Proiect RemMux

Radu-Florin Marinoiu

Facultatea de Informatic, Universitatea Alexandru Ioan Cuza
radumarinoiu98@gmail.com
http://students.info.uaic.ro/ radu.marinoiu

1 Introducere

1.1 Proiectul ales

Proiectul ales de mine este RemMux.

Cerinta: Implementati un client/server(concurent) ce permite executia de comenzi la distanta - similar cu SecureSHell, insa fara cerinte de encriptie.
Clientul va rula intr-un terminal, si va putea deschide ferestre multiple, afisate vizual simultan - similar cu Tmux.
Sugestie: folositi bibliotecile curses/ncurses.

1.2 Motivul alegerii

Am ales acest proiect deoarece mi s-a parut interesanta libraria ncurses si realizarea unei aplicatii cu ferestre intr-o consola.

1.3 Functionalitatea aplicatiei

Aplicatia client va rula intr-un terminal in care putem deschide mai multe ferestre, fiecare continand un shell.
La creerea unei ferestre, aceasta deschide o conexiune cu server-ul si face pipe la input prin socket-ul spre server.
Serverul isi creeaza cate un copil pentru fiecare client conectat caruia ii paseaza toate datele necesare comunicarii cu clientul. Acesta executa comenzile primite de la client intr-un shell propriu dupa care trimite output-ul comenzii inapoi la client.
Clientul afiseaza output-ul primit in terminalul deschis.

2 Tehnologii utilizate

Am folosit NCurses pentru a crea interfata in consola. Pentru backend am folosit socket-uri TCP, pipe-uri si subprocese (fork).

3 Arhitectura aplicatiei

3.1 Serverul

Serverul odata pornit va asculta pe un anumit port, asteptand conexiuni de la clienti. In momentul in care un client se conecteaza la server, acesta creeaza un copil dedicat acelui client si apoi revine la a astepta conexiuni.

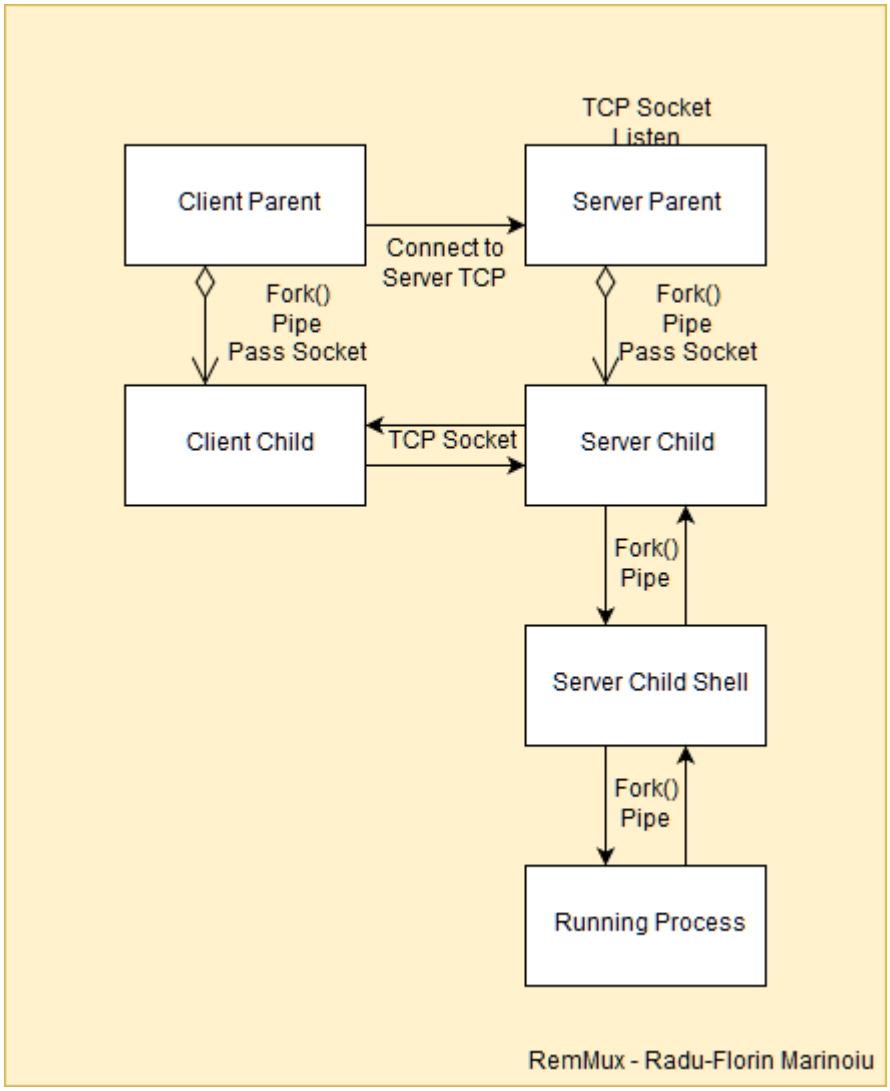
3.2 Shell-ul

Cand un client se conecteaza la server, acestia isi creeaza copii care continua comunicarea, iar copilul serverului creeaza un shell spre care redirecteaza tot input-ul de la client si tot output-ul spre client. Acest shell ia input-ul, il parseaza si apoi executa comenzile cand este posibil, ca apoi sa returneze output-ul acestora.

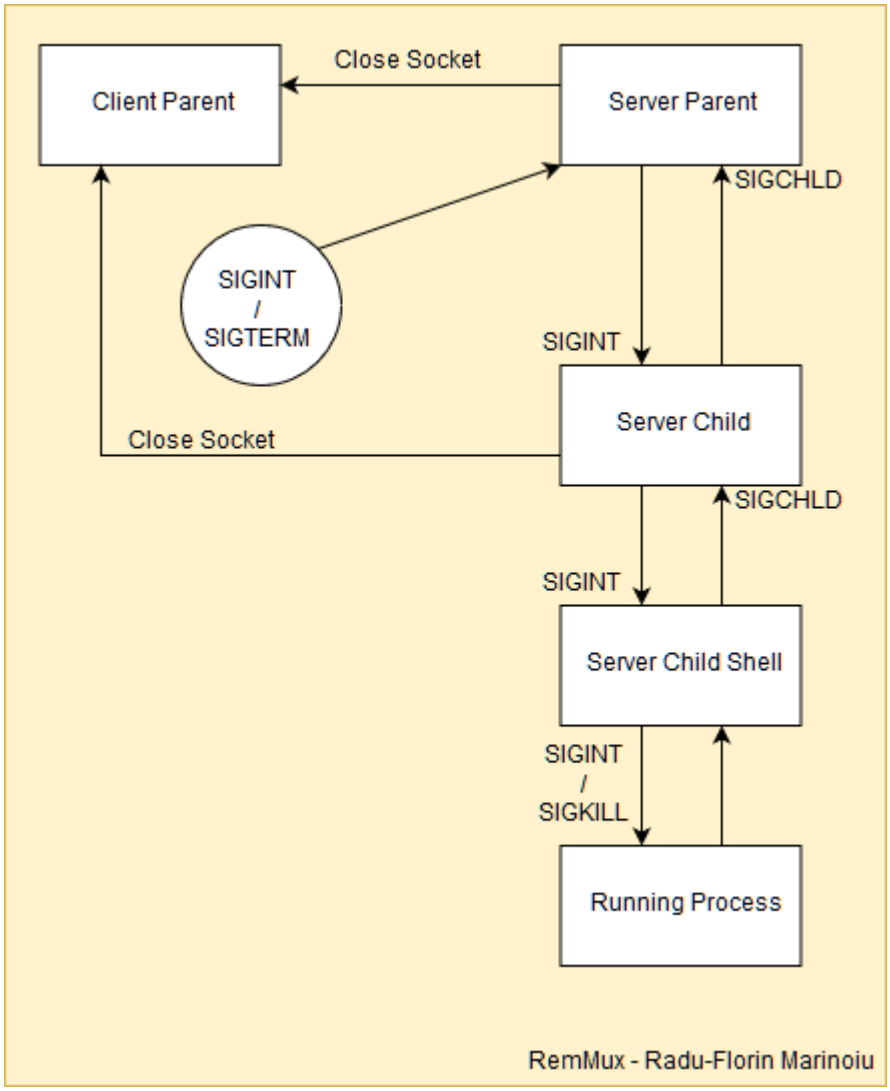
3.3 Clientul

Clientul isi creeaza un copil ce se conecteaza la server si pastreaza un socket tcp prin care sa comunice cu serverul. Copilul citeste tot ce primeste de la parinte printr-un pipe si trimite orin socket-ul deschis catre copilul serverului totul. In acelasi timp acesta citeste tot ce trimite serverul ca si raspuns, adica output-ul comenzilor, pe care il pune in alt pipe ce duce la Client, ca acesta sa afiseze pe consola potrivita. Pot rula mai multi clienti simultan.

3.4 Diagrama de comunicare



3.5 Diagrama ordinii de shutdown



4 Detalii de implementare

Serverul, conform ierarhiei de mai sus va avea grija de copiii sai prin intermediul unui SIGCHLD cand acestia se inchid. Acest handler se propaga si este folosit si de catre nepoti, etc. ai procesului Server, deci daca oricare din ei primeste comanda de shutdown, va trimite SIGINT la copiii lui si apoi va astepta ca acestia sa se inchida, dupa care se va inchide dand un semnal parintelui sau, pentru ca acesta sa se ocupe de el.

Pe client acest lucru nu este necesar intrucat clientul este constituit dintr-un singur proces.

Pentru a rezolva problema clientului deconectat care a parasit sesiunea fara sa ii comunice acest lucru serverului, am eliminat caile amiabile de shutdown si am lasat doar un timeout pe server, care va inchide conexiunea daca clientul nu raspunde in timp util.
In acelasi timp, daca serverul se inchide toti clientii se vor inchide la urmatorul refresh.

Interfata text-grafica va fi implementata cu ajutorul librariei ncurses. In momentul actual, fiecare fereastra fiind pozitionata dupa un algoritm simplu:

> Cand se umple spatiul de NxM console (Matrice), se mai adauga un rand/coloana in mod alternativ (coloana→rand→coloana→...)

Exemplu algoritm:

```
void Client::resize_event() {
    int CONSOLE_H, CONSOLE_W;
    getmaxyx(stdscr, CONSOLE_H, CONSOLE_W);
    int window_count = children.size();
    int rows = 0, cols = 0;
    int i = 0, j = 0;
    WINDOW_DESC w_desc;

    while(rows*cols < window_count)
        if(cols > rows)
            rows++;
        else
            cols++;

    erase();
    for(Child &child: children){
        if(j >= cols || i*cols + j >= window_count){
            i++; j = 0;
        }
        if(i >= rows)
            break;
        w_desc.height = (CONSOLE_H - 1) / rows;
        w_desc.width = CONSOLE_W / cols;
        w_desc.y = (CONSOLE_H - 1) / rows * i;
        w_desc.x = CONSOLE_W / cols * j;
        child.Set_Pos_Size(w_desc);
        child.Resize_Event();
        j++;
    }
}
```

5 Concluzie

Proiectul a fost putin mai dificil decat ma asteptam, intrucat s-a dovedit time-consuming sa creezi un shell de la 0, lucru care mi-a dat batai de cap. Din acest motiv shell-ul este atat de minimal, intrucat nu poate decat sa execute comenzi simple si sa returneze output-ul acestora, fara pipe-uri, fara change directory, fara redirect. Cu ceva timp suplimentar as fi capabil sa introduc aceste functionalitati daca ar fi necesare.

References

1. https://linux.die.net/man/3/ncurses
2. https://stackoverflow.com/
3. https://www.linuxquestions.org
4. https://www.geeksforgeeks.org/making-linux-shell-c/
5. Alte surse/tutoriale de ncurses/tcp/shell.