MC823 - Laboratório de Teleprocessamento e Redes

2º Semestre de 2011

Tarefa 01
Servidor de Eco Simples em TCP

Felipe Augusto da Silva RA 096993
Jesse de Moura Tavano Moretto RA 081704

Introdução

Implementação de um cliente (client_echo.c) e de um servidor concorrente (server_echo.c) de echo simples usando protocolo TCP. O cliente lê linha por linha da entrada padrão e as envia para o servidor, que retorna as linhas recebidas e então o cliente as imprime na saída padrão. Ao final da execução, o cliente envia para a saída de erro as seguintes estatísticas: número de linhas enviadas, tamanho da maior linha, número de caracteres enviados, número de linhas recebidas, número de caracteres recebidos e a contagem do tempo total utilizado, com precisão de 0,01s). O servidor exibe apenas o número total de leituras e de caracteres recebidos e ecoados após o fechamento da conexão.

Estatísticas

Para análise estatística do projeto implementado, foram utilizados:

- **Cliente:** Máquina xaveco do Instituto de Computação (xaveco.lab.ic.unicamp.br), acessado via SSH;

- **Servidor:** Máquina pessoal do aluno Felipe, localizada na cidade de Pedreira-SP, aproximadamente 45 km do IC;

- **Arquivos:**
    - /etc/services, da máquina xaveco (10931 linhas e 651949 caracteres *);
    - bigfile.txt, http://www.ic.unicamp.br/~celio/mc823-2011/exemplos/bigfile.txt (10367 linhas e 394433 caracteres *).

** Informações obtidas com o comando wc*

Estimativa de tempo

A estimativa de tempo foi calculada multiplicando o número de linhas do arquivo pelo tempo médio (19.427ms) retornado pelo comando ping (realizado da máquina xaveco para o computador servidor), obtendo-se:

/etc/services - (10931 linhas) x (19.427ms) = 212.36s

bigfile.txt - (10367 linhas) x (19.427ms) = 201.40s

Dados obtidos

Foram feitos cinco envios de cada arquivo para obtenção de uma média do tempo utilizado. Para que se pudesse ignorar o tempo do sistema de arquivos, cada arquivo foi cacheado em memória antes do uso e a saída do cliente foi direcionada para /dev/null.

- /etc/services

Os dados obtidos para o arquivo /etc/services foram os seguintes:

Linhas enviadas: 10931
Tamanho da maior linha: 131
Caracteres enviados: 651949
Linhas recebidas: 10931
Caracteres recebidos: 651949

Com tempos 284.09, 280.52, 197.47, 192.54, e 232.70 segundos, resultando em uma média de 237.464 segundos, ou seja, aprox. 11,82% acima do tempo estimado.

- bigfile.txt

Os dados obtidos para o arquivo bigfile.txt foram os seguintes:

Linhas enviadas: 10367
Tamanho da maior linha: 195
Caracteres enviados: 394433
Linhas recebidas: 10367
Caracteres recebidos: 394433

Com tempos 182.94, 223.23, 221.83, 272.15, 225.04 segundos, resultando em uma média de 225.038 segundos, ou seja, aprox. 11,74% acima do tempo estimado.

```c
/*
 * server_echo.c - Servidor concorrente de echo simples em TCP
 *
 * MC823 - Tarefa 01
 * Felipe Augusto da Silva        RA 096993
 * Jesse de Moura Tavano Moretto  RA 081704
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <unistd.h>
#include <arpa/inet.h>

#define MYPORT 3490        /* the port users will be connecting to */
#define BACKLOG 10         /* how many pending connections queue will hold */
#define MAXDATASIZE 1000   /* max number of bytes we can get at once */

int main()
{
    int sockfd, new_fd;              /* listen on sock_fd, new connection on new_fd */
    struct sockaddr_in my_addr;      /* my address information */
    struct sockaddr_in their_addr;   /* connector's address information */
    unsigned int sin_size;
    int numBytes, totalBytes, recLines;
    char buffer[MAXDATASIZE];
    int optval = 1;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    /* lose the pesky "address already in use" error message */
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    my_addr.sin_family = AF_INET;          /* host byte order */
    my_addr.sin_port = htons(MYPORT);      /* short, network byte order */
    my_addr.sin_addr.s_addr = INADDR_ANY;  /* automatically fill with my IP */
    bzero(&(my_addr.sin_zero), 8);         /* zero the rest of the struct */

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    while(1) {  /* main accept() loop */
        sin_size = sizeof(struct sockaddr_in);

        if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1) {
            perror("accept");
            continue;
        }
```

```c
        printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));

        if(!fork()) {

            recLines = totalBytes = 0;

            while ((numBytes = recv(new_fd, buffer, MAXDATASIZE, 0)) > 0) {

                if (send(new_fd, buffer, numBytes, 0) == -1) {
                    perror("send");
                    exit(1);
                }

                recLines += 1;
                totalBytes += numBytes;
            }

            fprintf(stderr, "Total de leituras:   %d\n", recLines);
            fprintf(stderr, "Total de caracteres: %d\n", totalBytes);

            close(new_fd);
            exit(0);
        }

        close(new_fd);

        while(waitpid(-1,NULL,WNOHANG) > 0);   /* clean up all child processes */
    }

    return 0;
}
```

```c
/*
 * client_echo.c - Cliente de echo simples em TCP
 *
 * MC823 - Tarefa 01
 * Felipe Augusto da Silva        RA 096993
 * Jesse de Moura Tavano Moretto  RA 081704
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <sys/times.h>
#include <time.h>

#define PORT 3490          /* the port client will be connecting to */
#define MAXDATASIZE 1000   /* max number of bytes we can get at once */


int main(int argc, char *argv[])
{
    int sockfd;
    int sentLines, recLines, sentBytes, recBytes, longestLine, lineSize, numBytes;
    struct hostent *he;
    struct sockaddr_in their_addr;   /* connector's address information */
    clock_t startTime, endTime;
    float elapsedTime;
    char *buffer = (char*)malloc(MAXDATASIZE*sizeof(char));



    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  /* get the host info */
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;         /* host byte order */
    their_addr.sin_port = htons(PORT);       /* short, network byte order */
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    bzero(&(their_addr.sin_zero), 8);        /* zero the rest of the struct */

    if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }

    sentLines = recLines = sentBytes = recBytes = longestLine = lineSize = 0;

    startTime = times(NULL);   /* start time counting */

    while((buffer = fgets(buffer, MAXDATASIZE, stdin)) != NULL) {
```

```
        lineSize = strlen(buffer);
        sentLines += 1;
        sentBytes += lineSize;
        if(lineSize > longestLine)
            longestLine = lineSize;

        if ((send(sockfd, buffer, strlen(buffer), 0)) == -1) {
            perror("send");
            exit(1);
        }

        if ((numBytes = recv(sockfd, buffer, MAXDATASIZE, 0)) == -1) {
            perror("recv");
            exit(1);
        }

        buffer[numBytes] = '\0';
        recLines += 1;
        recBytes += numBytes;

        fputs(buffer, stdout);
    }

    endTime = times(NULL);   /* stop time counting */
    elapsedTime = (float)((endTime - startTime) / (float)sysconf(_SC_CLK_TCK));

    close(sockfd);
    free(buffer);

    /* send statistics to stderr */
    fprintf(stderr, "Linhas enviadas:        %d\n", sentLines);
    fprintf(stderr, "Tamanho da maior linha: %d\n", longestLine);
    fprintf(stderr, "Caracteres enviados:    %d\n", sentBytes);
    fprintf(stderr, "Linhas recebidas:       %d\n", recLines);
    fprintf(stderr, "Caracteres recebidos:   %d\n", recBytes);
    fprintf(stderr, "Tempo total: %4.2fs\n", elapsedTime);

    return 0;
}
```