive in modo asincrono
- **x11**: Fornisce bindings per interfacciarsi con librerie X11 utilizzate su Unix

Dettagli dei File

backup.rs

Il file **backup.rs** è la componente del progetto del sistema di backup, responsabile della gestione e dell'esecuzione delle operazioni di backup dei file.

L'attività di backup inizia con la funzione *wrapper_backup* che organizza il processo di backup calcolando prima il numero totale di file e la dimensione complessiva dei dati da copiare attraverso la funzione *calculate_total_files*. Questa calcola ricorsivamente il numero di file e la loro dimensione totale, considerando solo i tipi di file specificati nel file di configurazione *config.yaml*, se non specificati tutti i tipi di file vengono copiati.

Una volta calcolati i totali dei file da copiare ed ottenuta la size totale, *wrapper_backup* avvia la funzione *backup*, che esegue effettivamente la copia dei file. La prima attività di questa funzione è di creare una lista di compiti (`tasks`), ognuno rappresentante un file da copiare dalla sorgente alla destinazione, utilizzando la funzione *schedule_backup_tasks*. Tramite questo metodo ogni file e directory viene esaminato ricorsivamente, e i task vengono pianificati. Una volta compilata la lista dei compiti, la funzione *backup* prosegue con l'esecuzione asincrona di queste operazioni di backup.
Le operazioni di backup sono gestite attraverso thread asincroni, ciascuno corrispondente a un compito di backup. Ogni task di backup viene eseguito in un thread separato. Questo approccio permette di sfruttare le I/O asincrone per una copia efficiente e non bloccante, garantendo che il sistema non venga sovraccaricato, poiché il numero di thread attivi contemporaneamente è controllato dal semaforo (definito dal massimo numero di file apribili contemporaneamente dal sistema operativo). Inoltre, la logica fornisce un output del progresso del backup, aggiornando tramite i log lo stato del trasferimento percentuale del backup.

In caso di errori, come percorsi sorgente o destinazione inesistenti, la logica notifica l'utente tramite popup di errore.

config.rs

Il file **config.rs** è incaricato di gestire e recuperare tutte le configurazioni necessarie per il funzionamento del programma. Esso definisce una struttura di configurazione, *Config*, che memorizza vari parametri essenziali come i percorsi delle directory di origine e destinazione del backup, i tipi di file da includere e il modo con cui viene attivata la procedura di backup.

La funzione *read_from_file* tenta di aprire e leggere il file *config.yaml*, utilizzando la libreria *serde_yaml* per deserializzare i dati in un oggetto Config. In caso di errori, come l'impossibilità di aprire o leggere il file, la funzione restituisce un errore, facilitando la gestione degli stessi a livelli superiori del sistema. Il metodo *retrieve_and_check_config_file* monitora attivamente il file di configurazione per le modifiche, utilizzando la libreria notify. Questo permette alla logica di backup di adattarsi dinamicamente senza necessità di riavvi, migliorando l'efficienza e riducendo i tempi di inattività.

Il file **config.rs** include una gestione degli errori attraverso la funzione *handle_config_error*, che quando si verifica un errore, la funzione visualizza notifiche popup specifiche, basate sul tipo di errore rilevato.

notification_popup.rs

Questo file è responsabile della gestione delle notifiche popup che informano l'utente sullo stato delle varie fasi del programma. Al suo interno contiene funzioni e strutture che permettono di mostrare messaggi di notifica all'utente in base a diverse tipologie di eventi, come l'inizio o la cancellazione di un backup.

L'implementazione supporta sia sistemi Windows che Linux, e utilizza differenti librerie e approcci a seconda del sistema operativo.

L'enumerazione *NotificationType* rappresenta i diversi tipi di notifiche che sono presenti all'interno del programma, che verranno poi utilizzate per personalizzare il popup.

Su sistemi Windows, la funzione *show_popup* utilizza la libreria native_dialog per mostrare una notifica popup basata sul tipo di notifica e un messaggio opzionale. La funzione *close_related_popups* chiude eventuali popup esistenti prima di mostrarne uno nuovo, per evitare sovrapposizioni. La funzione *show_notification_popup* mostra una notifica popup in un thread separato, utilizzando la libreria native_dialog per creare e visualizzare il popup. A seconda del tipo di notifica, viene mostrato un messaggio specifico.
A partire dalla funzione *close_related_popups*, viene richiamata la funzione *close_popup* tenta di chiudere un popup esistente con un titolo specificato simulando un click sul pulsante "OK". Questa funzione utilizza diverse funzioni di Windows API per enumerare le finestre e i loro componenti figli. Una volta effettuata la prima enumerazione e trovata la finestra target, si effettua una seconda enumerazione sui componenti presenti al suo interno fino a trovare il pulsante "OK". A quel punto utilizzerà l'handler trovato per simulare il click.
NOTA: tutta questa parte destinata alla chiusura dei popup è stata implementata solo per il sistema operativo Windows ed utilizza fortemente il crate di winapi (e quindi le Windows API). Ciò ha comportato anche l'utilizzo estensivo di blocchi unsafe. In aiuto alla lettura di queste parti, ognuno di questi metodi è stato, nel codice, documentato e commentato in maniera estremamene verbosa.

Su Linux, la gestione delle notifiche popup è implementata solo tramite il metodo *show_popup* che usa la libreria notify-rust, per visualizzare notifiche di sistema attraverso i server di notifica nativi del sistema. Le notifiche di sistema sono non bloccanti e vengono visualizzate sullo schermo e chiuse automaticamente dopo un certo intervallo di tempo.

In sintesi, **notification_popup.rs** gestisce la visualizzazione e la chiusura delle notifiche popup, fornendo un feedback visivo all'utente sulle operazioni dell'applicativo.

buttons_and_clicks_pattern_recognizer.rs

Il file **button_and_clicks_pattern_recognizer.rs** è una parte essenziale del progetto ed è l'implementazione di uno dei due pattern disponibili nell'applicativo per l'attivazione e la successiva conferma/annullo del backup. In questo caso il pattern si basa su un'attivazione data da una combinazione di pulsanti e una successiva validazione basata su click.

Al suo interno è definita una sola funzione *start_button_and_clicks_pattern_recognizer* che gestisce l'intero ciclo di vita del pattern.

In primis avvia un thread dedicato all'ascolto degli eventi di tastiera e mouse. Questa funzione utilizza una macchina a stati per gestire le diverse fasi del riconoscimento dei pattern. Gli stati includono *Waiting*, *CtrlAltBPressed*, *Activated* e *Sleeping*, e vanno a definire le possibili azioni e controlli da effettuare allo scatenarsi di specifici eventi.

All'inizio lo stato è in Waiting e si mette in attesa di rilevare la combinazione di tasti *ctrl + alt + b*. Una volta che vengo premuti, lo stato passa in *CtrlAltBPressed* e parte un timer atto a controllare che vengano premuti per cinque secondi. Una volta che questo avviene lo stato cambia in *Activated*, permettendo all'utente di confermare o annullare l'operazione attraverso tre click consecutivi del mouse (sinistro per confermare, destro per annullare). Se si annulla (3 click consecutivi del tasto destro del mouse) l'attivazione viene annullata e la macchina a stati ritorna nello stato *Waiting*. Viceversa, se si conferma (3 click consecutivi del tasto sinistro del mouse) l'attivazione viene confermata, viene aggiornata il booleano su cui viene effettuato il controllo della condition variable, inviata la notifica su di essa e cambiato lo stato in *Sleeping*. A questo punto il thread secondario si metterà in attesa, senza consumare cicli di CPU, mentre il thread principale proseguirà l'esecuzione ritornando nel main.

beeper.rs

Questo file è responsabile della gestione e riproduzione dei suoni di notifica, usati per avere un feedback acustico utile soprattutto in caso di problemi ai dispositivi grafici, così da potersi orientare al meglio in queste situazioni (ad esempio, confermando l'avvenuto completamento o l'annullamento di un'operazione). Al suo interno, la logica presente utilizza la libreria rodio per funzionare.

Sono presenti due funzioni principali: *emit_beep* e *beep*. La prima accetta un booleano *is_positive* utile a selezionare il file audio tra due disponibili ed emette un suono di notifica chiamando la funzione *beep* in un thread dedicato. Questa funzione ritorna un *JoinHandle* al thread creato, così il chiamante può eventualmente effettuare il join.
La funzione *beep* riproduce effettivamente il suono utilizzando la libreria rodio. Essa crea uno stream di output, carica il file audio corrispondente e lo riproduce. La funzione *beep* ritorna un *Result* che indica se il suono è stato riprodotto con successo o se si è verificato un errore.

logger.rs

Il file **logger.rs** è una parte essenziale del progetto ed è responsabile della gestione e registrazione dei dettagli relativi alle operazioni di backup. Questo file contiene funzioni e strutture che permettono di tracciare e memorizzare informazioni cruciali come la dimensione totale dei file, il numero totale di file e il tempo di CPU impiegato durante il processo di backup. La registrazione di questi dettagli è necessaria per monitorare l'efficienza e l'efficacia del sistema di backup, nonché per diagnosticare eventuali problemi che potrebbero sorgere.

Al suo interno è definita una struttura *Logger* utilizzata per registrare l'uso della CPU e i dettagli del backup in un file di log. Tale struttura contiene un campo *log_file_path* che specifica il percorso del file di log. Il metodo new crea un'istanza di *Logger*, generando un nome di file basato sul tipo di log, sulla data e l'ora corrente. Il metodo *log_cpu_usage* registra l'uso della CPU in un ciclo continuo, aggiornando le informazioni ogni 2 minuti. Il metodo *log_backup_details* registra, invece, i dettagli di un backup completato, inclusi la dimensione totale, il numero di file e il tempo di CPU impiegato. La funzione *bytes_to_human_readable* è una utility che converte i byte in una stringa leggibile dall'uomo. Infine, il metodo *write_log* scrive una voce di log nel file di log specificato, effettuando quindi la scrittura vera e propria.

main.rs

Il file **main.rs** è il punto di ingresso principale del progetto e coordina l'esecuzione delle varie componenti del sistema. Questo file contiene la funzione *main*, che è il punto di partenza dell'applicazione. Al suo interno vengono inizializzate e configurate le diverse parti del sistema, come il riconoscimento dei pattern del mouse, la gestione delle notifiche e l'avvio del processo di backup.

La funzione *main* inizia configurando l'ambiente e caricando le configurazioni necessarie dal file *config.yaml*. Viene anche configurato un logger per registrare l'uso della CPU ogni due minuti.
Successivamente, basandosi sul parametro *btn_rec*, viene selezionato ed eseguito il tipo di pattern di selezione/conferma scelto.

In questo caso entrambe le funzioni si metteranno in "ascolto" per rilevare il pattern designato. Da notare che, se non vengono effettuate le gesture necessarie, il programma resterà in ascolto per un tempo indefinito.
Una volta che il pattern viene riconosciuto e confermato, il flusso ritorna al main per le successive fasi.

Una volta ritornato al *main*, significa che è stata attivata e confermata la procedura di backup. Viene, quindi, configurato un ulteriore logger per registrare i dettagli del backup.
Il processo di backup è gestito dalla funzione *wrapper_backup*, che calcola il numero totale di file e la dimensione totale dei dati da copiare. Se ci sono file da copiare, viene avviato il backup utilizzando la funzione backup, che copia i file in modo asincrono, limitando il numero massimo di file aperti contemporaneamente per evitare sovraccarichi del sistema.

Una volta completato il backup, viene scritto il corrispettivo file di log. Tale avvenimento viene poi notificato da un popup di conferma e da un segnale acustico.
Dopo 10 secondi, il programma termina.

pattern_recognizer.rs

Il file **pattern_recognizer.rs** è una componente fondamentale del progetto dedicata alla gestione del movimento del mouse e al riconoscimento dei pattern per l'avvio o la cancellazione di operazioni di backup.

Questo file contiene la struttura *PatternRecognizer*, utile durante il processo di identificazione di gestures specifiche compiute con il mouse, in particolare, il disegno di un rettangolo in senso orario o antiorario, per attivare o annullare il processo di backup. Essa utilizza vari campi per monitorare la posizione del mouse, il lato e la direzione del pattern in corso, la tolleranza per il riconoscimento dei punti vicino agli angoli e ai lati e la soglia di movimento per evitare di effettuare i controlli dei pattern per ogni minimo movimento del mouse, ma solo per movimenti significativi, mantenendo comunque un buon livello di accuratezza.

Il metodo *new* crea l'istanza di *PatternRecognizer*, inizializzando i quattro angoli del rettangolo atteso in base alle dimensioni dello schermo (ottenute tramite il metodo *get_screen_size*) e generando dei thread per monitorare la posizione del mouse usando il crate *rdev*.

Il sistema di riconoscimento del pattern è basato su una serie di controlli implementati nei metodi *recognize_pattern, pattern_recognition*, *check_rectangle_gesture_clockwise* e *check_rectangle_gesture_counterclockwise*, attraverso i quali si monitorano i movimenti del mouse e, se il percorso tracciato corrisponde al pattern previsto, vengono attivate delle notifiche visive e sonore che indicano il completamento o l'annullamento dell'operazione di backup.

Riconoscimenti

L’applicazione è stata progettata e sviluppata da

Raffaele Pane - S305485,
Veronica Mattei - S310707,
Jacopo Spaccatrosi - S285891,

studenti presso il Politecnico di Torino, per il corso di API Programming (RUST) dell’esame di Programmazione di Sistema durante gli studi magistrali in Ingegneria Informatica.