Università degli Studi di Enna "Kore"
Facoltà di Ingegneria e Architettura
Corso di Studi in Ingegneria Informatica

Sistemi Operativi

RELAZIONE ELABORATO

STUDENTE 1	
Nome	Flavio
Cognome	Esposito
Matricola	19031681
Email	flavio.esposito@unikorestudent.it

STUDENTE 2	
Nome	Luigi Pio
Cognome	Faletra
Matricola	19030782
Email	luigipio.faletra@unikorestudent.it

SPECIFICHE ELABORATO	
Titolo	**GarageDucati: configurazione e realizzazione moto Ducati**
Descrizione dettagliata	**GarageDucati** è un programma che, attraverso un menù iniziale, offre quattro funzionalità: 1. **Creazione configurazione moto Ducati**, in cui bisogna effettuare delle scelte riguardanti il modello, il colore, il diametro dei cerchi, la sella, lo scarico, la frizione, il telaio, gli pneumatici, le pedane ed il cavalletto della Ducati, in questo ordine. Sono state previste 5 opzioni di scelta per il diametro dei cerchi, la sella, la frizione, il telaio, gli pneumatici, le pedane ed il cavalletto, mentre si hanno 10 opzioni di scelta per il modello, il colore e lo scarico. Ogni opzione viene selezionata attraverso un numero da 1 a 5 o da 1 a 10, in base al numero di opzioni previste. Alla fine della fase di scelta, viene effettuato il salvataggio su file di testo, con nome scelto dall'utente, della data e dell'ora di creazione della configurazione della relativa moto, del codice seriale costituito dai numeri rappresentanti le opzioni scelte, del prezzo complessivo della suddetta moto e di una frase contenente delle istruzioni sul corretto utilizzo del programma. Il file viene salvato in una cartella denominata "Ducati GarageMoto"; 2. **Caricamento configurazione moto Ducati**, in cui viene richiesto il codice seriale presente nel file creato tramite la precedente funzionalità, da inserire cifra per cifra nello stesso ordine della sequenza del suddetto file. Ogni valore, come già evidenziato sopra, corrisponde all'opzione scelta per ognuna delle parti da configurare della moto. Alla fine della fase di inserimento del seriale, vengono visualizzate le scelte riguardanti il modello, il colore, il diametro dei cerchi, la sella, lo scarico, la frizione, il telaio, gli pneumatici, le pedane ed il cavalletto della Ducati, ognuna dipendente dalla relativa cifra del seriale, ed il prezzo complessivo della moto configurata;

	3. **Contatto operatore GarageDucati in chat per assistenza**, in cui l'utente inserisce il proprio nome ed inizia una conversazione con uno dei 20 operatori GarageDucati previsti. Questi sono 10 uomini e 10 donne e l'operatore selezionato per la chat viene estratto in maniera casuale. La chiusura della chat viene effettuata dal client inserendo la parola "Chiudi"; 4. **Chiusura immediata del programma**, qualora non si volesse intraprendere nessuna delle azioni elencate in precedenza.
System call utilizzate	❏ processi: • **getpid()**, sia nel client sia nel server. Nel client è utilizzata per assegnare il PID ad una variabile di tipo int da richiamare all'atto di terminazione del processo. Nel server è applicata per visualizzare a schermo il PID del processo in esecuzione; • **getppid()**, solo nel server. E' utilizzata per per visualizzare a schermo il PPID del processo in esecuzione, ossia il PID del processo padre di quello in esecuzione; • **fork()**, sia nel client sia nel server. Nel client è utilizzata per creare un processo figlio per la gestione del menù iniziale. Nel server è applicata per realizzare un processo figlio, un processo nipote e due processi pronipoti usati rispettivamente per la gestione del menù iniziale, per la gestione della configurazione della moto scelta e del file ad essa associato e per la creazione della cartella e lo spostamento del suddetto file in quest'ultima; • **exit()**, sia nel client sia nel server. In entrambi è utilizzata per far terminare un processo in maniera corretta, tramite l'argomento "EXIT_SUCCESS", o per una terminazione di un processo con errori, attraverso l'argomento "EXIT_FAILURE; • **wait()**, solo nel server. E' utilizzata per attendere la terminazione di uno qualsiasi dei processi figli generati in precedenza;

	• **waitpid()**, solo nel client. E' utilizzata per attendere la terminazione di uno specifico processo, il processo figlio, generato per la gestione del menù iniziale; • **execl()**, solo nel server. E' utilizzata per effettuare la vera e propria creazione della cartella "Ducati GarageMoto" ed il reale spostamento del file generato una volta costruita una configurazione all'interno di tale cartella tramite la sostituzione del processo corrente con un nuovo programma di cui viene passato il pathname come argomento. ❏ segnali tra processi: • **raise()**, solo nel client. E' utilizzata per richiamare l'handler che si occupa di controllare il semaforo necessario per instaurare la comunicazione tra client e server. A questa è passata come argomento il segnale "SIGURS1"; • **kill()**, solo nel client. E' utilizzata per inviare il segnale "SIGQUIT" passato come argomento al processo figlio che gestisce il menù iniziale, al fine di terminarlo. • **signal()**, sia nel client sia nel server. In entrambi, è utilizzata per assegnare un handler ad ogni segnale. In particolare, nel client associa degli handler ai segnali "SIGQUIT", "SIGINT", "SIGALRM", "SIGUSR1" e "SIGUSR2", mentre nel server associa al segnale "SIGQUIT" l'handler per la terminazione dello stesso server; • **alarm()**, solo nel client. E' utilizzata per la generazione del segnale "SIGALRM" necessario per richiamare l'handler della connessione tra client e server. ❏ pipe e fifo: • **pipe()**, solo nel server. E' utilizzata prima dell'avvio del server per realizzare la pipe destinata per la scrittura e lettura, subito dopo la creazione di una configurazione, di un codice di controllo, identificato dal valore 1, necessario per visualizzare su

	schermo dei messaggi e per inviare al client un flag; • **write()**, solo nel server. E' utilizzata per scrivere sulla pipe il valore 1 rappresentante un codice di controllo; • **read()**, solo nel server. E' utilizzata per leggere dalla pipe il codice di controllo 1 scritto in precedenza; ❏ shared memory: • **shmget()**, solo nel server. E' utilizzata per creare la shared memory e restituirne l'ID; • **shmat()**, solo nel server. E' utilizzata per effettuare l'attach, ossia l'attaccamento, della shared memory allo spazio di indirizzamento del programma; • **shmdt()**, solo nel server. E' utilizzata per effettuare il detach, ossia lo scollegamento, della shared memory dallo spazio di indirizzamento del programma; • **ftok()**, sia nel client sia nel server. In entrambi è utilizzata per costruire la chiave da applicare per il funzionamento della shared memory. ❏ coda di messaggi: • **msgget()**, sia nel client sia nel server. Nel client è utilizzata per creare una coda di messaggi necessaria per la comunicazione con il server in fase di contatto di un operatore per assistenza. Nel server è applicata per generare una coda di messaggi fondamentale per la comunicazione con il client nella medesima fase di prima; • **msgsnd()**, sia nel client sia nel server. Nel client è utilizzata per inviare i messaggi dell'utente al server quando necessita di assistenza. Nel server è applicata per inviare messaggi di risposta al client, ovvero all'utente, in modo da fornirgli il servizio di assistenza richiesto; • **msgrcv()**, sia nel client sia nel server. Nel client è utilizzata per ricevere i messaggi del server, ed in particolare, dell'operatore che sta fornendo assistenza. Nel server è

	applicata per ricevere i messaggi del client, e nello specifico, dell'utente che sta richiedendo assistenza; • **msgctl()**, sia nel client sia nel server. In entrambi è utilizzata per rimuovere la coda di messaggi costruita per il servizio di assistenza offerto dal programma. ❏ gestione dei thread: • **pthread_create()**, sia nel client sia nel server. Nel client è utilizzata per creare un thread per la gestione in fase di creazione della configurazione e due thread in fase di caricamento della stessa, il primo per la ricezione del seriale ed il salvataggio delle scelte delle componenti della moto sull'apposita struttura dati ed il secondo per la stampa su schermo di tali scelte. Nel server è applicata per realizzare un thread per la gestione della funzionalità scelta dal menù iniziale ed un thread per l'invio della configurazione caricata dall'utente allo stesso server; • **pthread_exit()**, sia nel client sia nel server. Nel client è utilizzata per la chiusura di quattro thread: uno per la gestione del menù iniziale, uno per la gestione in fase di creazione della configurazione e due impiegati in fase di caricamento della configurazione per la ricezione del seriale ed il salvataggio delle scelte delle componenti della moto sull'apposita struttura dati e per la stampa su schermo di tali scelte. Nel server è applicata per la terminazione di due thread: il primo per la gestione della funzionalità scelta dal menù iniziale ed il secondo per l'invio della configurazione caricata dall'utente allo stesso server; • **pthread_join()**, sia nel client sia nel server. Nel client è utilizzata per attendere la terminazione del thread impiegato per la gestione del menù iniziale prima dell'avvio di una delle sue funzionalità e di quello avviato per la gestione di queste ultime,

	sia in fase di creazione sia in quella di caricamento della configurazione. Nel server è applicata per attendere la chiusura del thread adoperato per la gestione della funzionalità scelta dal menù iniziale e di quello utilizzato per l'invio della configurazione caricata dall'utente allo stesso server; ❏ semafori System-V per processi: • **semget()**, solo nel server. E' utilizzata per la creazione di un semaforo System-V immediatamente prima che il server riceva il valore corrispondente alla funzionalità scelta dall'utente nel menù iniziale; • **semop()**, solo nel server. E' utilizzata per la gestione e, nello specifico, per il blocco e lo sblocco del semaforo System-V costruito in precedenza; • **semctl()**, solo nel server. E' utilizzata per la gestione e, nello specifico, per l'inizializzazione e la deallocazione del semaforo System-V realizzato precedentemente; ❏ semafori POSIX con nome per thread: • **sem_open()**, solo nel client. E' utilizzata per creare il semaforo POSIX dopo aver impostato la connessione tra client e server; • **sem_wait()**, solo nel client. E' utilizzata per il blocco del thread sul semaforo POSIX precedentemente realizzato dopo ognuna delle opzioni scelte in fase di creazione della configurazione ed anche successivamente all'inserimento del nome della medesima; • **sem_post()**, solo nel client. E' utilizzata per lo sblocco del thread sul semaforo POSIX costruito in precedenza dopo ogni aggiunta del prezzo della componente della moto scelta al prezzo complessivo della stessa e dopo aver realizzato il file relativo a tale configurazione;

	• **sem_close()**, solo nel client. E' utilizzata per la chiusura del semaforo POSIX in fase di chiusura del menù iniziale e del server; • **sem_unlink()**, solo nel client. E' utilizzata per la rimozione del nome e la distruzione del semaforo POSIX in fase di chiusura del menù iniziale e del server. ❏ mutex per thread: • **pthread_mutex_destroy()**, solo nel client. E' utilizzata per rimuovere tutte le risorse allocate per la gestione del Mutex in fase di chiusura del menù iniziale e del server; • **pthread_mutex_lock()**, sia nel client sia nel server. In entrambi è utilizzata per bloccare il Mutex in fase di caricamento della configurazione prima dell'inserimento del relativo seriale; • **pthread_mutex_unlock()**, sia nel client sia nel server. In entrambi è utilizzata per sbloccare il Mutex in fase di caricamento della configurazione dopo l'inserimento del relativo seriale; • **pthread_cond_wait()**, solo nel server. E' utilizzata per attendere la variabile condizione necessaria per il funzionamento del Mutex in fase di caricamento della configurazione prima della ricezione del seriale; • **pthread_cond_init()**, solo nel server. E' utilizzata per inizializzare la variabile condizione necessaria per far funzionare il Mutex all'atto di avvio del server; • **pthread_cond_signal()**, solo nel server. E' utilizzata per sbloccare il thread che si occupa della ricezione del seriale in fase di caricamento della configurazione, il quale è bloccato sulla variabile condizione precedente, necessaria per il funzionamento del Mutex. Lo sblocco avviene dopo la ricezione di tale seriale; • **pthread_cond_wait()**, solo nel server. E' utilizzata per bloccare il thread che si occupa della ricezione del seriale in fase di caricamento della configurazione, il

	quale è sbloccato sulla variabile condizione precedente, necessaria per il funzionamento del Mutex. Il blocco avviene prima della ricezione del seriale; ● **pthread_cond_destroy()**, solo nel server. E' utilizzata per deallocare tutte le risorse indispensabili per la gestione della variabile condizione precedente, necessaria per il funzionamento del Mutex, dopo la ricezione del seriale; ● **pthread_sigmask()**, solo nel client. E' utilizzata per impostare la maschera dei segnali all'atto di avvio del server e per modificare tale maschera in fase di terminazione di ognuna delle funzionalità scelte attraverso il menù iniziale, sbloccando il segnale "SIGQUIT". ❏ segnali tra thread: ● **SIGUSR2**, solo nel client. E' inviato con "pthread_kill()" per gestire il menù iniziale; ● **SIGQUIT**, solo nel client. E' sbloccato dalla maschera con "pthread_sigmask()" e successivamente inviato per la terminazione di ognuna delle funzionalità scelte tramite il menù iniziale.
Funzioni implementate	Funzioni implementate nel **client**: 1. **void *funzioneThread1();** ● **Parametri di ingresso**: nessuno; ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di creazione della configurazione, vale a dire selezionando la prima funzionalità dal menù iniziale. Contiene tutte le possibili scelte per ognuna delle componenti della moto e somma il valore delle opzioni selezionate al costo complessivo del mezzo. Consente anche il salvataggio su file di testo della configurazione appena realizzata. Interagisce con il thread che si occupa del lancio del menù contenente le opzioni per ogni componente della moto; 2. **void *funzioneThread2_1();** ● **Parametri di ingresso**: nessuno; ● **Parametri di uscita**: nessuno.

	Si avvia solamente in fase di caricamento della configurazione, vale a dire selezionando la seconda funzionalità dal menù iniziale. Contiene una richiesta di inserimento dell'opzione scelta per ognuna delle componenti della moto. Interagisce con il thread che si occupa del lancio del menù richiedente le opzioni per ogni componente della moto, ossia il seriale, da inserire cifra per cifra; 3. **void *funzioneThread2_2();** ● **Parametri di ingresso**: nessuno; ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di caricamento della configurazione, ovvero selezionando la seconda funzionalità dal menù iniziale. Riceve la configurazione relativa al seriale inserito dal server e la stampa su schermo nel client. Interagisce con il thread che si occupa di recuperare e far visualizzare la configurazione ottenuta inserendo un determinato seriale; 4. **void hanlerExit(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale "SIGINT", associato a questa funzione attraverso la system call "signal()" all'interno del main. ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di terminazione del client da parte dell'utente attraverso la combinazione di comandi "Ctrl + C" della tastiera. Contiene istruzioni per terminare il client ed individuare eventuali errori in fase di chiusura e disconnessione del semaforo POSIX e di distruzione del semaforo Mutex precedentemente realizzati. Interagisce con il processo che si occupa di avviare ed eseguire i passaggi necessari per il funzionamento del client, del suo menù e delle sue funzionalità, ad esclusione della prima; 5. **void hanlerMenu(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale "SIGUSR2",

	associato a questa funzione attraverso la system call “signal()” all’interno della funzione “*menu()”. ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di instaurazione della connessione tra server e client. Contiene istruzioni per visualizzare la riuscita connessione del client con il server, le proprie porte ed il menù iniziale. Interagisce con la funzione “*menu()” per attendere dapprima il segnale “SIGUSR2”, successivamente l’instaurazione della connessione ed infine l’avvio del menù iniziale; 6. **void hanlerQuit(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale “SIGQUIT”, associato a questa funzione attraverso la system call “signal()” all’interno del main. ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di terminazione di ognuna delle quattro funzionalità scelte attraverso il menù iniziale. Contiene istruzioni per terminare ognuna delle quattro funzionalità scelte dal menù iniziale ed individuare eventuali errori in fase di chiusura e disconnessione del semaforo POSIX e di distruzione del semaforo Mutex precedentemente realizzati. Interagisce con il processo che si occupa di avviare ed eseguire i passaggi necessari per il funzionamento del client, del suo menù e delle sue funzionalità, ad esclusione della prima; 7. **void hanlerConnessione(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale “SIGALRM”, associato a questa funzione attraverso la system call “signal()” all’interno del main. ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di corretta instaurazione della connessione del client al server. Contiene istruzioni per verificare la

	riuscita connessione del client al server, specificando le porte di entrambi, o, altrimenti, per visualizzare su schermo eventuali errori. Interagisce con il processo che si occupa di avviare ed eseguire i passaggi necessari per il funzionamento del client, del suo menù e delle sue funzionalità, ad esclusione della prima; 8. **void handlerSemaforo(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale "SIGUSR1", associato a questa funzione attraverso la system call "signal()" all'interno del main. ● **Parametri di uscita**: nessuno. Si avvia solamente in fase di mancata instaurazione della connessione del client al server, situazione in cui il segnale in questione non è disponibile a causa della connessione precedente da parte di un altro client. Contiene istruzioni per far visualizzare su schermo l'errore di mancata connessione ed il tempo di attesa prima di riprovare a connettersi con il relativo conto alla rovescia. Interagisce con il processo che si occupa di avviare ed eseguire i passaggi necessari per il funzionamento del client, del suo menù e delle sue funzionalità, ad esclusione della prima; 9. **void inizializzatoreIndirizzo(struct sockaddr_in *indirizzo,int porta);** ● **Parametri di ingresso**: 1. ***indirizzo**, puntatore di una struct sockaddr_in, contenente informazioni sulla famiglia del protocollo di rete, sulla porta e sull'indirizzo IP del server, il quale corrisponde, in questo caso, all'indirizzo "127.0.0.1"; 2. **porta**, variabile di tipo int, che rappresenta la porta utilizzata per la comunicazione con il server e coincidente con la porta 5000. ● **Parametri di uscita**: nessuno.

	Si avvia appena viene eseguito il client. Contiene istruzioni per far impostare alcuni parametri relativi all'indirizzo IP ed alla porta. Interagisce con il processo che si occupa di avviare ed eseguire i passaggi necessari per il funzionamento del client, del suo menù e delle sue funzionalità, ad esclusione della prima. Funzioni implementate nel **server**: 1. **int deallocazioneSemaforo(int IDSemaforo);** • **Parametri di ingresso**: 1. **IDSemaforo**, variabile di tipo int, che rappresenta il semaforo System-V creato appena prima dell'avvio di una delle funzionalità scelte dal menù iniziale. • **Parametri di uscita**: 1. **semctl(IDSemaforo,0,IPC_RMID,semaforoDeallocato)**, che restituisce un valore intero ottenuto dalla deallocazione del semaforo System-V effettuata attraverso la system call "semctl()". Riceve l'ID del semaforo System-V ed effettua la deallocazione dello stesso semaforo tramite la system call "semctl()". Interagisce con il processo figlio che si occupa della ricezione dei dati inseriti una volta scelta la prima funzionalità dal menù iniziale; 2. **int inizializzazioneSemaforo(int IDSemaforo);** • **Parametri di ingresso**: 1. **IDSemaforo**, variabile di tipo int, che rappresenta il semaforo System-V creato appena prima dell'avvio di una delle funzionalità scelte dal menù iniziale. • **Parametri di uscita**: 1. **semctl(IDSemaforo,0,SETALL,semaforoInizializzato)**, che restituisce un valore intero ottenuto dall'inizializzazione del semaforo System-V effettuata attraverso la system call "semctl()".

	Riceve l'ID del semaforo System-V, imposta alcuni parametri ed inizializza il semaforo tramite la system call "semctl()". Interagisce con il processo figlio che si occupa della ricezione dei dati inseriti una volta scelta la prima funzionalità dal menù iniziale; 3. **int bloccaSemaforo(int IDSemaforo);** ● **Parametri di ingresso**: 1. **IDSemaforo**, variabile di tipo int, che rappresenta il semaforo System-V creato appena prima dell'avvio di una delle funzionalità scelte dal menù iniziale. ● **Parametri di uscita**: 1. **semop(IDSemaforo,operazioniBlocco,1)**, che restituisce un valore intero ottenuto dal blocco del semaforo System-V effettuata attraverso la system call "semop()". Riceve l'ID del semaforo System-V, imposta alcuni parametri e blocca il semaforo tramite la system call "semop()". Interagisce con il processo figlio che si occupa della ricezione dei dati inseriti una volta scelta la prima funzionalità dal menù iniziale; 4. **int sbloccaSemaforo(int IDSemaforo);** ● **Parametri di ingresso**: 1. **IDSemaforo**, variabile di tipo int, che rappresenta il semaforo System-V creato appena prima dell'avvio di una delle funzionalità scelte dal menù iniziale. ● **Parametri di uscita**: 1. **semop(IDSemaforo,operazioniSblocco,1)**, che restituisce un valore intero ottenuto dallo sblocco del semaforo System-V effettuato attraverso la system call "semop()". Riceve l'ID del semaforo System-V, imposta alcuni parametri e sblocca il semaforo tramite la system call "semop()". Interagisce con il processo figlio che si occupa della ricezione dei dati inseriti una volta scelta la prima funzionalità dal menù iniziale;

	5. **void hanlerExit(int segnale);** ● **Parametri di ingresso**: 1. **segnale**, variabile di tipo int, che rappresenta il segnale "SIGQUIT", associato a questa funzione attraverso la system call "signal()" all'interno del main. ● **Parametri di uscita**: nessuno. Si avvia in fase di terminazione di ognuna delle quattro funzionalità scelte attraverso il menù iniziale e all'atto di chiusura del client da parte dell'utente attraverso la combinazione di comandi "Ctrl + \" della tastiera. Contiene istruzioni per chiudere il socket del client e quello del server, effettuare il detach della memoria condivisa e visualizzare su schermo i messaggi di chiusura del server e di connessione terminata correttamente. Interagisce con il processo che si occupa di impostare la connessione e permettere al server di ricevere richieste da parte di diversi client; 6. **void inizializzatoreIndirizzo(struct sockaddr_in *indirizzo,int porta,long indirizzoIP);** ● **Parametri di ingresso**: 1. ***indirizzo**, puntatore di una struct sockaddr_in, contenente informazioni sulla famiglia del protocollo di rete, sulla porta e sull'indirizzo IP del server, il quale corrisponde, in questo caso, all'indirizzo "127.0.0.1"; 2. **porta**, variabile di tipo int, che rappresenta la porta utilizzata per la comunicazione con il client e coincidente con la porta 5000; 3. **indirizzoIP**, variabile di tipo long, che coincide con il parametro "INADDR_ANY", a sua volta corrispondente all'indirizzo generico "0.0.0.0" da assegnare al parametro di ingresso "*indirizzo", e che rende il server raggiungibile da ogni client; ● **Parametri di uscita**: nessuno.

	Si avvia appena viene eseguito il server. Contiene istruzioni per impostare alcuni parametri relativi all'indirizzo IP ed alla porta. Interagisce con il processo che si occupa di eseguire i passaggi necessari per far funzionare il server e ricevere dati dal client. 7. **void *stampaThread1();** • **Parametri di ingresso**: nessuno; • **Parametri di uscita**: nessuno. Si avvia solamente in fase di stabilimento della connessione tra client e server. Contiene istruzioni per ricevere il seriale, visualizzare le opzioni selezionate per ogni componente della moto, salvarne il nome sulle relative variabili della struttura appositamente creata e sommarne il valore sulla variabile destinata a contenere il prezzo complessivo del mezzo. Tale struttura viene poi copiata sulla memoria condivisa, in modo da essere trasferita al client. Interagisce con il thread che si occupa di ricevere il valore della funzionalità scelta dal client tramite il menù iniziale ed avviarla. 8. **void *stampaThread2();** • **Parametri di ingresso**: nessuno; • **Parametri di uscita**: nessuno. Si avvia solamente in fase di caricamento della configurazione da parte del client, dopo la ricezione dell'opzione selezionata dal medesimo dal menù iniziale. Contiene istruzioni per prelevare la struttura definita in precedenza dalla memoria condivisa, effettuare il detach della memoria ed inviare tale struttura al client. Interagisce con il thread che restituisce la configurazione relativa al seriale del client.