

**Dipartimento di Scienze e Innovazione Tecnologica**

**CORSO DI LAUREA IN INFORMATICA**

**PROGETTO DI: Sistemi Operativi 1**

**Progetto: Traccia A**

ANNO ACCADEMICO 2016/2017

Docente: Maurizio Matteo Munafò Gruppo 4.

Riccardo Pidello

Matricola: 10033872

Carlo Daniele Giavazzi

Matricola: 20013889

**Server:**

**Variabili Globali**:

* MESSAGE\_WELCOME: Stringa di benvenuto che il server invia al Client una volta stabilita la connessione.
* MESSAGE\_MINOR: Stringa inviata al Client nel caso il numero ricevuto sia minore del numero da indovinare.

* MESSAGE\_MAJOR: Stringa inviata al Client nel caso il numero ricevuto sia maggiore del numero da indovinare.

* MESSAGE\_CORRECT: Stringa inviata per avvisare di aver indovinato il numero.

* MESSAGE\_ERROR: Stringa inviata per segnalare di aver ricevuto un numero non valido.
* MESSAGE\_ERROR\_TENTATIVI: Stringa inviata per segnalare al Client di aver esaurito i tentativi per indovinare il numero.
* TENTATIVI­\_MAX: costante per indicare il numero massimo dei tentativi oltre cui il server chiude la connessione con un opportuno messaggio.

**Funzione main di tipo int**:

Variabili:

* random: variabile in cui risiede il numero da indovinare generato casualmente.

* tentativo: contatore dei tentativi.
* valore\_client: variabile in cui risiede il valore inserito dall’utente.
* end: flag per terminare il ciclo di identificazione del valore.
* buffer: stringa dove salvare il valore inviato dal client.
* compare: stringa di appoggio per rilevare eventuali caratteri invalidi.

Esecuzione:

1. Il server si mette in ascolto sulla porta passata come argomento.
2. Tramite la funzione socket stabilisce una connessione con client e salva il descrittore in “Simple\_socket”, esegue un controllo su “Simple\_socket” per capire se la connessione è stabilità, in caso contrario stampa un messaggio di errore, altrimenti invia il messaggio di benvenuto contenuto in MESSAGE\_WELCOME al client.
3. Modifica il seme della funzione random e salva nella variabile random il numero casuale compreso da 1 a 100 da indovinare.
4. Inizia il ciclo while che sarà terminato con l’apposito flag “end”, la prima cosa che fa il ciclo è svuotare tramite la funzione memset le due stringhe “buffer” e “compare”, tramite una read legge il valore mandato dal client e lo salva in buffer dopo di che effettua un controllo per evitare la lettura del comando di invio da tastiera.
5. Il contenuto di buffer è quindi trasformato in intero tramite la funzione atoi e salvato in “valore\_client”, il cui contenuto viene scritto in “compare”, tramite la strcmp le due stringhe “buffer” e “compare” sono confrontate e il risultato inserito in un controllo per evitare di interpretare lettere come numeri, oltre a questo viene controllato che il numero rispetti gli estremi forniti all’utente, il non rispetto di una qualsiasi di queste condizioni termina il ciclo e invia il messaggio di errore contenuto in “MESSAGE\_ERRORE”.
6. Viene controllato l’esito della read e il flag “end” il non superamento del controllo porta alla terminazione del ciclo e alla chiusura della comunicazione, superato ciò viene effettuato un controllo su la quantità di tentativi utilizzati, e se anche questo controllo viene superato si può accedere ai controlli per determinare la correttezza del tentativo o nel caso di fallimento la tipologia di aiuto da inviare.
7. Se il valore inserito non è uguale a quello contenuto in random si passa alle casistiche: se il valore è maggiore di 0 e minore di “random” allora si invia il contenuto di “MESSAGE\_MINOR”, se il numero inserito è maggiore di “random” e minore o uguale a 100 allora si invia il contenuto di “MESSAGE\_MAJOR”, infine viene incrementato il numero di “tentativi”.
8. Il ciclo si ripete fino ad indovinare oppure fino all’utilizzo di tutti i tentativi messi a disposizione.

**Client:**

**Funzione split\_message di tipo int:**

Variabili:

* source: il messaggio ricevuto dal server.

* dest\_keyword: puntatore alla stringa di destinazione della parola chiave.
* dest\_message: puntatore alla stringa di destinazione del messaggio.
* n: dimensione della stringa.
* split: carattere di split.
* terminazione: carattere di terminazione.
* pointer\_read: indice da cui partire a scansionare source.
* split\_counter: contatore che salva l'indice in cui si è effettuato lo split.

Esecuzione:

* 1. Effettua a cascata controlli sui puntatori del messaggio del server, e delle stringhe di destinazione, in caso puntino a NULL ritorna l’opportuno intero,

alla dimensione n della stringa viene sommato l’indice da cui partire a leggere il messaggio del server.

* 1. Inizia il ciclo che continuerà fino a che l’indice da cui leggere il messaggio sarà minore della dimensione n, viene eseguito un controllo sull’indice da leggere, in caso corrisponda al carattere di terminazione viene incrementato il contatore dell’indice da cui leggere e termina.
  2. Si effettua un controllo su “splitted” che avvisa di aver splittato il messaggio del server e si controlla di non incontrare il carattere di split in caso il controllo venga superato l’elemento del messaggio di indice “pointer\_read” viene copiato nella stringa di destinazione all’indice “pointer\_keyword ” +1 e viene incrementato il contatore di split.
  3. Se il prima citato controllo non viene superato si controlla se abbiamo già splittato un messaggio, se non è così mettiamo “splitted” a 1, incrementiamo “split\_counter” e inseriamo il carattere di terminazione in “dest\_keyword”, all’indice “pointer\_read”.
  4. Se non abbiamo ancora splittato il messaggio il carattere nell’indice “pointer\_read”, della stringa source viene copiato in “dest\_message” all’indice “pointer\_redad++”.
  5. Se al passo 3) abbiamo incontrato il carattere di terminazione incrementiamo “pointer\_read” e ritorniamo “split\_counter”.

**Funzione main di tipo int:**

Variabili:

* pointer\_read: indice per leggere il messaggio del server.
* return\_status: utilizzata per registrare la condizione di uscita delle funzioni.

Esecuzione:

* 1. Si interrompe per un periodo variabile sfruttando la funzione sleep (per fermarsi) e la funzione random (per determinare il tempo in modo variabile).
  2. Quando riprenderà comunicherà di voler accedere alle risorse di tipo pentolone attraverso un messaggio e tenterà di accederci in modo esclusivo fra i selvaggi attraverso la funzione pthread\_mutex\_lock sulla variabile selvaggi o aspetterà fino a quando sarà possibile farlo, si comporta in questo modo per evitare di avere più processi selvaggi che restino in attesa nel punto successivo.
  3. Quando sarà riuscito controllerà il numero restanti di istanze della risorsa pentolone confrontando il valore della variabile porzioni e se fosse uguale a 0 allora farà partire il processo cuoco con la funzione sem\_post sul semaforo cuocoInizio e il processo selvaggio si fermerà con la funzione sem\_wait sul semaforo cuocoFine fino a quando il processo cuoco lo farà ripartire.
  4. Superati il punto 2 e 3 tenterà di accedere in modo esclusivo anche con il cuoco alla gestione delle risorse di tipo pentolone con la funzione pthread\_mutex\_lock sulla variabile pentolone per estrare una istanza andando a decrementare il contatore delle porzioni di 1 unità. Subito dopo rilascerà l’accesso esclusivo del pentolone con la funzione pthread\_mutex\_unlock sulla variabile pentolone e rilascerà l’accesso degli altri processi selvaggi con la funzione pthread\_mutex\_unlock sulla variabile selvaggi.
  5. Il processo consuma la risorsa con un tempo varibile (sistema simile al punto 1) e fornisce un messaggio.
  6. Il processo termina il consumo e fornisce un messaggio per tornare al punto 1 per un numero di volte pari al valore di appetito (numero di volte che ogni processo deve consumare una risorsa, questo parametro era stato fornito come argv[1] sempre per mezzo della funzione atoi).
  7. Quando il processo avrà terminato il ciclo precedente allora fornirà un messaggio e comunica che il processo è terminato con la funzione pthread\_exit con argomento NULL.