Giacomo Ferro 21/06/2017

RELAZIONE DELL'ELABORATO DI SYSTEM CALL

Struttura generale dell'elaborato:

Ho creati tre file sorgente .c che gestiscono il server, clientSend e clientRecive. Oltre a questi ho creato anche una libreria "mex.h" che definisce la struttura base del messaggio che si può inviare, il valore della chiave della coda e il valore del messaggio di tipo 1. Ora proseguirò descrivendo il funzionamento dei tre file sorgente.

Struttura di server.c:

Ho dichiarato delle variabili globali che serviranno per la gestione della coda e per i vari segnali. Ho anche dichiarato il prototipo di una funzione per interpretare il messaggio di tipo 1 ricevuto. Ora procedo spiegando il funzionamento delle funzioni:

1. **InterruzioneCtrloZ** = questa funzione viene lanciata se viene intercettato dalla signal() un segnale di tipo SIGINT o SIGTSTP. Essa elimina la coda grazie alla funzione msgctl(IDcoda, IPC_RMID, NULL) ed invia la terminazione al figlio grazie alla kill(pidFiglio,SIGKILL).
2. **InterruzioneCtrlCoZFiglio** = se l'interruzione avviene nel figlio esso termina direttamente.
3. **SegnaleUsr1** = funzione che gestisce il segnale SIGUSR1. Essa rimuove la coda e termina il figlio indipendentemente dal numero di messaggi rimasti nella coda.
4. **SegnaleUsr2** = funzione che gestisce il segnale SIGUSR2. Essa rimuove la coda con la msgctl() se e solo se non ci sono messaggi nella coda. Per verificare che la coda non sia vuota salvo le statistiche della coda grazie alla funzione msgctl(msgid,IPC_STAT,&buf) dove buf è una variabile di tipo struct msqid_ds. Se il numero dei messaggi contenuti in buf.msg_qnum è > 0 allora non termino la coda e proseguo leggendo i prossimi messaggi, altrimenti eseguo una msgctl() e una kill() per terminare coda e figlio.
5. **AttivaUsr1** = funzione che si attiva dopo alarm(n) attivata dall'invocazione di una closetime <n>. Essa gestisce il segnale lanciato da SIGALRM e, dopo averlo intercettato, lancia una SIGUSR1 grazie a kill(getppid(), SIGUSR1) al padre.

Successivamente nel main() ho dichiarato un puntatore di tipo messaggio e, dopo aver aperto la coda grazie alla msgget(MSGKEY, <permessi>), verifico che sia stata aperta senza errori. Dopo dichiaro un puntatore al messaggio allocando la memoria necessaria tramite malloc(): se l'allocazione non va a buon fine elimino la coda e termino altrimenti eseguo una serie di signal() che servono per trattare le interruzioni ctrl-c, ctrl-z, SIGUSR1, SIGUSR2. Dopo avere eseguito una fork(), se sono nel figlio eseguo una msgrcv() solo di messaggi di tipo 1 dentro un ciclo while infinito. Se un messaggio di tipo 1 è presente, passo

il puntatore al messaggio alla funzione che lo interpreta per eseguire le operazioni desiderate.

Funzione IntrepretaMex(): Eseguo 3 confronti tramite la funzione strcmp() della libreria string.h che ritorna 0 se le due stringhe passate come parametri sono uguali (altrimenti >0 o <0). Se l'utente ha digitato "niceclose", allora lancio una kill(getppid(), SIGUSR1) al padre per eliminare la coda e terminare il figlio.

Se invece la stringa inserita è "freeall" allora tramite la msgctl(IDcoda, IPC_STAT, &buf) salvo in buf le statistiche sulla coda e così posso accedere al numero dei messaggi della coda (grazie a msg_qnum). Se non ci sono messaggi nella coda, segnalo la cosa e ritorno, altrimenti eseguo una msgrcv() con tipo del messaggio uguale a 0 (ovvero leggo qualsiasi messaggio) aggiornando ogni volta il valore di buf finché la coda non si svuota.

Da ultimo, se la stringa inserita è uguale a "closetime" riconosciuta tramite la funzione strtok(ptr->stringa, " \n") allora dichiaro un puntatore a caratteri che mi serve per salvare il numero di secondi da passare alla alarm(). Al puntatore p assegno l'indirizzo di &(p->stringa[10]) e poi salvo in un intero n il valore intero salvato in p. Setto l'alarm con valore n e poi mando il figlio in attesa della terminazione con pause().

Struttura di clientSend.c:

Definisco una funzione che gestisce i segnali ctrl-c e ctrl-z digitati dall'utente. Questa funzione crea un messaggio "closetime 1" da inviare al server per permettere la terminazione del programma.

Nel main() segnalo la gestione per ctrl-c e ctrl-z e poi alloco la memoria per la creazione del messaggio. Dopo eseguo un controllo sui parametri tramite la funzione controlloParametri(). Se l'apertura della coda va a buon fine, converto in long il tipo di messaggio fornito da riga di comando e salvo nel secondo campo la stringa passata. Con un ciclo while() concateno gli eventuali token presenti come argomenti.

Da ultimo, eseguo la msgsnd(msgid, ptr, sizeof(message), 0) che permette di inviare il messaggio alla coda.

Funzione controlloParametri(): nella stringa ‘str’ concateno tutta la stringa passata da riga di comando per controllare che non superi 256 caratteri. Controllo che non ci siano più di 3 parametri, che il valore della mailbox non sia <1 e quindi che strlen(str) sia < 256. In seguito eseguo dei controlli particolari nel caso di mailbox uguale a 1 grazie alla funzione controlloStringhe(). Quest'ultima funzione ritorna 0 se l'utente ha inserito uno dei tre comandi previsti per mailbox=1. Successivamente controllo che, se l'utente ha inserito "closetime", sia presente un altro parametro che definisca il numero dei secondi (e che sia affettivamente un numero valido cioè >0) ed in caso di errore ritorno 0.

Da ultimo controllo che nel caso di "freeall" o "niceclose" l'utente non abbia fornito troppi parametri non richiesti.

Se il messaggio supera tutti questi controlli, la funzione ritornerà 1.

Struttura di clientRecive.c:

Anche questo gestisce ctrl-c o ctrl-z in maniera uguale a clientSend. Nel main() eseguo un controllo dei parametri tramite la funzione controlloParametri() che controlla se il numero di parametri non è eccessivo e se il numero della mailbox è >=2. In caso di errore ritorna 0 altrimenti 1. Se i controlli sono stati superati con successo, il programma alloca dinamicamente la memoria per il messaggio, apre la coda controllando che non ci siano errori e, convertendo a long l'mtype del messaggio fornito da riga di comando, esegue una msgrcv() fornendo il tipo specificato dall'utente e stampando il messaggio a video.