RELAZIONE PROGETTO SISTEMI OPERATIVI FASE 0

Tale relazione ha lo scopo di illustrare sinteticamente il funzionamento del codice del file *communication.c*, rappresentante la fase 0 del progetto di sistemi operativi dell'anno accademico 2018/19.

In linea di massima il proposito del programma concerne la comunicazione di due terminali della macchina Umps2, definiti nel codice **terminal** e **terminal_2**. Ciò che contraddistingue tale interscambio di informazioni è la sua sequenzialità: prima di poter inviare qualsiasi messaggio il terminale 1 deve necessariamente attenderne un altro dal terminale 0. Questo perché è presente un solo core e siccome la funzione che legge sul terminale rimane in attesa che l'utente scriva qualcosa e in quel momento è operativo il terminale 0, deve per forza aspettare che arrivi l'input del terminale 0. Se vengono scritti più messaggi dal terminale 0, questo invierà solo il primo, ma se in seguito avviene la risposta del secondo terminale, il secondo messaggio scritto dal primo andrà in buffer per essere inviato, e così via fino al progressivo esaurimento di tutti i messaggi.

Iniziando l'analisi del codice, si nota subito la definizione di alcuni registri, le cui funzionalità incidono sulla chiarezza dei segnali necessari alla corretta comunicazione dei due terminali. I primi registri, relativi allo STATUS, informano quando il terminale è pronto o meno per ricevere o inoltrare messaggi, grazie a **ST_READY** e a **ST_BUSY**, e quando l'informazione, nello specifico i singoli caratteri da inviare o ricevere, sono stati trasmessi o al contrario sono arrivati correttamente. I secondi registri, in cui la sigla CMD significa COMMAND, operano nel momento in cui vi è il caricamento dei valori all'interno dei campi di **CMD_TRASMIT** o **CMD_RECV**. Al termine di questa operazione, **CMD_ACK** si occupa di settare di nuovo il terminale nello status di ready, e permettere quindi un ulteriore caricamento di dati.

Proseguendo, è importante notare come la trasmissione dei messaggi avviene attraverso l'utilizzo di due vettori di char, **buf** e **buf_2**, da cui prenderemo appunto le parole che vogliamo inserire nel terminale. Come già anticipato precedentemente, mentre l'utente è intento a scrivere sul primo terminale, il secondo rimane in sospeso in attesa che arrivi il messaggio di quello iniziale. La funzione **readline()** si occupa di leggere l'input di terminal, poi questo input viene caricato su buf_2, e quindi mandato a terminal_2. Viceversa, per far si che il secondo terminale inoltri un qualsiasi messaggio al primo il procedimento si svolge in modo analogo, utilizzando però il primo buffer. Nello specifico, readline() prende carattere per carattere l'input dato al terminale per poi memorizzarlo nel vettore di buffer di competenza. Il valore **count** viene decrementato progressivamente poiché come indicato da **LINE_BUF_SIZE** il vettore può ricevere al massimo 64 meno 1 caratteri, ad ovvia esclusione dell'ultimo char, il "\0".

Successivamente, la funzione **term_getchar()** opera prendendo e restituendo singolarmente tutti i caratteri precedentemente raccolti in input e continua a girare fino a quando non viene interrotta dal comando di invio. Prima di effettuare questi passaggi però, tale funzione si preoccupa di assicurarsi che effettivamente l'utente possa scrivere, chiedendosi grazie ai registri se il terminale è pronto per ricevere dati.

Altre funzioni da evidenziare sono **term_puts()** e **term_putchar(),** adibite alla lettura e scrittura su terminale. term_puts() ha il compito di passare la stringa al terminale designato in quel momento

all'invio di un messaggio. Attraverso un ciclo for controlliamo che il vettore di caratteri sia o meno arrivato al termine della propria capacità. Di conseguenza il programma inoltra ogni carattere all'interno delle celle all'alto metodo, term_putchar().

Compito di quest'ultimo è di verificare prima dell'invio che il terminale sia libero avvalendosi della funzione **tx_status(),** la quale nell'effettivo controlla che ci sia il permesso alla scrittura. L'input del carattere avviene quindi attraverso la funzione **trasm_command**. Mentre tx_status viene richiamata per verificare che ci sia la libertà di scrittura**, rx_status()** permette di conoscere se il terminale è possibilitato a ricevere dati, funzionando analogamente alla funzione precedentemente descritta.