UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

PROGETTO DI RETI DI CALCOLATORI E LABORATORIO DI RETI DI CALCOLATORI

A.A. 2021 – 2022

GREEN PASS

Giordano Orsini Massimiliano - 0124002214

Indice

Elenco delle figure....2
Elenco delle tabelle....2
Descrizione del progetto....3
Descrizione e schemi dell'architettura....3
Relazioni di comunicazione....3
Descrizione e schemi del protocollo applicazione....4
Pacchetti applicazione....4
Protocolli applicazione....6
SEARCH operation....8
STORE operation....9
Dettagli implementativi dei client....10
Dettagli implementativi dei server....10
Modelli di I/O adoperati....10
Memorizzazione dei dati....10
Considerazioni generali e soluzioni alternative....11
Manuale utente....11
Istruzioni per la compilazione....11
Istruzioni per l'esecuzione....12
Tabella delle istruzioni per l'esecuzione....12
Tabella di descrizione dei parametri....13
Esempio di esecuzione....13

Elenco delle figure

Figura 1. Architettura client-server multilivello descritta dalla soluzione proposta.....4
Figura 2. Struttura del pacchetto applicazione di tipo Record....5
Figura 3. Struttura del pacchetto applicazione di tipo ClientPacket....5
Figura 4. Struttura del pacchetto applicazione di tipo Packet.....5
Figura 6. Protocollo applicazione utilizzato da Client, CentroVaccinale, ServerV, ServerDB.....6
Figura 7. Protocollo applicazione utilizzato da ClientS, ServerG, ServerV, ServerDB.....7
Figura 8. Protocollo applicazione utilizzato da ClientT, ServerG, ServerV, ServerDB.....8
Figura 9. Protocollo applicazione utilizzato da ServerV e ServerDB per l'operazione SEARCH.....8
Figura 10. Protocollo applicazione utilizzato da ServerV e ServerDB per l'operazione STORE.....9
Figura 11. Registrazione di un green pass....14
Figura 12. Controllo di un green pass valido....15
Figura 13. Revoca di un green pass....16
Figura 14. Controllo di un green pass non valido....17
Figura 15. Validazione di un green pass revocato....18

Elenco delle tabelle

Tabella 1. Istruzioni per la compilazione di ciascun programma.....11
Tabella 2. Istruzioni per l'esecuzione dei file eseguibili ottenuti dopo il processo di compilazione.....12
Tabella 3. Descrizione dei parametri richiesti da linea di comando per l'esecuzione....13

Descrizione del progetto

La traccia richiede di implementare un servizio per la gestione dei green pass secondo delle specifiche ben definite: in particolare tale servizio deve permettere di registrare, validare o invalidare il green pass associato ad un particolare codice fiscale. Tali elaborazioni sono richieste da diversi client ma eseguite soltanto da un server principale.

I client quindi si interfacciano con dei server intermedi che hanno il compito di inviare tali richieste al server principale e di smistare i risultati delle elaborazioni, ottenuti da questi, ad i client che ne hanno fatto richiesta.

I client possono essere di tipo *Client*, *ClientS*, *ClientT* mentre i server intermedi sono rappresentati dai *Centri Vaccinali* e dal *ServerG*. Il server principale è rappresentato dal *ServerV*.

I dati relativi ai green pass generati sono memorizzati all'interno di un database al quale è possibile accedere soltanto dal *ServerV*. Per tale motivo, anche se non richiesto in alcun modo dalla traccia, è aggiunta una nuova entità allo schema per rappresentare un server *Database* con il quale il *ServerV* interagisce nella memorizzazione delle informazioni associate ad un green pass e nel recupero di tali informazioni.

Descrizione e schemi dell'architettura

Dalla descrizione del problema, si evincono le diverse entità che compongono l'architettura su cui si basa tale software. Si tratta dunque di un'architettura *client-server multilivello* descritta come segue: al primo livello troviamo i client, al secondo livello troviamo i server intermedi, al terzo livello troviamo il server principale, al quarto livello il server database.

E' possibile quindi visualizzare tale architettura come una particolare architettura a strati chiusa, in quanto ogni entità ad un particolare livello, escluso il primo e l'ultimo, richiede un servizio allo strato strettamente superiore e fornisce un servizio allo strato strettamente inferiore.

Al primo livello di tale architettura sono posti quindi uno o più processi *client*, processi *clientS* e processi *clientT*. Al secondo livello sono posti uno o più processi *centroVaccinale* e il processo *serverG*. Al terzo livello è posto il processo *serverV* e al quarto il processo *serverDB*.

Relazioni di comunicazione

Le relazioni di comunicazione tra le entità dell'architettura proposta sono descritte quindi come seguono:

- i client comunicano le proprie richieste ad un server intermedio;
- i server intermedi comunicano sia con i client che con il server principale;
- il server principale comunica con i server intermedi e con il server database;
- il server database comunica unicamente con il server principale.

Tali relazioni vengono descritte nel dettaglio per ogni entità dell'architettura proposta:

- un processo *client* comunica unicamente con un processo *centroVaccinale*, pertanto invierà una richiesta a tale processo e riceverà una risposta;
- un processo *clientS* comunica unicamente con il processo *serverG*, pertanto invierà una richiesta a tale processo e riceverà una risposta;
- un processo *clientT* si comporta in maniera analoga ad un processo *clientS* per quanto concerne le relazioni di comunicazione;
- un processo *centroVaccinale* accetta richieste da uno o più processi client e le inoltra al processo *serverV*. Una volta ottenuta la risposta relativa alla particolare richiesta inviata, questa viene smistata al processo *client* corrispondente;
- il processo *serverG* accetta richieste da uno o più processi *clientS* oppure uno o più processi *clientT*. Inoltra quindi tali richieste al processo *serverV* ed una volta ottenuta la risposta alla particolare richiesta inviata, così come per un processo *centroVaccinale*, questa viene smistata al processo *clientT* o *clientS* corrispondente;
- il processo *serverV* accetta richieste da uno o più processi *centroVaccinale* e/o dal processo *serverG* ed invia il risultato dell'elaborazione di tali richieste al processo corrispondente. Tale processo si interfaccia con il processo *serverDB* per recuperare e/o memorizzare informazioni relative ai green pass generati;
- il processo *serverDB* accetta richieste da parte del processo *serverV* e svolge le operazioni richieste da tale processo una volta che ha stabilito una connessione.

Di seguito viene riportata l'architettura relativa alla soluzione proposta sulla base delle relazioni appena descritte.

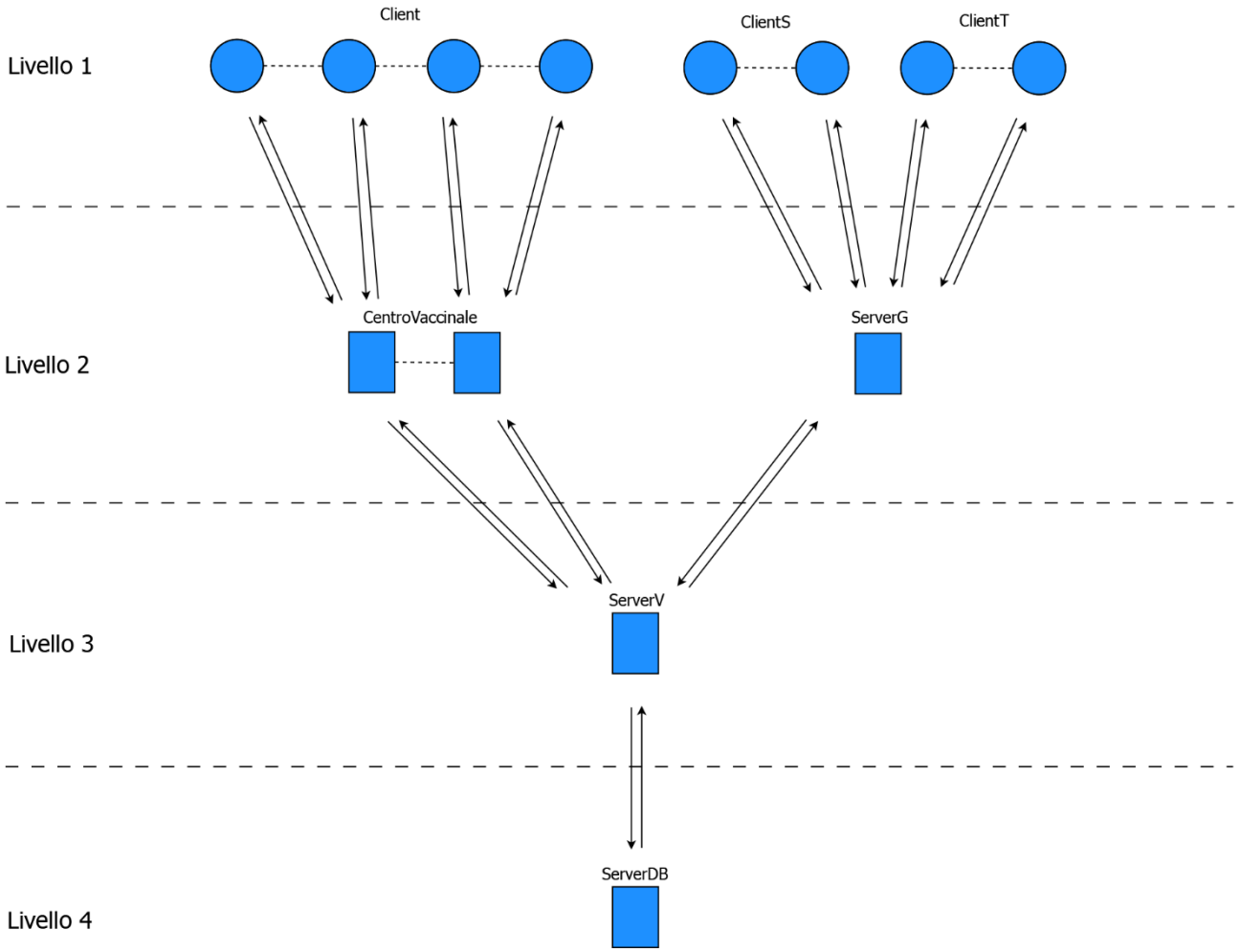


Figura 1. Architettura client-server multilivello descritta dalla soluzione proposta.

Descrizione e schemi del protocollo applicazione

Prima di introdurre il protocollo applicazione per ciascuna operazione di comunicazione tra le entità all'interno dell'architettura, vengono introdotti i pacchetti applicazione scambiati tra le entità all'interno del sistema. Tali pacchetti sono di diverse tipologie e differiscono a partire dall'entità che crea il pacchetto applicazione; tuttavia, solo uno di questi è direttamente scambiato tra le entità.

La struttura del pacchetto che verrà poi effettivamente scambiato è fortemente ispirata al *principio di incapsulamento* sfruttato nei protocolli noti, ovviamente applicato con le dovute differenze. Pertanto, i dati relativi al green pass sono incapsulati in un pacchetto di tipo *Record*, che, a sua volta, è incapsulato in un pacchetto di tipo *ClientPacket*, che, a sua volta, è incapsulato in un pacchetto di tipo *Packet*.

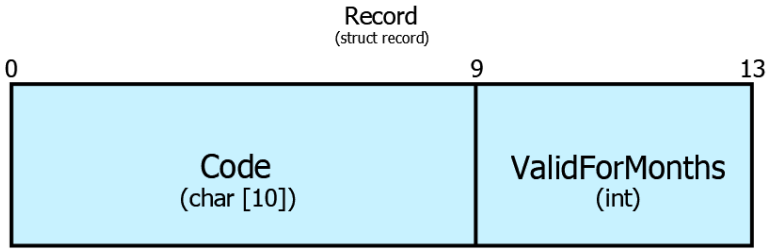
Pacchetti applicazione

Il pacchetto applicazione più semplice è di tipo *Record*, il quale contiene due campi:

- il campo *Code* contiene il codice fiscale rappresentato da una stringa alfanumerica associato al green pass; tale campo è quindi una stringa rappresentata da 10 caratteri al più;
- il campo *ValidForMonths* contiene il numero di mesi per cui il green pass associato a *Code* è valido; tale campo è quindi rappresentato da un intero.

Tipicamente il *Record* è creato dai vari client – *client, clientT, clientS* – ed è incapsulato all'interno di un pacchetto di tipo *ClientPacket* quando i suddetti client richiedono le operazioni messe a disposizione dal sistema.

Di seguito viene mostrata la struttura di tale pacchetto appena descritto e la relativa implementazione in linguaggio C.



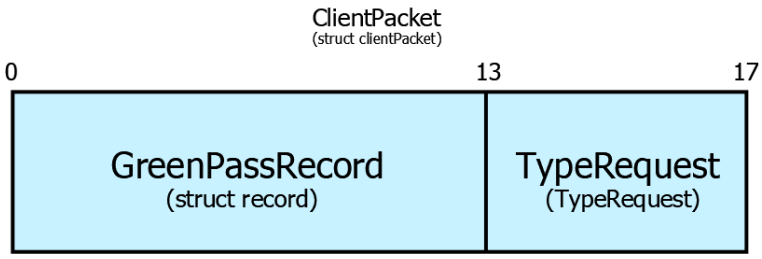

```
struct record {
  char code[MAXIMUM_FISCAL_CODE_LENGTH];
  int validForMonths;
};
```

Figura 2. Struttura del pacchetto applicazione di tipo Record

Il pacchetto applicazione di tipo *ClientPacket* è generato dai client e contiene due campi:

- il campo *GreenPassRecord* contiene le informazioni relative al green pass; tale campo è di tipo *struct record*, ossia il pacchetto applicazione di tipo *Record*;
- il campo *TypeRequest* contiene il tipo di operazione che il client richiede al sistema; tale campo è di tipo *TypeRequest* ed assume valori interi ben definiti – *CHECK, REVOKE, VALIDATE, REGISTER, NO_OPERATION* – i quali descrivono quindi il tipo di operazione che il client vuole effettuare.

Di seguito è riportata la struttura del pacchetto appena descritto.



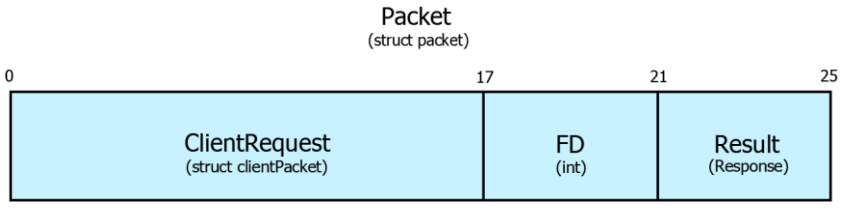

```
struct clientPacket {
  struct record greenPassRecord;
  TypeRequest typeRequest;
};
```

Figura 3. Struttura del pacchetto applicazione di tipo ClientPacket

Il pacchetto applicazione di tipo *Packet*, a differenza degli altri pacchetti applicazione, è l'unica tipologia di pacchetto scambiata tra le entità all'interno del sistema. Questo pacchetto è quindi composto da tre campi:

- il campo *ClientRequest* contiene la richiesta effettuata dal client; tale campo è di tipo *struct clientPacket*, ossia il pacchetto applicazione di tipo *ClientPacket*;
- il campo *FD* contiene il valore del *file descriptor* associato quindi al descrittore che ha ricevuto tale pacchetto; tale informazione serve a ricordare da quale descrittore è giunta la richiesta;
- il campo *Result* ha una doppia valenza poiché può contenere il risultato dell'operazione richiesta dal client oppure lo stato del green pass incapsulato nel campo *ClientRequest* ed a sua volta nel campo *GreenPassRecord*. Tale campo può assumere quindi valori ben definiti: *INVALID* oppure *VALID*.
-

Di seguito è riportata la struttura del pacchetto appena descritto.




```
struct packet {
  struct clientPacket clientRequest;
  int fd;
  Response result;
};
```

Figura 4. Struttura del pacchetto applicazione di tipo Packet.

Protocolli applicazione

A partire dai livelli gerarchicamente inferiori, vengono descritti i protocolli di comunicazione utilizzati. Le figure mostrano il protocollo applicazione adattato per ogni tipologia di client.

Il processo *client* crea un pacchetto di tipo *struct packet* attraverso la funzione *createPacket(char*, TypeRequest)*, la quale restituisce un *puntatore a struct packet*; il primo argomento passato rappresenta il codice fiscale e il secondo rappresenta il tipo di operazione da effettuare. Nel caso specifico, la chiamata sarà del tipo:

*struct packet *packet = createPacket(fiscal_code, REGISTER)*

Il pacchetto generato viene inviato ad un processo *centroVaccinale*, il quale compilerà tale pacchetto inserendo la durata di validità del green pass e lo stato e, a sua volta, invierà tale pacchetto al processo *serverV*.

Il processo *serverV* effettuerà una ricerca per verificare se il pacchetto si trova nella sua "cache" di pacchetti, riempita man mano che sovvengono le richieste. Viene effettuata la ricerca attraverso la funzione *searchGreenPass(struct packet**, *struct packet*, int)* che prende in input un *array di puntatori a struct packet* che rappresenta la "cache", un *puntatore a struct packet* rappresentante la richiesta sovvenuta dal processo *centroVaccinale* ed il numero di pacchetti all'interno della "cache". Tale funzione restituisce un valore negativo se la ricerca ha dato esito negativo, altrimenti l'indice in cui si trova il pacchetto all'interno della "cache".

In caso di esito negativo, il processo *serverV* richiede un'operazione di *SEARCH* al processo *serverDB*, il quale controllerà quindi all'interno del database dei pacchetti, precedentemente caricato da file di testo, se esiste un green pass associato per il codice fiscale contenuto all'interno del pacchetto sovvenutogli, in seguito all'operazione *SEARCH*. Analogamente a quanto accaduto prima, il processo *serverDB* controlla l'esistenza di tale green pass attraverso la funzione *searchGreenPass(struct packet **, struct packet *, int).* L'esito dell'operazione *SEARCH* è inviato al processo *serverV* che si comporterà di conseguenza.

In caso di esito positivo, il pacchetto richiesto viene restituito al processo *serverV* che elaborerà la richiesta sovvenuta e restituirà il risultato di tale elaborazione al processo *centroVaccinale* all'interno di tale pacchetto; il processo *centroVaccinale* restituisce a sua volta il pacchetto ricevuto al processo *client* che ha effettuato la richiesta.

Di seguito è riportato il *Sequence Diagram* relativo al protocollo applicazione appena descritto.

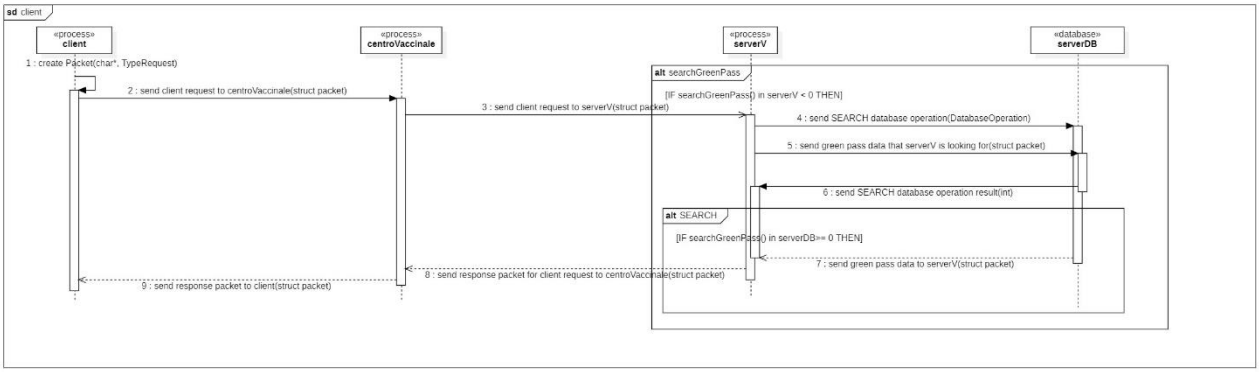


Figura 5. Protocollo applicazione utilizzato da Client, CentroVaccinale, ServerV, ServerDB.

Il processo *clientS* crea un pacchetto di tipo *struct packet* attraverso la funzione *createPacket(char*, TypeRequest)* che restituisce un *puntatore a struct packet*; il primo argomento passato rappresenta il codice fiscale e il secondo rappresenta il tipo di operazione da effettuare. Nel caso specifico, la chiamata sarà del tipo:

*struct packet *packet = createPacket(fiscal_code, CHECK)*

Il pacchetto generato viene inviato ad un processo *serverG*, il quale invierà, a sua volta, tale pacchetto al processo *serverV*.

Il processo *serverV* effettuerà una ricerca per verificare se il pacchetto si trova nella sua "cache" di pacchetti. Viene effettuata la ricerca attraverso la funzione *searchGreenPass(struct packet**, struct packet*, int)* che prende in input un *array di puntatori a struct packet* che rappresenta la "cache", un *puntatore a struct packet* rappresentante la richiesta sovvenuta dal processo *serverG* ed il numero di pacchetti all'interno della "cache". Tale funzione restituisce un valore negativo se la ricerca ha dato esito negativo, altrimenti l'indice in cui si trova il pacchetto all'interno della "cache".

In caso di esito negativo, il processo *serverV* richiede un'operazione di *SEARCH* al processo *serverDB*, il quale controllerà quindi all'interno del database dei pacchetti, precedentemente caricato da file di testo, se esiste un green pass associato per il codice fiscale contenuto all'interno del pacchetto sovvenutogli, in seguito all'operazione *SEARCH*. Analogamente a quanto accaduto prima, il processo *serverDB* controlla l'esistenza di tale green pass attraverso la funzione *searchGreenPass(struct packet **, struct packet *, int).* L'esito dell'operazione *SEARCH* è inviato al processo *serverV* che si comporterà di conseguenza.

In caso di esito positivo, il pacchetto richiesto viene restituito al processo *serverV* che elaborerà la richiesta sovvenuta e restituirà il risultato di tale elaborazione al processo *serverG* all'interno di tale pacchetto; il processo *serverG* restituisce a sua volta il pacchetto ricevuto al processo *clientS* che ha effettuato la richiesta.

Di seguito è riportato il *Sequence Diagram* relativo al protocollo applicazione appena descritto.

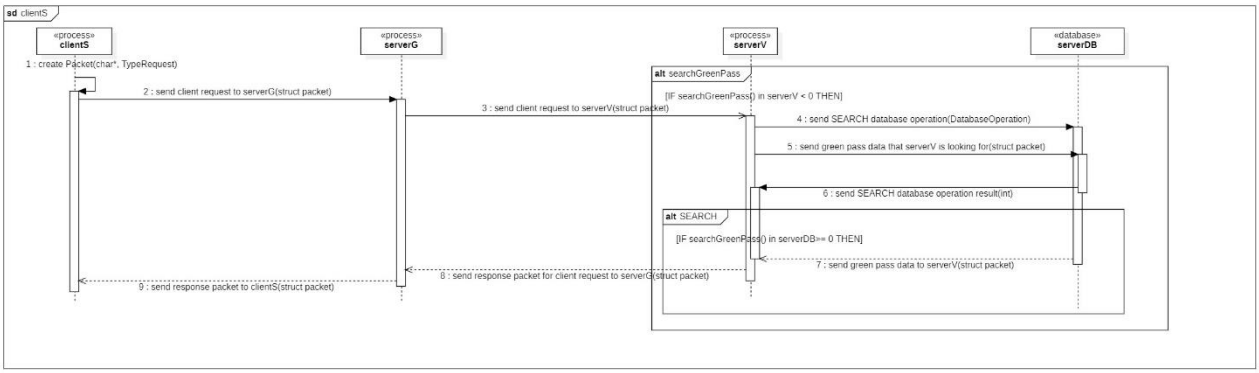


Figura 6. Protocollo applicazione utilizzato da ClientS, ServerG, ServerV, ServerDB.

Il processo *clientT* crea un pacchetto di tipo *struct packet* attraverso la funzione *createPacket(char*, TypeRequest)* che restituisce un *puntatore a struct packet*; il primo argomento passato rappresenta il codice fiscale e il secondo rappresenta il tipo di operazione da effettuare. Nel caso specifico, la chiamata sarà del tipo:

*struct packet *packet = createPacket(fiscal_code, VALIDATE)*

per validare il green pass associato al codice fiscale passato come primo argomento, oppure:

*struct packet *packet = createPacket(fiscal_code, REVOKE)*

per revocare il green pass associato al codice fiscale passato come primo argomento.

Il pacchetto generato viene inviato ad un processo *serverG*, il quale invierà, a sua volta, tale pacchetto al processo *serverV*.

Il processo *serverV* effettuerà una ricerca per verificare se il pacchetto si trova nella sua "cache" di pacchetti. Viene effettuata la ricerca attraverso la funzione *searchGreenPass(struct packet**, struct packet*, int)* che prende in input un *array di puntatori a struct packet* che rappresenta la "cache", un *puntatore a struct packet* rappresentante la richiesta sovvenuta dal processo *serverG* ed il numero di pacchetti all'interno della "cache". Tale funzione restituisce un valore negativo se la ricerca ha dato esito negativo, altrimenti l'indice in cui si trova il pacchetto all'interno della "cache".

In caso di esito negativo, il processo *serverV* richiede un'operazione di *SEARCH* al processo *serverDB*, il quale controllerà quindi all'interno del database dei pacchetti, precedentemente caricato da file di testo, se esiste un green pass associato per il codice fiscale contenuto all'interno del pacchetto sovvenutogli, in seguito all'operazione *SEARCH*. Analogamente a quanto accaduto prima, il processo *serverDB* controlla l'esistenza di tale green pass attraverso la

funzione *searchGreenPass(struct packet **, struct packet *, int).* L'esito dell'operazione *SEARCH* è inviato al processo *serverV* che si comporterà di conseguenza.

In caso di esito positivo, il pacchetto richiesto viene restituito al processo *serverV* che elaborerà la richiesta sovvenuta e restituirà il risultato di tale elaborazione al processo *serverG* all'interno di tale pacchetto; il processo *serverG* restituisce a sua volta il pacchetto ricevuto al processo *clientT* che ha effettuato la richiesta.

Di seguito è riportato il *Sequence Diagram* relativo al protocollo applicazione appena descritto.

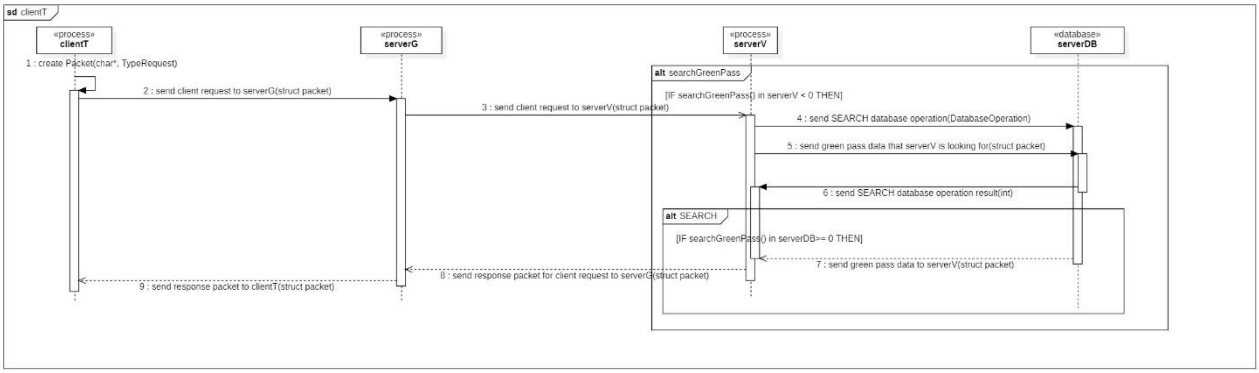


Figura 7. Protocollo applicazione utilizzato da ClientT, ServerG, ServerV, ServerDB.

SEARCH operation

Ogni qualvolta sovviene una richiesta al processo *serverV*, viene quindi effettuata una ricerca del green pass associato al codice fiscale all'interno del pacchetto, prima nella "cache" locale ed eventualmente poi nel database del processo *serverDB.* Viene riportato più nel dettaglio il *Sequence Diagram* di tale operazione, ampiamente descritto nei precedenti protocolli applicazione.

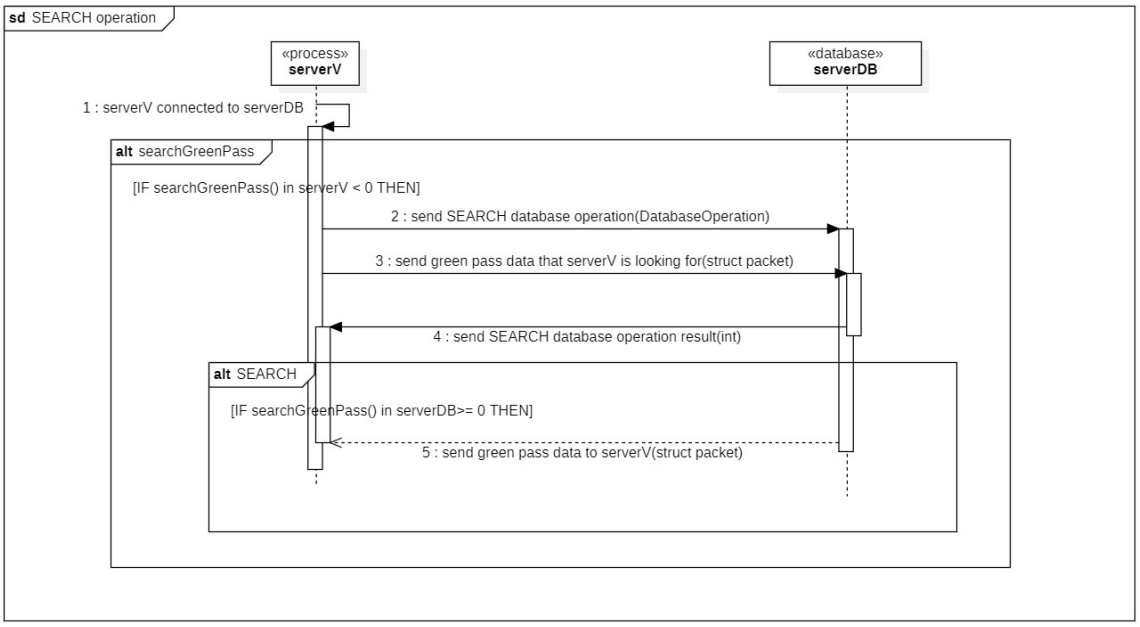


Figura 8. Protocollo applicazione utilizzato da ServerV e ServerDB per l'operazione SEARCH.

STORE operation

Il processo *serverV* può richiedere un'ulteriore operazione al processo *serverDB*. Oltre alla già menzionata operazione *SEARCH*, infatti è possibile effettuare l'operazione *STORE* che permette di aggiornare i dati relativi ai green pass memorizzati nel database. Tale operazione è invocata ad intervalli periodici oppure quando la "cache" del processo *serverV* è satura.

Il processo *serverV* invia l'operazione *STORE* al processo *serverDB*. Le operazioni effettuate dal processo *serverDB* – *SEARCH*, *STORE* – sono memorizzate nel tipo di dato apposito *DatabaseOperation*, come per quanto fatto con il tipo di dato *TypeRequest* oppure *Response*. Tali tipi di dato non sono altro che enumerazioni e quindi assimilabili ad un tipo di dato intero, in questo caso.

Successivamente il processo *serverV* invia il numero di pacchetti da memorizzare, ovvero il numero di pacchetti presenti nella "cache" al processo *serverDB*. I dati relativi ai green pass memorizzati all'interno della "cache" del processo *serverV* sono quindi inviati al processo *serverDB*, il quale, dopo aver ricevuto i pacchetti, aggiorna le proprie copie e risponde con un valore intero, tipicamente 1, per indicare il termine dell'operazione.

Di seguito è riportato il *Sequence Diagram* relativo al protocollo applicazione appena descritto.

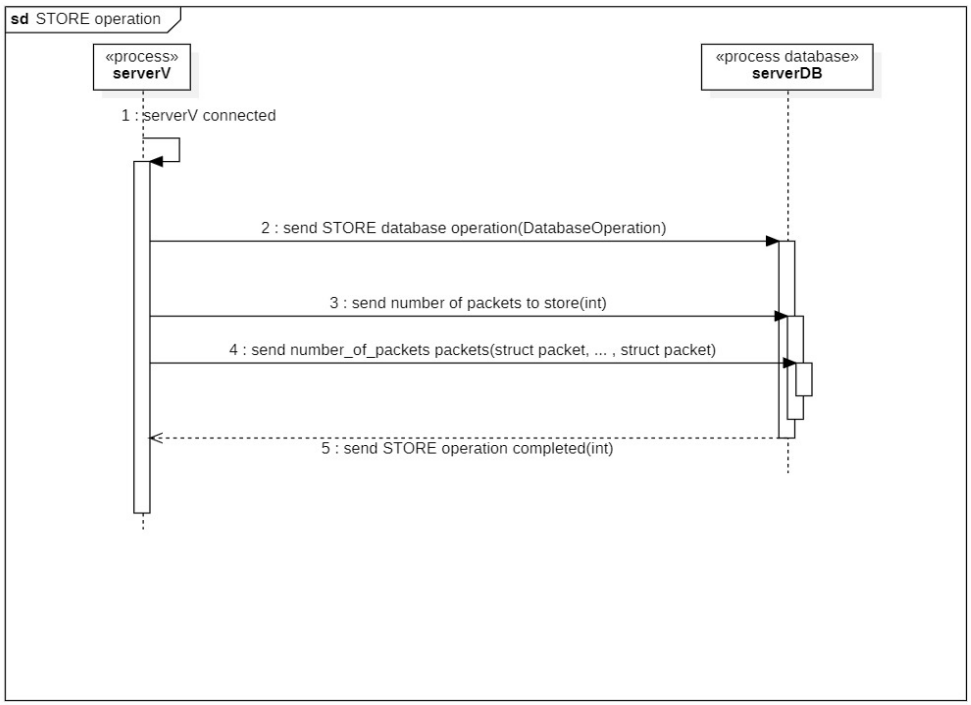


Figura 9. Protocollo applicazione utilizzato da ServerV e ServerDB per l'operazione STORE.

Dettagli implementativi dei client

I client descritti nei precedenti paragrafi – *client, clientS, clientT* – dispongono circa della medesima implementazione, differendo però talvolta nel numero di parametri passati da linea di comando e al server intermedio a cui si rivolgono.

Generalmente, i client sono implementati mediante il modello *I/O bloccante* secondo cui il processo attende durante entrambe le fasi.

Quindi, un client costruisce il pacchetto che contiene l'operazione che si vuole richiedere per un particolare codice fiscale associato quindi ad un certo green pass; il valore del campo *FD* è inizializzato a -1. Il client invia tale pacchetto ad un'entità che si trova al Livello 2 dell'architettura, ovvero un *Centro Vaccinale* oppure il *ServerG* ed attende la risposta da parte di questi.

In seguito alla ricezione della risposta viene quindi mostrato il risultato dell'operazione richiesta.

Dettagli implementativi dei server

Data la struttura iniziale proposta del sistema, i server sono implementati come *server iterativi* di un'architettura multilivello; per tale motivo, le operazioni sono effettuate considerando la complessità – di tempo, prevalentemente – come fattore determinante, allo scopo di assicurare una certa efficienza nella gestione delle richieste dei client.

I server che si trovano al Livello 2 dell'architettura proposta – *centroVaccinale, serverG* – sono piuttosto simili in quanto entrambi non effettuano particolari operazioni ma si occupano di prevalentemente di fare da "tramite" per le richieste dei client al *ServerV*.

Modelli di I/O adoperati

Un server al Livello 2 dell'architettura, quindi, adotta il modello *I/O multiplexing* secondo cui il processo attende su uno o più descrittori mediante una chiamata di sistema *select()* passando come argomenti il massimo descrittore incrementato di uno e il set di descrittori da monitorare in lettura.

Inoltre, questo riesce a soddisfare le richieste più velocemente effettuando un controllo sul campo *FD* del pacchetto ricevuto (di tipo *struct packet*). Se il valore di tale campo è negativo, si tratta quindi di una richiesta appena sovvenuta, pertanto questa sarà inoltrata al processo *serverV*. Se il valore è positivo, si tratta di una risposta da parte del processo *serverV* ed il valore rappresenta il *file descriptor* che ha accettato la richiesta inizialmente; pertanto, la risposta sarà inoltrata al client sul descrittore corrispondente.

Il processo *serverV* adotta il modello *I/O multiplexing*, trattando separatamente il *socket* associato alla comunicazione con il processo *serverDB* e differentemente da quanto fatto dal processo *centroVaccinale* e dal processo *serverG*. Infatti, la chiamata di sistema *select()*, oltre al massimo descrittore incrementato di uno ed il set di descrittori da monitorare in lettura, prende in input come ultimo argomento un *puntatore a struct timeval*, struttura inizializzata opportunamente, che consente di ritornare dalla *select()* secondo il tempo prefissato. Tale struttura, infatti, contiene un campo relativo alla quantità massima di secondi ed un campo relativo alla quantità massima di microsecondi per cui il processo rimane bloccato nella chiamata *select()*.

In questa maniera, se nell'arco di tempo specificato dalla struttura di tipo *struct timeval* non sovviene alcuna richiesta, allora il processo *serverV* ritorna dalla chiamata *select()* e richiede di effettuare un'operazione *STORE* al processo *serverDB* per aggiornare i dati relativi ai green pass, memorizzati nella "cache" del processo *serverV*.

Il processo *serverDB* adotta il modello *I/O sincrono* in quanto non ha necessità di monitorare più descrittori contemporaneamente poiché si suppone che possa ricevere una richiesta di connessione soltanto dall'unico processo *serverV* in esecuzione.

Memorizzazione dei dati

Il processo *serverV* ed il processo *serverDB* mantengono temporaneamente i pacchetti contenenti i dati relativi ai green pass all'interno di una struttura dati di tipo *struct packet***, mantenuta ordinata alfabeticamente rispetto al campo *Code* contenuto in *Record*, a sua volta in *ClientPacket* ed a sua volta in *Packet*, come mostrato nel paragrafo relativo ai pacchetti applicazione. L'ordinamento avviene secondo l'algoritmo *hybrid-quicksort* che permette di applicare gli algoritmi di ordinamento *insertion sort* e *quicksort* nei loro casi migliori. Questa soluzione consente di velocizzare le operazioni di ricerca all'interno di tali strutture, effettuate dalla funzione *searchGreenPass()* implementata mediante *ricerca binaria*.

Quando il processo *serverV* viene interrotto mediante un segnale *SIGINT*, il contenuto della "cache" viene inviato al processo *serverDB* prima di terminare, attraverso un'operazione *STORE*. Quando il processo *serverDB* viene interrotto mediante un segnale *SIGINT*, il contenuto del database, ovvero i pacchetti che contengono i dati relativi ai green pass, sono esportati all'interno di un file di testo, denominato *db.txt*, che viene utilizzato per ricaricare i dati nella struttura dati apposita quando il processo *serverDB* verrà nuovamente mandato in esecuzione. L'esistenza del file di testo *db.txt* è un prerequisito fondamentale per il corretto funzionamento del sistema.

Considerazioni generali e soluzioni alternative

Per la natura del sistema di gestione richiesto dalla traccia, è necessario utilizzare un protocollo di livello trasporto affidabile e sicuro quale è il protocollo *TCP*. Alternativamente, è possibile utilizzare il protocollo *UDP* implementando le caratteristiche di affidabilità, sicurezza e protezione nel protocollo applicazione.

Gli indirizzi utilizzati per il testing del sistema sono coppia di indirizzi *IPv4* e *numero di porta*, tipicamente a partire da *1024*.

Alternativamente alla soluzione proposta, i server potrebbero essere implementati come *server concorrenti*, creando tanti processi figli quante sono le richieste da parte dei client, fornendo quindi una "connessione dedicata".

Per quanto concerne invece l'implementazione del database questa può essere trovata in diverse soluzioni. Un esempio banale è l'utilizzo di un vero e proprio database, in *sqlite3* ad esempio, con cui è possibile interfacciarsi mediante l'omonima libreria per linguaggio C (*sqlite3.h*). Tuttavia tale approccio è stato evitato in quanto avrebbe reso complicata la riproducibilità del sistema proposto.

Un'altra soluzione alternativa si sarebbe potuta applicare memorizzando i pacchetti all'interno di una *hash table* indicizzata attraverso una *funzione di hash*, più o meno complessa, che associ un indice specifico ad una particolare stringa relativa al codice fiscale del green pass collegato, in questo caso.

Manuale utente

Il controllo dell'input è effettuato all'interno del codice sorgente. Tipicamente l'unico input richiesto per ogni file eseguibile avviene da riga di comando.

E' possibile digitare direttamente l'indirizzo *IPv4* oppure il *nome canonico* dell'host ove richiesto.

La lunghezza del codice fiscale è definita dalla macro *MAXIMUM_FISCAL_CODE_LENGTH*, impostata a 10, all'interno del file *greenpass.c*. Il codice fiscale e l'operazione, immessi da linea di comando, sono *case-insensitive*.

Qualora si verificasse un errore a seguito dell'inserimento dei parametri da linea di comando, l'esecuzione è interrotta e la causa dell'errore viene evidenziata a schermo. Ad ogni modo lo stato del sistema è sempre percettibile in quanto per ogni processo viene mostrato il log delle operazioni effettuate o in atto.

La libreria è quindi composta dal file *greenpass.c*, il quale contiene il codice relativo alle funzioni utilizzate, corredate da una breve descrizione dello scopo, dei parametri di input e del valore di ritorno. Gli *header* relativi alle funzioni standard – *POSIX* – sono contenuti all'interno del file *greenpass.h*, richiamato dal file *greenpass.c*.

Istruzioni per la compilazione

Per ogni programma vengono definite le istruzioni in merito alla compilazione. Non è richiesta l'inclusione di alcuna libreria particolare; gli *header* necessari sono inclusi all'interno del file *greenpass.h*. Il compilatore utilizzato è *gcc* siccome il sistema è implementato in linguaggio C.

Programma	Istruzioni per la compilazione
client	*gcc -o client client.c*
clientT	*gcc -o clientT clientT.c*
clientS	*gcc -o clientS clientS.c*
centroVaccinale	*gcc -o centroVaccinale centroVaccinale.c*
serverG	*gcc -o serverG serverG.c*
serverV	*gcc -o serverV serverV.c*
serverDB	*gcc -o serverDB serverDB.c*

Tabella 1. Istruzioni per la compilazione di ciascun programma.

E’ possibile che vengano notificati eventuali *warning* a seguito della fase di compilazione, non rilevanti ai fini della corretta esecuzione dell’applicazione. Tali *warning* dovrebbero essere dovuti alla configurazione del compilatore *gcc* per la macchina in uso, causati dal valore di alcuni flag, ad esempio.

Istruzioni per l’esecuzione

Affinché sia garantito il corretto funzionamento del sistema, è fondamentale che i processi siano mandati in esecuzione secondo il seguente ordine:

1. *serverDB*
2. *serverV*
3. *serverG/centroVaccinale*
4. *client/clientS/clientT*

Tale ordine rispetta quindi la gerarchia tra i server evidenziata dall’architettura proposta ed assicura che non vi siano errori dovuti dall’assenza di un processo server in ascolto.

Tabella delle istruzioni per l’esecuzione

Per ogni file eseguibile ottenuto dalle istruzioni di compilazione, vengono definite le istruzioni in merito all’esecuzione in ambiente UNIX.

File eseguibile	Istruzioni per l’esecuzione	Esempio di istruzione per l’esecuzione
serverDB	*./serverDB <serverDB-ip> <serverDB-port>*	`./serverDB 127.0.0.1 1024`
serverV	*./serverV <serverV-ip> <serverV-port> <serverDB-ip> <serverDB-port>*	*`./serverV 127.0.0.1 1025 127.0.0.1 1024`*
serverG	*./serverG <serverG-ip> <serverG-port> <serverV-ip> <serverV-port>*	*`./serverG 127.0.0.1 1026 127.0.0.1 1025`*
centroVaccinale	*./centroVaccinale <center-ip> <center-port> <serverV-ip> <serverV-port>*	*`./centroVaccinale 127.0.0.1 1027 127.0.0.1 1025`*
client	*./client <center-ip> <center-port> <fiscal_code>*	*`./client 127.0.0.1 1027 ABCD`*
clientT	*./clientT <serverG-ip> <serverG-port> <fiscal_code> <operation>*	*`./clientT 127.0.0.1 1026 ABCD REVOKE`*
clientS	*./clientS <serverG-ip> <serverG-port> <fiscal_code>*	*`./clientS 127.0.0.1 1026 ABCD`*

Tabella 2. Istruzioni per l'esecuzione dei file eseguibili ottenuti dopo il processo di compilazione

Poiché viene utilizzato il protocollo *TCP* a livello Trasporto, una particolare porta non può essere condivisa tra i processi in esecuzione.

Tabella di descrizione dei parametri

Di seguito viene riportata una tabella che descrive il significato dei parametri da passare da linea di comando.

Parametro	Descrizione del parametro
center-ip	L'indirizzo *IPv4* oppure il nome canonico del *Centro Vaccinale* a cui ci si riferisce
center-port	La porta su cui è in ascolto il processo *centroVaccinale*
fiscal_code	Il codice fiscale associato al green pass di cui si vuole effettuare l'operazione
operation	L'operazione che si vuole effettuare sul green pass. Tipicamente tale parametro è messo a disposizione unicamente al processo *clientT* che può validare un green pass a seguito di una guarigione con *VALIDATE* oppure revocare un green pass a seguito di un contagio con *REVOKE*
serverDB-ip	L'indirizzo *IPv4* oppure il nome canonico del *ServerDB* a cui ci si riferisce
serverDB-port	La porta su cui è in ascolto il processo *serverDB*
serverG-ip	L'indirizzo *IPv4* oppure il nome canonico del *ServerG* a cui ci si riferisce
serverG-port	La porta su cui è in ascolto il processo *serverG*
serverV-ip	L'indirizzo *IPv4* oppure il nome canonico del *ServerV* a cui ci si riferisce
serverV-port	La porta su cui è in ascolto il processo *serverV*

Tabella 3. Descrizione dei parametri richiesti da linea di comando per l'esecuzione

Esempio di esecuzione

L'esempio di esecuzione ha l'obiettivo di mostrare il funzionamento del sistema, ponendo l'attenzione sulle funzionalità che tale sistema deve soddisfare, ovvero, la registrazione, il controllo, la validazione e la revoca del green pass associato ad un particolare codice fiscale. Pertanto, viene mostrata la registrazione di un green pass associato al codice "*AABBCC00*", per opera di un processo *client*¸ in seguito ad una vaccinazione.

Il primo processo mandato in esecuzione è il processo *serverDB* fornendo la coppia *IPv4-porta* assegnatagli. Nel caso specifico viene passato il comando:

```
./serverDB 127.0.0.1 1024
```

Il secondo processo mandato in esecuzione è il processo *serverV* fornendo una prima coppia *IPv4-porta* che rappresenta l'indirizzo assegnatogli ed una seconda coppia relativa all'indirizzo del processo *serverDB* al quale il processo *serverV* deve connettersi. Nel caso specifico viene passato il comando:

```
./serverV 127.0.0.1 1025 127.0.0.1 1024
```

Il terzo processo mandato in esecuzione è il processo *centroVaccinale* fornendo una prima coppia *IPv4-porta* che rappresenta l'indirizzo assegnatogli ed una seconda coppia relativa all'indirizzo del processo *serverV* al quale il processo *centroVaccinale* deve connettersi. Nel caso specifico viene passato il comando:

```
./centroVaccinale 127.0.0.1 1027 127.0.0.1 1025
```

Il quarto processo mandato in esecuzione è il processo *client* fornendo la coppia *IPv4-porta* che identifica il processo *centroVaccinale* mandato in esecuzione e il codice fiscale di cui si vuole registrare il green pass. Nel caso specifico viene passato il comando:

```
./client 127.0.0.1 1027 AABBCC00
```

Di seguito è riportato uno *screenshot* dimostrativo per tale operazione. In alto a sinistra, in alto a destra, in basso a sinistra, in basso a destra sono riportati, rispettivamente, il funzionamento del processo *serverDB*, *serverV*, *centroVaccinale*, *client*.

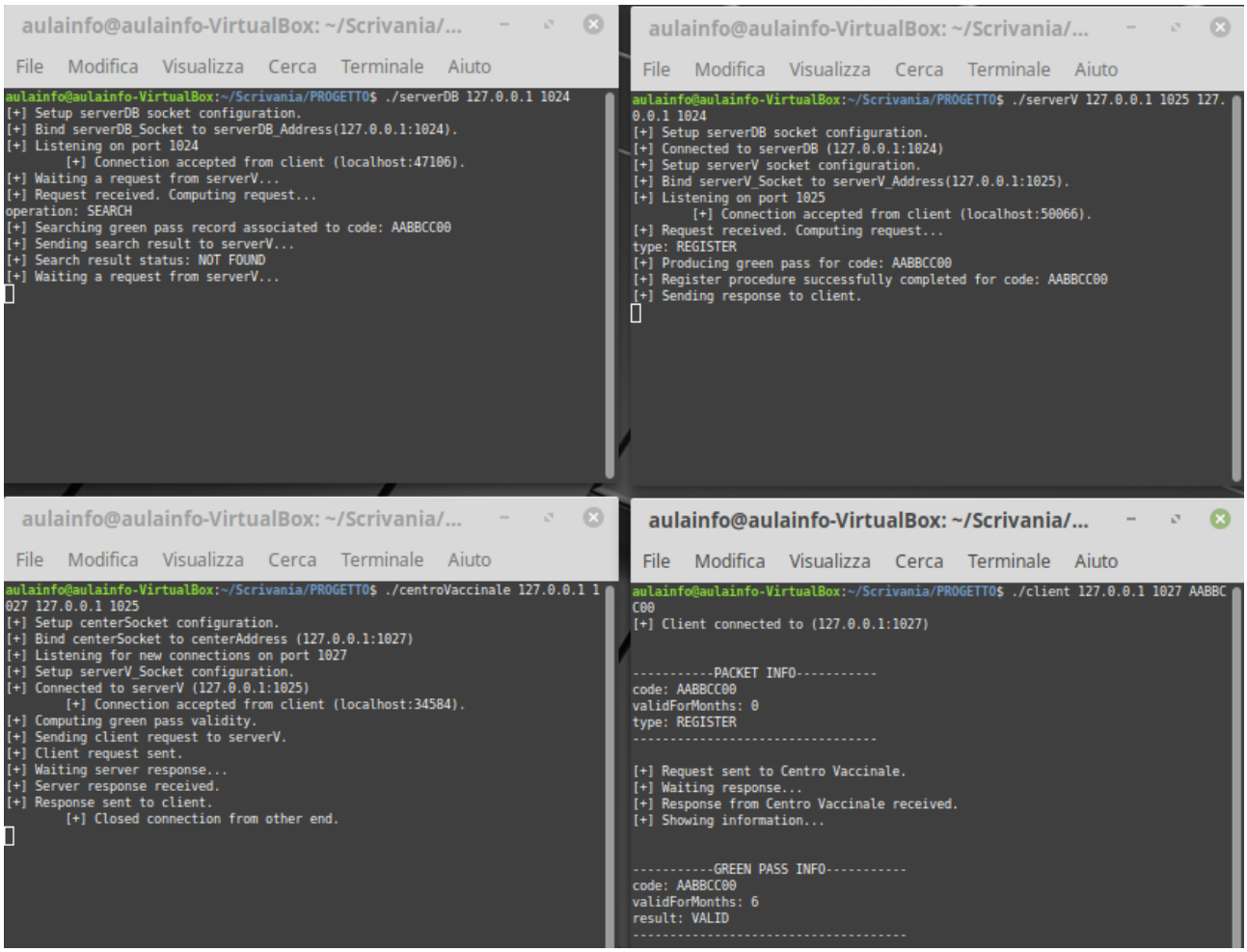


Figura 10. Registrazione di un green pass

Dai messaggi stampati a schermo, è possibile percepire chiaramente l'operazione richiesta e il risultato di tale operazione. Il green pass è registrato correttamente, come previsto.

E’ possibile verificare lo stato del green pass appena registrato attraverso il processo *clientS*.

Pertanto, viene mandato in esecuzione il processo *serverG* fornendo una prima coppia *IPv4-porta* che rappresenta l’indirizzo assegnatogli ed una seconda coppia relativa all’indirizzo del processo *serverV* al quale il processo *serverG* deve connettersi. Nel caso specifico viene passato il comando:

```
./serverG 127.0.0.1 1026 127.0.0.1 1025
```

Successivamente il processo *clientS* viene mandato in esecuzione fornendo la coppia *IPv4-porta* che identifica il processo *serverG* mandato in esecuzione e il codice fiscale di cui si vuole controllare il green pass. Nel caso specifico viene passato il comando:

```
./clientS 127.0.0.1 1026 AABBCC00
```

Di seguito è riportato uno *screenshot* dimostrativo per tale operazione. In alto a sinistra, in alto a destra, in basso a sinistra, in basso a destra sono riportati, rispettivamente, il funzionamento del processo *serverDB*, *serverV*, *serverG*, *clientS*.

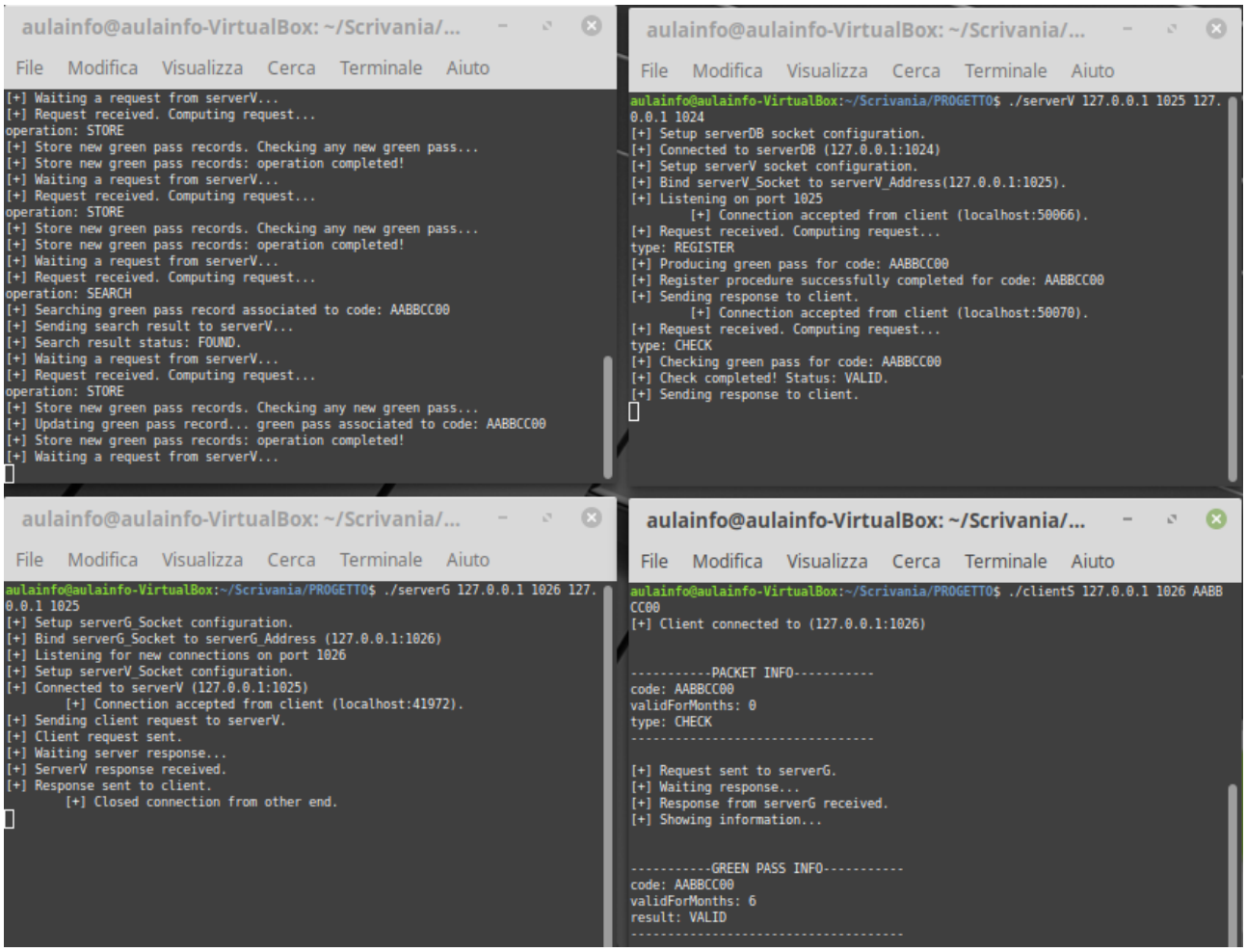


Figura 11. Controllo di un green pass valido

Le informazioni relative al green pass coincidono con quelle mostrate in precedenza, pertanto, il controllo è stato effettuato con successo.

E’ possibile revocare il green pass appena registrato, in seguito ad un contagio, attraverso il processo *clientT*.

Il processo *clientT* viene mandato in esecuzione fornendo la coppia *IPv4-porta* che identifica il processo *serverG* mandato in esecuzione, il codice fiscale di cui si vuole revocare il green pass e l’operazione *REVOKE*. Nel caso specifico viene passato il comando:

```
./clientT 127.0.0.1 1026 AABBCC00 REVOKE
```

Di seguito è riportato uno *screenshot* dimostrativo per tale operazione. In alto a sinistra, in alto a destra, in basso a sinistra, in basso a destra sono riportati, rispettivamente, il funzionamento del processo *serverDB*, *serverV*, *serverG*, *clientT*.

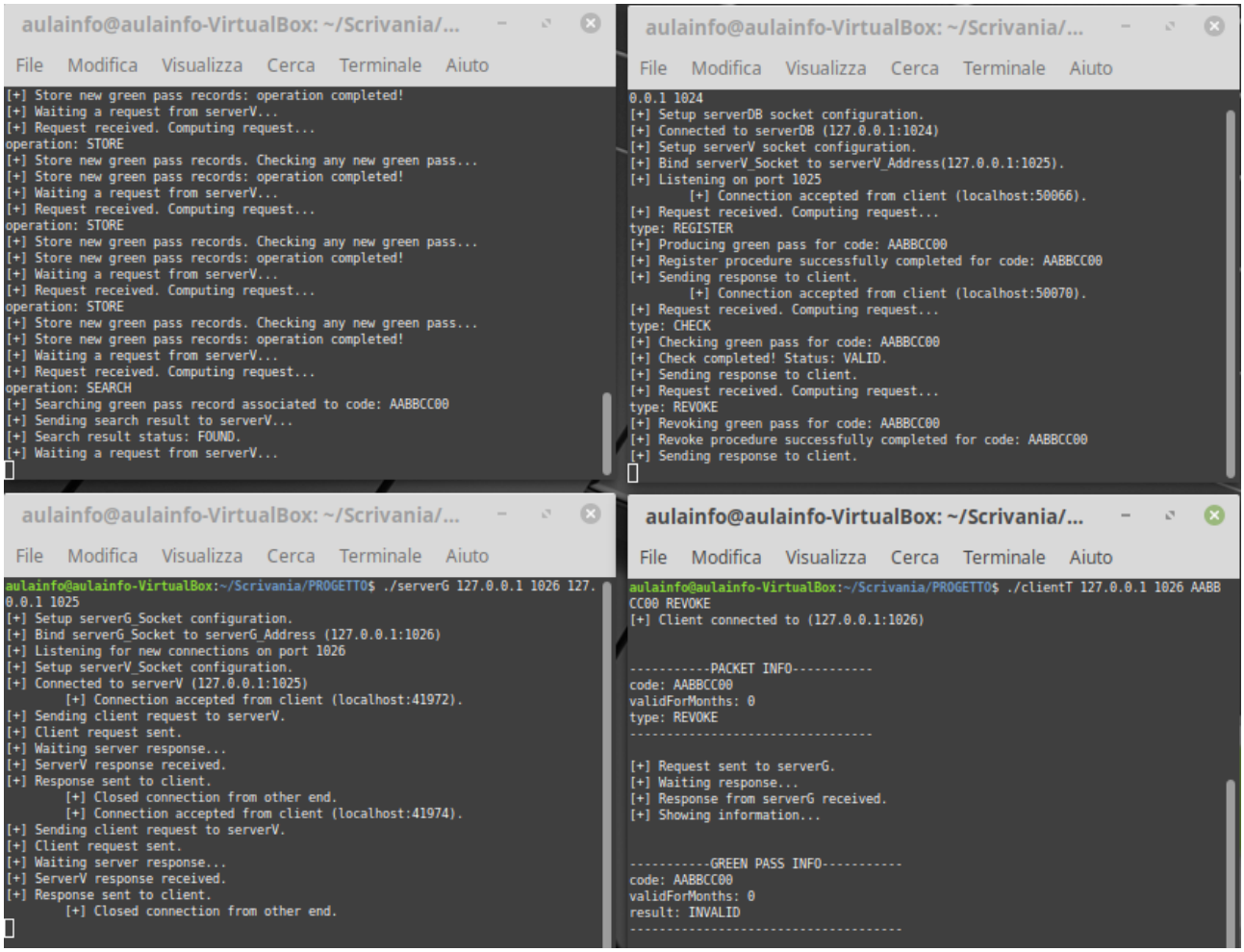


Figura 12. Revoca di un green pass

Come mostrato dalla figura, il green risulta non essere più valido e la durata di validità è impostata a 0, coerentemente con l’operazione di revoca del green pass.

Per completezza viene mostrato uno *screenshot* che mostra il controllo del green pass a seguito della revoca. I risultati mostrati coincidono con lo stato del green pass.

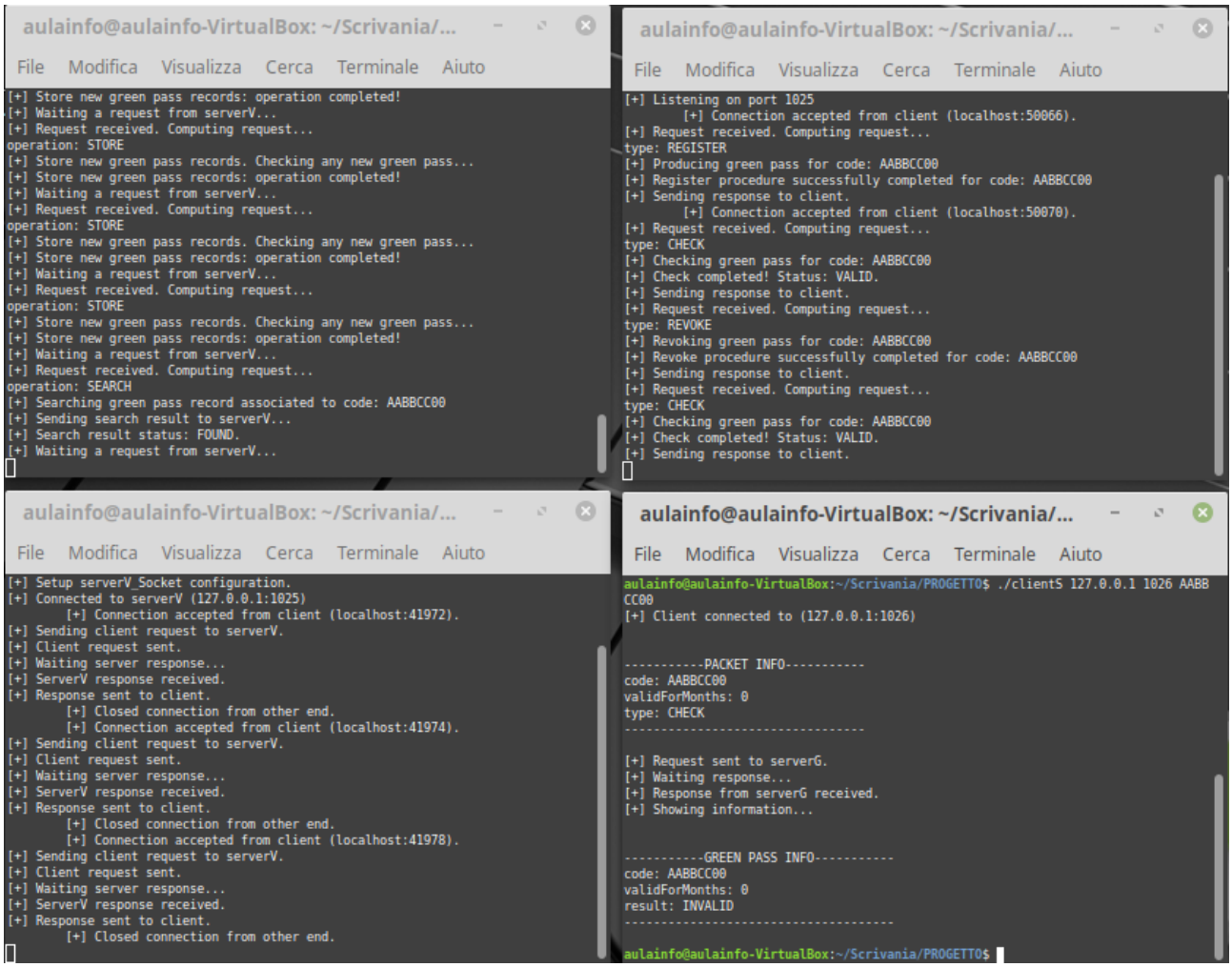


Figura 13. Controllo di un green pass non valido

E’ possibile validare il green pass appena registrato, a seguito di una guarigione, attraverso il processo *clientT*.

Il processo *clientT* viene mandato in esecuzione fornendo la coppia *IPv4-porta* che identifica il processo *serverG* mandato in esecuzione, il codice fiscale di cui si vuole validare il green pass e l’operazione *VALIDATE*. Nel caso specifico viene passato il comando:

```
./clientT 127.0.0.1 1026 AABBCC00 VALIDATE
```

Di seguito è riportato uno *screenshot* dimostrativo per tale operazione. In alto a sinistra, in alto a destra, in basso a sinistra, in basso a destra sono riportati, rispettivamente, il funzionamento del processo *serverDB*, *serverV*, *serverG*, *clientT*.

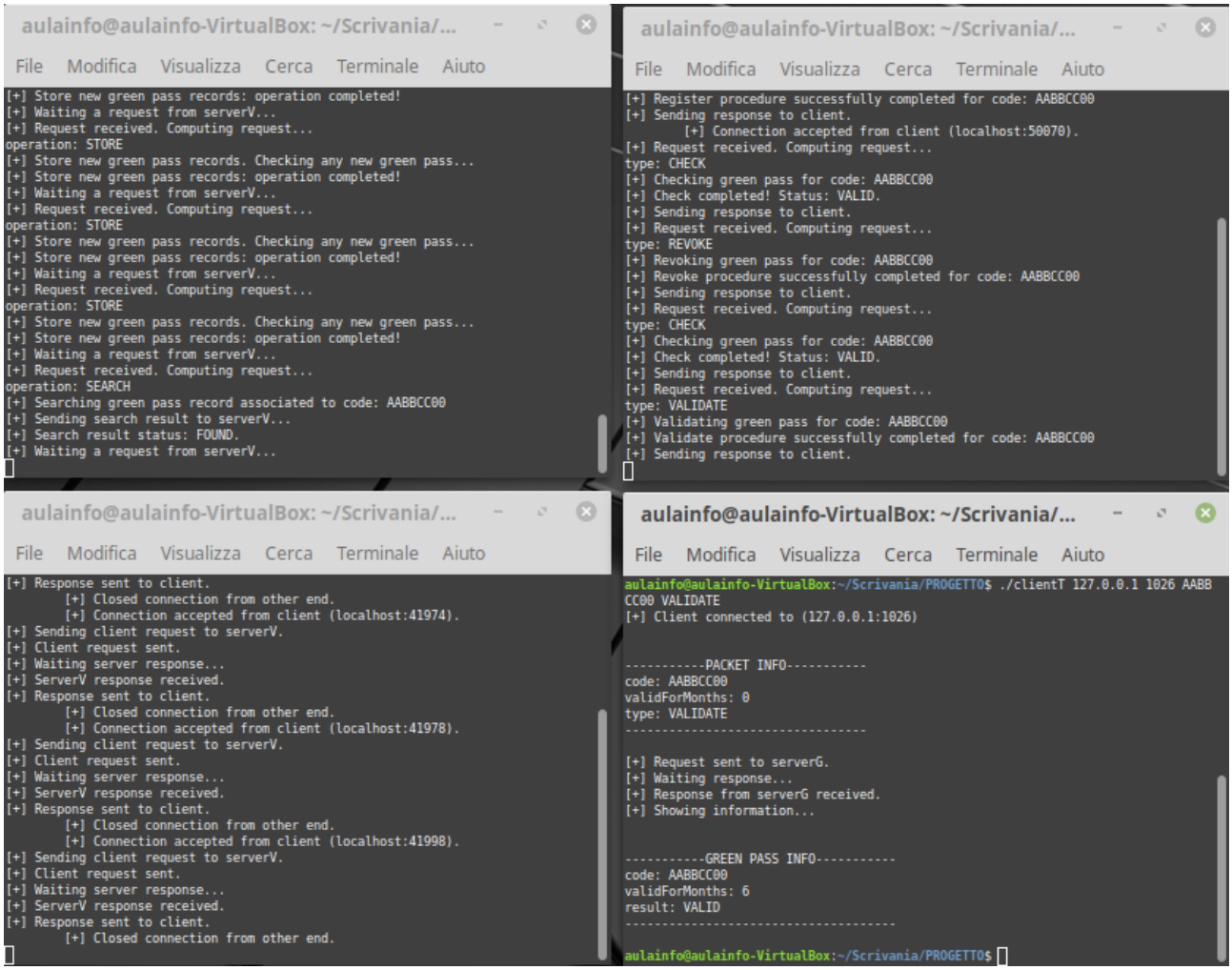


Figura 14. Validazione di un green pass revocato