Prevenção de Fork Bombs

PARTICIPANTES:

KEVIN WALLACY.

GIOVANNE DOS SANTOS.

Introdução

- Neste projeto buscamos implementar um fork bomb e também sua respectiva prevenção.
- Pensamos em 2 caminhos de prevenção (Limitar o maximo de processos criados por um usuário e matar o processo do bomb), que serão melhores descritos nos proximos slides.

Primeiro caminho: Matar o processo que causa o fork

- Identificar os processos criados.
- Buscar um grupo de processos de um usuário com vários filhos.
- Eliminar esse processo irergular.

Problemas da 1° abordagem

- Não conseguir executar durante o bomb.
- Desligar o computador.

Segundo caminho: Manipular o limite de processos do usuário.

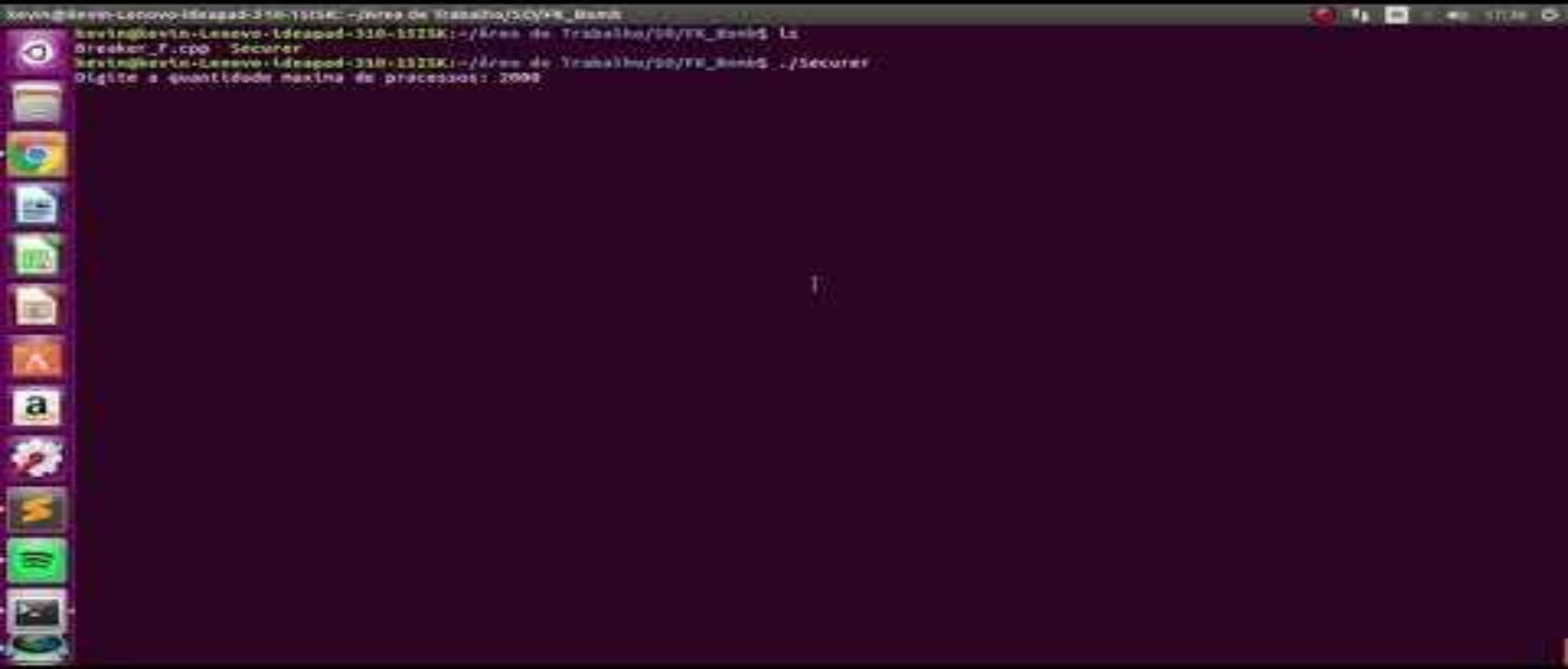
- Utilizar funções para acessar e modificar o limite de processos abertos.
- Executar o bomb simultâneamente e verificar se houve erro ao criar um novo processo.

Breaker_F.cpp

```cpp
#include <unistd.h>

int main(int argc, char const *argv[])
{

    int limit;

    //std::cout<<argc;

    if(argc < 2) // Caso o usuário não tenha fornecido a quantidade de processos maxima como parâmetro de execução.
    {
        std::cout << "Digite a quantidade maxima de processos: ";
        std::cin >> limit;
    }
    else
    {
        limit = std::atoi(argv[1]);
    }

    struct rlimit limite_mod; // Criamos a struct que vai guardar o limite superior e inferior de processos.

    getrlimit(RLIMIT_NPROC, &limite_mod); // Usamos a função getrlimit para receber o limite soft e hard de processos atuais.

    limite_mod.rlim_cur = limit; // Modificamos o limite soft de processos na nossa struct.

    setrlimit(RLIMIT_NPROC, &limite_mod); // Moficamos o limite de processos no sistema, utilizando a struct modificada

    pid_t pid = 0; // Variável responsável por guardar retorno da fork para saber se a modificação foi efetiva e o fork impedido.

    while (true)
    {

        pid = fork(); // Tentamos executar o fork bomb e guardamos o resultado da operação em pid.

        if (pid == -1) // Caso haja um erro ao tentar criar o fork.
        {
            if (errno == EAGAIN) // Se houve um erro, e este erro foi por exceder o limite de processos, a prevenção do fork foi um sucesso.
            {
                std::cout << "Limite de processos execido, pressione Ctrl + C para finalizar a execução." << std::endl;
                return 0;
            }
        }
```

Breaker_F.cpp

```cpp
        std::cin >> limit;
    }
    else
    {
        limit = std::atoi(argv[1]);
    }

    struct rlimit limite_mod; // Criamos a struct que vai guardar o limite superior e inferior de processos.

    getrlimit(RLIMIT_NPROC, &limite_mod); // Usamos a função getrlimit para receber o limite soft e hard de processos atuais.

    limite_mod.rlim_cur = limit; // Modificamos o limite soft de processos na nossa struct.

    setrlimit(RLIMIT_NPROC, &limite_mod); // Moficamos o limite de processos no sistema, utilizando a struct modificada

    pid_t pid = 0; // Variável responsável por guardar retorno da fork para saber se a modificação foi efetiva e o fork impedido.

    while (true)
    {

        pid = fork(); // Tentamos executar o fork bomb e guardamos o resultado da operação em pid.

        if (pid == -1) // Caso haja um erro ao tentar criar o fork.
        {
            if (errno == EAGAIN) // Se houve um erro, e este erro foi por exceder o limite de processos, a prevenção do fork foi um sucesso.
            {
                std::cout << "Limite de processos execido, pressione Ctrl + C para finalizar a execução." << std::endl;
                return 0;
            }
        }

        // Caso um processo pai ou filho tenha sido criado, damos um wait para que o print, caso ocorra, fique bem visível
        if (pid >=0 )
        {
            wait(NULL);
        }
    }

    return 0;
}
```


kevin@kevin-Lenovo-Ideapad-310-15ISK:~/área de Trabalho/SO/FK_Kevin$ ls
Breaker_F.cpp Securer
kevin@kevin-Lenovo-Ideapad-310-15ISK:~/área de Trabalho/SO/FK_Kevin$ ./Securer
Digite a quantidade maxima de processos: 2000