UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO
Sistemas Operacionais
Prof. Herbert Rocha

GEORGE LUCAS MONÇÃO ZAMBONIN

RELATÓRIO PROJETO FINAL
SISTEMAS OPERACIONAIS

Boa Vista - RR
2022

OBJETIVO

Quando uma aplicação (por exemplo, shell) é usada remotamente, geralmente não é suficiente simplesmente substituir E/S de dispositivo local (por exemplo, entre o terminal do usuário e o shell) com a E/S de rede. Sessões remotas e protocolos de rede adicionaram opções e comportamentos que não existiam quando o aplicativo estava sendo usado localmente. Para manipular esse processamento adicional sem fazer nenhuma alteração no aplicativo, é comum criar agentes do lado do cliente e do lado do servidor que manipulem a comunicação de rede e protejam o aplicativo das complexidades dos protocolos de acesso remoto. Esses agentes intermediários podem implementar recursos valiosos (por exemplo, criptografar o tráfego para aumentar a segurança ou compactar o tráfego para melhorar o desempenho e reduzir o custo da comunicação em escala de WAN), de forma totalmente transparente para o usuário e o aplicativo.

Neste projeto, será criado um cliente e um servidor de telnet que pode ser dividido em três etapas principais:

- Passar a entrada e saída através de um soquete TCP;
- Execução de comandos shell via pipe, fork, dup2 e execv.
- Compactar a comunicação entre o cliente e o servidor.

DESENVOLVIMENTO

SOCKET TCP

O socket tcp foi desenvolvido utilizando a biblioteca <sys/socket.h> da linguagem C. Para o estabelecimento e funcionamento da comunicação entre máquinas esta biblioteca fornece as seguintes funções chaves:

bind() - Quando um socket é criado com socket, ele existe em um namespace (adress family), mas não possui endereço assinado a ele. Bind() declara um endereço específico por addr para o socket referenciado por um file descriptor sockfd. Addrlen especifica o tamanho, em bytes, de uma estrutura de endereço apontada por um addr. Tradicionalmente esta operação é chamada de “declaração de nome a um socket”.

socket() - cria um endpoint para comunicação e retorna um file descriptor que se refere a esse endpoint. O file descriptor retornado por uma chamada bem-sucedida será o file descriptor de número mais baixo não aberto no momento para o processo. O argumento de domínio especifica um domínio de comunicação; isso seleciona a família de protocolos que será usada para comunicação. Essas famílias são definidas em <sys/socket.h>.

accept() - é usada com socket baseado em conexão tipos (SOCK_STREAM, SOCK_SEQPACKET). Ele extrai o primeiro solicitação de conexão na fila de conexões pendentes para o socket de escuta, sockfd, cria um novo socket conectado e retorna um novo file descriptor referente a esse socket. o socket recém-criado não está no estado de escuta. O original socket sockfd não é afetado por esta chamada.

listen() - marca o socket referido por sockfd como um socket passivo, ou seja, como um socket que será usado para aceitar requisições de conexão de entrada usando accept().

Para estabelecer uma conexão tcp entre o server.c e o client.c primeiramente o programas deverão inicializar um socket utilizando o socket(). Após a inicialização do socket, utilizando a função bind(), um endereço será declarado a estes sockets. Então com o listen(), o server.c aguarda uma contato de um client.c, após o contato ser realizado, ambos programas estabelecem uma conexão utilizando o accept().

Quando a conexão é estabelecida ambos programos utilizam a função *func()*, que é um loop for sem condição de parada contento recursos para

realizar a comunicação cliente servidor, permitindo a transferência de dados por ambos lados com o read() e write() utilizados em um buffer de tamanho 256.

EXECUÇÃO DE COMANDOS SHELL

Após a comunicação de transferência de comandos shell do client.c para o server.c, o server.c primeiro trata os pipes presentes no comando, separando-os em comandos distintos por meio de uma estrutura que foi denominada **pipesc**. Está estrutura possui dois atributos, *commands, que é responsável por armazenar os comandos separados, e o length, que é guarda o tamanho alocado pelo atributo *commands.

Para executar estes comandos em shell, a linguagem de programação C disponibiliza uma função que possibilita a passagem de argumentos e execução de um programa, chamada de **execv()**.

A utilização da função **execv** pode gerar um problema crítico no programa que a tem, pois, após uma chamada bem sucedida, o processo do programa é finalizado. Com isso, existem alguns métodos para contornar este problema, como a criação de um segundo processo para execução desta função.

Uma função que permite a criação de um processo para execução do execv é o **fork()**, que cria um processo filho como "cópia" do processo atual. Então, utilizando um if simples, o programa pode redirecionar o processo pai e filho as suas respectivas seções por meio das ids de seus processos.

Outros problemas podem ocorrer com a utilização do **execv**, como conseguir o output da execução do comando shell, já que o processo que o executa é destruído. Para isso, podemos utilizar um conjunto de recursos disponibilizados pela linguagem C, são eles: **pipe, dup2, stdin, stdout e stderr**. Com a junção destes recursos, uma interação bem distinta pode ocorrer. O **pipe** cria uma passagem com uma entrada de leitura e outra de escrita, com o **dup2** podemos duplicar e declarar outras entradas a esta passagem, como o **stdout**, que é responsável de carregar a sequência de dados de saída do programa, que por fim podemos utilizar para ler a saída do comando **execv**.

CONCLUSÃO

A desistência de um integrante do projeto acabou desistindo do curso, o que acarretou em uma série de atrasos e sobrecarga do outro integrante no desenvolvimento do projeto. Que seguiu em frente com pura força de vontade na reta final da entrega do projeto (Fica aqui uma menção honrosa pro Lucas Daniel, que desistiu do curso por conta da sobrecarga das disciplinas nos últimos momentos do semestre; risos).

A conexão tcp entre o server.c e client.c nos testes foi estabelecida com sucesso e houve a troca de dados por ambos os lados. Onde o client.c passa comandos shell para o server.c, que então executa estes comandos utilizando chamadas de sistema, então o server.c captura a saída destes comandos utilizando pipes, manipulando os fluxos de entrada e saída do programa.

Até o presente problema alguns problemas persistem, como a leitura de somente 8 bytes da saída do execv e a incapacidade de injetar essa saída no stdin, e a falta da implementação da compressão de dados na comunicação tcp.

Com os problemas enfrentados durante o processo de desenvolvimento do projeto, houve dificuldade na entrega do projeto com a proposta completa. Todavia, o projeto foi finalizado com diversas principais funcionalidades propostas inicialmente.

https://github.com/glucard/GeorgeZamboninLucasDaniel_FinalProject_OS_RR_2022