Studente: Francesco Gallicchio
Matricola: 579131 (corso B)
Indirizzo mail: f.gallicchio1@studenti.unipi.it

Relazione sul progetto di Laboratorio di Sistemi Operativi
(a.a. 2020-21)

1. *Il protocollo File Storage*

La comunicazione tra client e server avviene mediante il protocollo FSP (File Storage Protocol). Lo scambio dei messaggi avviene secondo il paradigma richiesta-risposta. Il server deve stare in ascolto dei messaggi di richiesta da parte dei client e, dopo l'elaborazione delle richieste contenute nei messaggi, invia i messaggi di risposta ai relativi client. La connessione tra client e server è persistente, ovvero, una volta instaurata, può avvenire più di uno scambio di messaggi tra client e server senza che essa venga chiusa. Ad eccezione dei file che vengono trasferiti dal client al server e viceversa (contenuti nel campo DATA), sia i messaggi di richiesta che quelli di risposta sono codificati in testo ASCII.

Per aprire la connessione con il server il client deve connettersi ad esso attraverso un socket (connessione di tipo AF_UNIX) e ricevere dal server un messaggio di risposta con codice 220 se la connessione è stata instaurata con successo o 421 altrimenti.

Per chiudere la connessione il client deve inviare al server un messaggio di richiesta QUIT. Se la connessione è stata chiusa con successo il client riceverà un messaggio di risposta con codice 221.

1.1 *Il formato dei messaggi di richiesta*

I messaggi di richiesta sono composti da due righe ognuna delle quali termina con i caratteri ritorno carrello e fine linea (CRLF). Nella prima riga sono presenti il nome del comando (COMMAND) seguito da uno spazio (SP) e dall'eventuale argomento (ARG). Nella seconda riga sono presenti la lunghezza del campo DATA espressa in byte (DATA_LEN) seguita da uno spazio e il campo contenente gli eventuali dati (DATA).

Il campo DATA può non essere vuoto e, di conseguenza, DATA_LEN può essere maggiore di 0, soltanto nel caso di messaggi WRITE e APPEND.

COMMAND	SP	ARG	CRLF
DATA_LEN	SP	DATA	CRLF

1.2 *I comandi e i relativi argomenti dei messaggi di richiesta*

Comando	**Argomento**	**Descrizione**
APPEND	*pathname*	Chiede al server di scrivere in append i dati contenuti nel campo DATA nel file specificato in *pathname*. L'operazione di scrittura in append avviene in modo atomico. Il file deve essere aperto prima di poterlo scrivere altrimenti il server risponde con il relativo messaggio di errore. In caso di capacity miss il server include nel campo DATA del messaggio di risposta i file espulsi.
CLOSE	*pathname*	Chiede al server di chiudere il file specificato in *pathname*. Il file deve essere aperto prima di poterlo chiudere, altrimenti il server risponde con un messaggio di errore. Se l'utente che chiude il file ne detiene anche la lock, allora il server la rilascia automaticamente.
LOCK	*pathname*	Chiede al server di detenere la lock sul file aperto/creato specificato in *pathname*. Il file deve essere aperto prima di poter eseguire tale operazione, altrimenti il server risponde con un messaggio di errore. Se un altro utente detiene la lock sul file, il server, prima di inviare il messaggio di risposta, attenderà che essa venga rilasciata. Un utente può attendere la lock solo per un certo periodo di tempo (4 secondi nella mia implementazione). Se il tempo scade prima che il client riesca a ottenere la lock, allora il server risponde con un messaggio di errore.

Comando	Argomento	Descrizione
OPEN	*pathname*	Chiede al server di aprire il file specificato in *pathname*. Se il file esiste, il server apre il file in lettura e scrittura (solo in append) e risponde con un messaggio di successo. Altrimenti, il server risponde con un messaggio di errore.
OPENC	*pathname*	Chiede al server di creare e aprire il file specificato in *pathname*. Se il file non esiste, il server crea e apre il file in lettura e scrittura (la prima scrittura deve essere eseguita con l'invio di un messaggio WRITE solo dopo aver ottenuto la lock con un messaggio LOCK) e risponde con un messaggio di successo. Altrimenti, il server risponde con un messaggio di errore. In caso di capacity miss il server include nel campo DATA del messaggio di risposta il file espulso.
OPENCL	*pathname*	Chiede al server di creare e aprire il file specificato in *pathname* in modalità locked. Se il file non esiste, il server crea e apre il file in lettura e scrittura in modalità locked (la prima scrittura deve essere eseguita con un messaggio WRITE) e risponde con un messaggio di successo. Altrimenti, il server risponde con un messaggio di errore. In caso di capacity miss il server include nel campo DATA del messaggio di risposta il file espulso.
OPENL	*pathname*	Chiede al server di aprire il file specificato in *pathname* in modalità locked. Se il file esiste e nessun altro utente detiene la lock su di esso, il server apre il file in lettura e scrittura in modalità locked e risponde con un messaggio di successo. Se un altro utente detiene la lock sul file, il server, prima di inviare il messaggio di risposta, attenderà che essa venga rilasciata. Un utente può attendere la lock solo per un certo periodo di tempo (4 secondi nella mia implementazione). Se il tempo scade prima che il client riesca a ottenere la lock, allora il server risponde con un messaggio di errore.
QUIT		Chiede al server di chiudere la connessione. Il campo ARG è vuoto.
READ	*pathname*	Chiede al server di inviare una copia del contenuto del file specificato in *pathname*. Il file deve essere aperto prima di poterlo leggere e nessun altro client deve detenerne la lock altrimenti il server risponde con il relativo messaggio di errore.
READN	*N*	Chiede al server di inviare una copia di *N* file qualsiasi. Se il server ha meno di *N* file disponibili, li invia tutti. Se *N* <= 0, la richiesta al server è quella di inviare tutti i file memorizzati al suo interno. Non è necessario aprire i file con uno dei messaggi OPEN* prima di inviare il messaggio di richiesta READN. Se un altro utente detiene la lock su un file, esso non viene considerato come disponibile e quindi non verrà restituito nel campo DATA del messaggio di risposta.
REMOVE	*pathname*	Chiede al server di rimuovere il file specificato in *pathname*. Il file deve trovarsi in stato locked da parte del client che ne fa richiesta. Se un altro client ne detiene la lock, allora l'operazione fallisce e il server risponde con un messaggio di errore.
UNLOCK	*pathname*	Chiede al server di rilasciare la lock sul file specificato in *pathname*. Il file deve essere aperto e bisogna detenerne la lock prima di poter eseguire tale operazione, altrimenti il server risponde con un messaggio di errore.
WRITE	*pathname*	Chiede al server di scrivere tutto il file contenuto nel campo DATA nel file specificato in *pathname*. Il file deve essere aperto con un messaggio OPENC (e in seguito con uno di LOCK) o OPENCL prima di poterlo scrivere.

1.3 *Il formato dei messaggi di risposta*

Anche i messaggi di risposta sono composti da due righe ognuna delle quali termina con i caratteri ritorno carrello e fine linea. Nella prima riga sono presenti il codice di risposta (CODE) seguito da uno spazio (SP) e

da una descrizione (*descrizione*). Nella seconda riga sono presenti la lunghezza del campo DATA espressa in byte (DATA_LEN) seguita da uno spazio e il campo contenente gli eventuali dati (DATA).

Il campo DATA può non essere vuoto e, di conseguenza, DATA_LEN può essere maggiore di 0, soltanto nel caso di messaggi di risposta ai comandi READ e READN. Inoltre, il server potrebbe inserire nel campo DATA i file che ha espulso dopo aver eseguito un comando WRITE o APPEND.

CODE	SP	*descrizione*	CRLF
DATA_LEN	SP	DATA	CRLF

1.4 *I codici dei messaggi di risposta*

I codici dei messaggi di risposta sono ispirati al protocollo FTP.

Il campo *descrizione* relativo a un codice può essere modificato purché si mantenga una semantica affine. La modifica del campo *descrizione* può rivelarsi necessaria se si intente dare maggiori dettagli o se si vuole dare una risposta più specifica.

I messaggi di errore hanno sempre il campo DATA vuoto.

Codice	**Descrizione**
421	Service not available, closing connection.
501	Syntax error, message unrecognised.
550	Requested action not taken. File not found.
552	Requested file action aborted. Exceeded storage allocation.
554	Requested action not taken. No access.
555	Requested action not taken. File already exists.
556	Cannot perform the operation.

I messaggi di successo sono i seguenti:

Codice	**Descrizione**
200	The requested action has been successfully completed.
220	Service ready.
221	Service closing connection.

1.5 *Il formato del campo DATA*

Il campo DATA contiene per ogni dato il suo nome (PATHNAME) seguito da uno spazio, la sua dimensione in byte (SIZE) seguita da uno spazio e il dato stesso (*data*) seguito da uno spazio.

Se DATA_LEN è uguale a 0, allora il campo DATA è vuoto.

PATHNAME_1	SP	SIZE_1	SP	*data_1*	SP
…					
PATHNAME_N	SP	SIZE_N	SP	*data_N*	SP

2. *Formato del file di configurazione (~/.file_storage/config.txt)*

Ogni riga del file di configurazione ha il seguente formato: il nome del parametro seguito dal simbolo di uguaglianza e dal valore assegnato ad esso. Ogni riga termina con un carattere fine linea (LF).

La lista dei parametri all'interno del file di configurazione può presentarsi in un qualsiasi ordine. Se nel file manca la definizione di un parametro, allora il server assegnerà il valore di default a tale parametro. Se il file contiene più parametri con lo stesso nome, allora il server considererà soltanto l'ultima definizione.

PARAM_NAME	=	*value*	LF

Lista dei possibili parametri:

Nome del parametro (PARAM_NAME)	**Tipo del valore (*value*)**	**Valore di default**	**Descrizione**
SOCKET_FILE_NAME	<string>	/tmp/ file_storage.sk	Nome del socket sul quale il server sta in ascolto per accettare nuove connessioni.
LOG_FILE_NAME	<string>	~/.file_storage/ file_storage.log	Nome del file di log sul quale il server scrive tutte le operazioni che vengono richieste dai client e quelle di gestione interna.
FILES_MAX_NUM	<int>	1000	Numero massimo di file memorizzabili sul server.
STORAGE_MAX_SIZE	<int>	64	Spazio massimo di memorizzazione sul server in MB.
MAX_CONN	<int>	16	Numero massimo di connessioni con i client.
WORKER_THREADS_NUM	<int>	4	Numero dei thread worker che eseguono le richieste dei client concorrentemente sul server.

3. *Formato del file di log*
Nel file di log vengono scritte le seguenti informazioni (tra parentesi viene indicato il numero associato all'evento usato nella tabella sottostante):
- Inizio dell'esecuzione del server (1);
- Termine dell'esecuzione del server (2);
- Apertura di una connessione (3);
- Chiusura di una connessione (4);
- Esito positivo dell'esecuzione di un comando (5);
- Esito negativo dell'esecuzione di un comando (6);
- Evento di capacity miss (7).

Il formato in cui appaiono le informazioni è descritto nella seguente tabella:

Evento	**Formato**													
1	hh:mm:ss	SP	SERVER_PROCESS_STARTED											LF
2			SERVER_PROCESS_TERMINATED											LF
3			CONNECTION_OPENED:					SP	sfd					LF
4			CONNECTION_CLOSED:					SP	sfd			SP	(description)	LF
5			thread_id:	SP	sfd	SP	CMD	SP	arg	SP	SUCCESS	SP	(bytes)	LF
6			thread_id:	SP	sfd	SP	CMD	SP	arg	SP	FAILURE	SP	(description)	LF

Evento	Formato							
7			CAPACITY_MISS					LF
			REJECTED_FILE:	SP	rejected_file_name	SP	(bytes)	LF

Legenda:
- arg: argomento del comando CMD fsp (non presente se il comando è QUIT);
- bytes: numero di byte letti, scritti o rimossi;
- CMD: nome del comando fsp;
- description: descrizione dell'errore;
- hh:mm:ss: ore, minuti e secondi del momento in cui è stato generato il messaggio;
- LF: line feed (carattere fine linea);
- rejected_file_name: nome del file espulso dalla cache a seguito di capacity miss;
- sfd: socket file descriptor (identifica il client);
- SP: space (spazio);
- thread_id: identificatore del thread worker che esegue il comando.

Note:
- Se l'evento è la chiusura di una connessione, il campo description è presente solo se la connessione è stata chiusa a causa di un errore;
- Se l'evento è l'esito positivo di un comando, il campo bytes è presente solo se il comando è APPEND, READ, READN, REMOVE o WRITE;
- Se l'evento è quello di capacity miss, allora prima scrive il messaggio CAPCITY_MISS e in seguito scrive per ogni file espulso dalla cache un messaggio REJECTED_FILE contenente il nome del file rimosso assieme alla sua dimensione in byte.

4. *Link al repository Git pubblico*
https://github.com/franchik92/file_storage.git