Universidade Estadual de Campinas

EA872 - Laboratório de Programação de Software Básico

Atividade 9

João Marcos Rodrigues RA: 136253 TURMA K

Exercício 3.1

Para esta atividade continuou-se o desenvolvimento do servidor visto nas atividades anteriores. Nesta, em específico, foi implementado o controle de acesso paralelo ao servidor através da criação de processos filhos para executar cada uma das requisições. A fim de não sobrecarregar o servidor, contanto, foi especificado que o programa deveria receber como parâmetro um número N de quantidade máxima de processos filhos simultâneos. Com esse intuito ainda foi implementado, através do método select(), uma alternativa a prevenir o bloqueio de execução do servidor quando não há mais conexões a partir do cliente. O código encontrado em anexo segue os seguintes tópicos:

- O programa principal se não receber nenhuma conexão nos últimos 15 segundos ele será terminado a fim de poupar processamento do computador
- Um processo filho, que receber uma requisição, ao respondê-la esperará até 10s para ver se ocorre uma reconexão caso a mesma não explicitar um Connection: close. Passado os 10s o processo filho encerra a conexão e termina o seu processamento.
- O controle da contagem do número de processos filhos atuais é feito através da atualização de uma variável no processo principal dentro de um método de tratamento de sinais SIGCHLD
- Caso o servidor atinja o limite de processos simultâneos, qualquer conexão que tentar se conectar a ele durante este período terá como resposta uma página Service Unavailable de código 503

Utilizando N = 2 obtivemos os seguintes resultados mostrados a seguir executando requisições simultâneas de 3 navegadores diferentes. Foram utilizados o Safari, Chrome e Firefox.

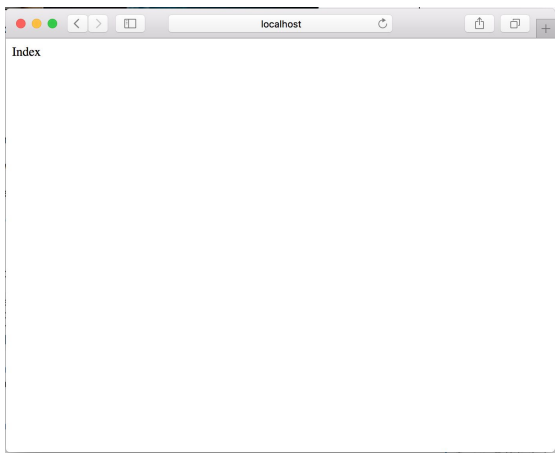

Figura 1: Requisição da raiz de localhost pelo Safari

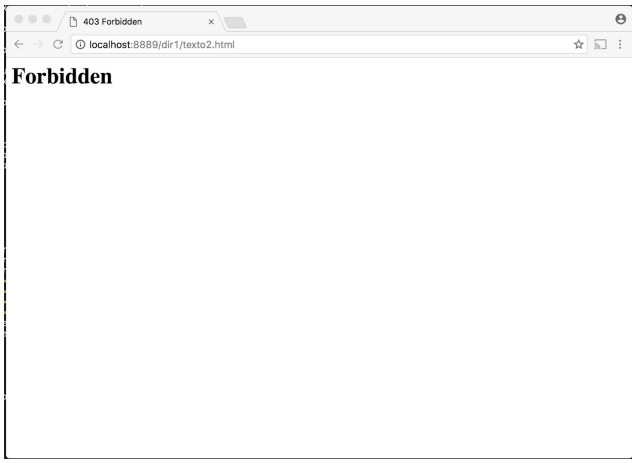

Figura 2: Requisição de texto2.html pelo Chrome

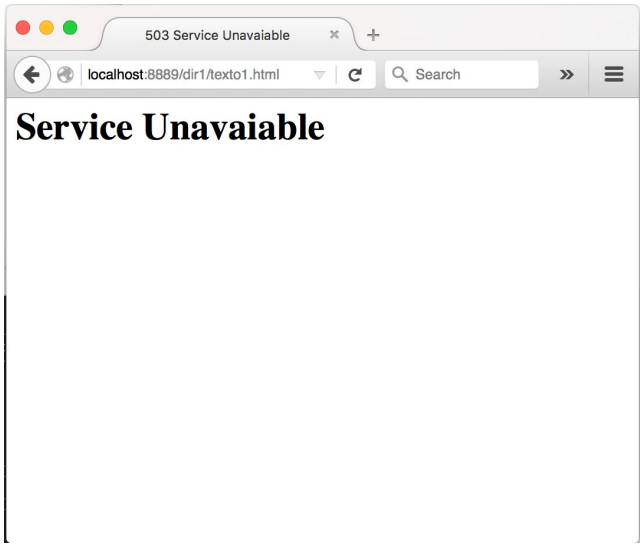

Figura 3: Requisição de texto1.html pelo Firefox. Devido a ser uma requisição a mais do que o servidor suportaria recebeu o erro de serviço indisponível

Aumentando N para 5 seria possível acessar os recursos do servidor pelos 3 navegadores simultâneamente. Feito isso pelos navegadores Safari e Chrome o resultado foram os mesmos obtidos anteriormente demonstrados nas figuras 1 e 2 respectivamente. Entretanto, para o navegador Firefox, o resultado foi diferente. Agora foi possível acessar o recurso mostrado na figura a seguir:

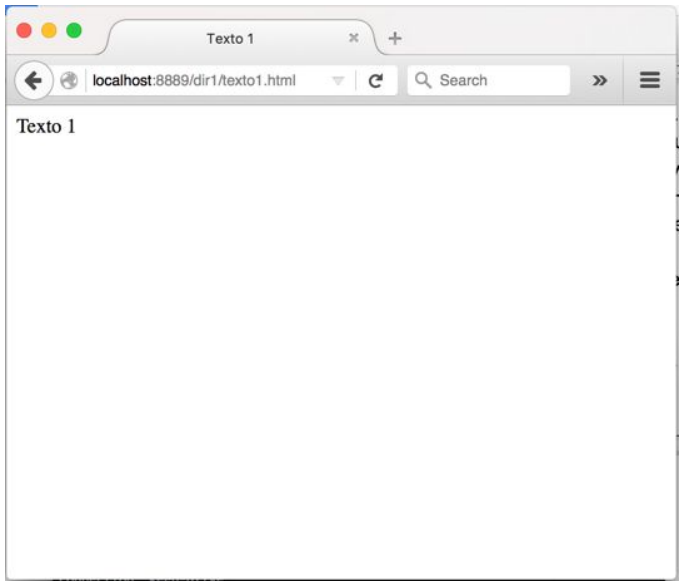

Figura 4: Nova requisição de texto1.html pelo Firefox. Devido ao aumento de N foi possível acessar o recurso

Exercício 3.2

c) Primeiramente temos que a mensagem "Processo xxxx terminou!" não apareceu para o primeiro filho pois o mesmo, eu seu bloco de execução após o primeiro fork(), executa o método exit(20) fazendo com que ele encerre a si mesmo e termine sua execução com um código 20. Caso houvesse a inexistência deste comando o primeiro filho exibiria a mensagem em questão e terminaria seu processo com código 30.

De fato, devido a presença do método wait no código do processo pai evitou que os processos filhos permanecessem no estado terminal. Isso ocorre pois o processo pai espera o término da execução dos processos filhos para continuar a execução de seu código. Assim, o pai sempre terminará após os processos filhos fazendo com que os mesmo sejam inaptos a estarem em estado terminal.

d) Após a realização de alguns testes temos as seguintes três execuções do programa mostrada na figura 5.

Durante a primeira execução não foi forçado nenhum sinal SIGINT no programa de modo que ele exibiu somente as mensagens dos trechos.

Durante a segunda execução forçamos o sinal SIGINT em 3 trechos. No primeiro trecho que, devido a sua configuração, o ignora e nada ocorre. No segundo trecho é definido uma função para tratamento da interrupção deste sinal definido como ger_nova e portanto a mensagem é impressa. No terceiro trecho é definido que o sinal SIGINT deverá ter o seu tratamento padrão (ou seja término da execução do programa) e portanto encerra a execução mas não antes de definir a variável ger_velha que recebe o antigo tratamento deste sinal (ou seja, ger_nova).

Este último detalhe é de grande importância na terceira e última execução pois novamente foram forçados o sinal SIGINT nos dois primeiros trechos, não foi forçado no terceiro, a fim de não encerrar o programa, e encerrou-se forçando o sinal SIGINT no quarto trecho. Conforme já explicado a variável ger_velha recebeu o antigo tratamento de SIGINT e portanto quando ocorreu este sinal ele exibiu a mensagem conforme ocorreu no segundo trecho.

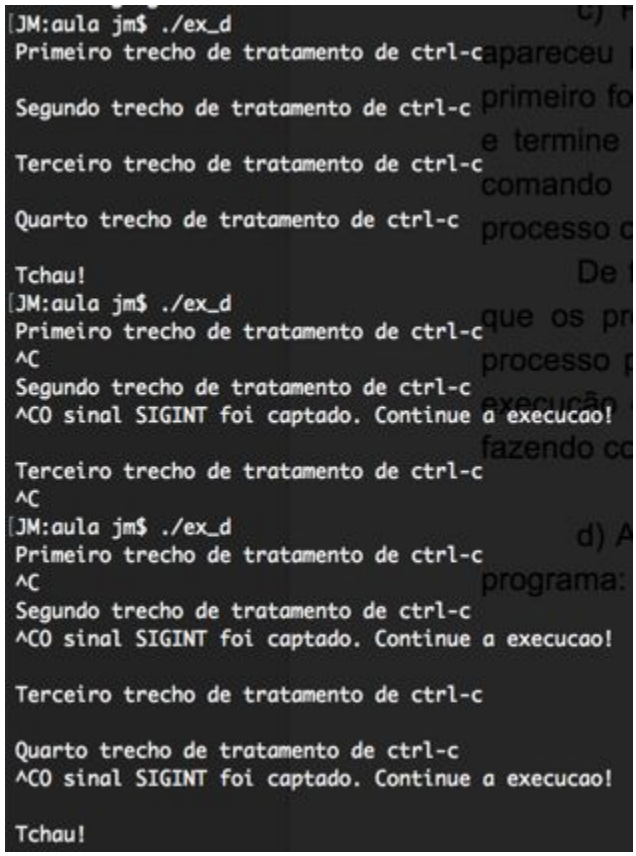

Figura 5: Três execuções do programa ex_d