EA872 - Laboratório de Programação de Software Básico
Módulo 6

Atividades Práticas – Parte I - Aula 10

Larissa Saemi Ganaha **RA: 171730**
Lucca Maia Bollani **RA: 158182**

```
#include <stdlib.h>
#include <stdio.h>
#include "y.tab.h"
#include "serverlib.h"
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>

main(arg_cont, arg_valor)
int arg_cont;
char **arg_valor;
{
	int r , i , j , sz ;
	char *req;
	struct command_list * result = NULL;

	unsigned short porta;	/* porta a se conectar no servidor */
	char area[1024];	/* area para envio e recebimento de dados */
	struct sockaddr_in cliente; /* estrutura de informações sobre os clientes*/
	struct sockaddr_in servidor; /* estrutura de informações sobre o servidor*/
	int soquete;		/* soquete para aceitação de pedidos de conexão */
	int novo_soquete;		/* soquete de conexão aos clientes	*/
	int nome_compr;		/* comprimento do nome de um cliente	*/
	int mensagem_compr;		/* comprimento da mensagem recebida	*/
	FILE * fin, *fout;
	char * buffer = 0;
	long length;
	FILE * f;


	/*Verifique os argumentos. Deve haver apenas um: o número da porta a se
conectar.*/

	if (arg_cont != 2) {
		fprintf(stderr, "Uso: %s número_da_porta\n", arg_valor[0]);
		exit(1);
	}

	/*
	* Primeiro argumento deve ser o número da porta.
	*/
	porta = (unsigned short) atoi(arg_valor[1]);

	/*
```

```
	* Crie um soquete para aceitar pedidos de conexão.*/
	if ((soquete = socket(AF_INET, SOCK_STREAM, 0)) < 0){
		perror("Erro em socket()");
		exit(2);
	}

	/*Ligue o soquete ao endereço do servidor.*/
	servidor.sin_family     = AF_INET;
	servidor.sin_port    = htons(porta);
	servidor.sin_addr.s_addr = INADDR_ANY;

	if (bind(soquete, (struct sockaddr *)&servidor, sizeof(servidor)) < 0){
		perror("Erro em bind().\n");
		exit(3);
	}

	/* Aguarde por pedidos de conexão com um fila de até 5 pedidos.*/
	if (listen(soquete, 5) != 0){
		perror("Erro em listen().\n");
		exit(4);
	}

	/* Entre em laço infinito*/
	printf("\n%s já está aceitando conexões de clientes HTTP.\n",arg_valor[0]);
	fflush(stdout);

	while (1) {
		/*Aceite um pedido de conexão.*/
	nome_compr = sizeof(cliente);
		if ((novo_soquete = accept(soquete, (struct sockaddr *)&cliente,
&nome_compr)) == -1){
			perror("Erro em accept().\n");
			break;
		}
	/*Receba a mensagem no novo soquete.*/
		if ((mensagem_compr = recv(novo_soquete, area, sizeof(area), 0)) ==
	-1){
			perror("Erro em recv().\n");
			break;
		}

		/*Imprima o que recebeu e feche a conexão.*/
		printf("\nMensagem recebida:\n");

		area[mensagem_compr] = '\0';

		yy_scan_string(area);

		printf("Starting program to call the parser and process a request...
	\n");


		if((fout = fopen("tmpOut.txt", "w+")) == NULL){
		printf("Cannot write into %s\n", "tmpOut.txt");
```

```
		exit (0);
	}

	if( yyparse() ){
		//requisicao esta mal formada
		getOutput(NULL, fout);
		return 0;
	}

	result = symtab_get_parse_result();
	getOutput(result, fout);

	/* Fechando o arquivo... */
	fprintf(fout, "\n\0" );
	rewind(fout);
	f = fout;

	//envia o conteudo requisitado para o cliente
	if (f){
		fseek (f, 0, SEEK_END);
		length = ftell (f);
		fseek (f, 0, SEEK_SET);
		buffer = malloc (length);
		if (buffer){
			fread (buffer, 1, length, f);
		 }
		fclose (f);
	}

	if (buffer){
		buffer[length]='\0';
	}

	send(novo_soquete, buffer, strlen(buffer), 0);

	close(novo_soquete);
	close(f);


	break;
	}  /* Laço termina aqui */

	close(soquete);
	printf("O servidor terminou.\n");
	exit(5);

	printf("Finished!\n");
}
```

Nessa atividade, foram realizadas alterações no código fonte para que o servidor estabelecesse uma conexão via *socket* com o cliente vinda do browser. Para isso, foi adaptado o código do *http-dump* utilizado durante a aula 5.
Tais adaptações foram:

1. Para a chamada de *yy_scan_string,* o vetor que contém os comandos e as requisições (área), é passada como parâmetro, diferentemente de um ponteiro *req* que enviava a lista dos comandos.
2. Caso o parser obtenha sucesso, o resultado do parser é salvo em uma variável result e enviada para a função getOutput que irá gerar os cabeçalhos correspondentes, já utilizada na aula passada.
3. Então o arquivo é voltado para o começo, com o comando rewind e, então, através da atuação conjunta do fseek e da função *ftell*, obtém-se o tamanho do arquivo. Sendo assim, aloca-se um buffer com o tamanho correspondente para conseguir armazenar em um vetor os dados contidos no arquivo.
4. Antes de se fechar a conexão soquete, é enviado de volta para o cliente o processamento gerado através da chamada *send(novo_soquete, buffer, strlen(buffer), 0)*, que com as flags (último parâmetro) setadas em zero, se comporta como a função write.

Para realizar a **compilação** do programa, basta dar o comando no terminal `make all`, para que o Makefile criado compile o programa e, então, executá-lo com `./aula8 5000` (número da porta de conexão). Ir em seguida para o browser e digitar na barra de navegação `localhost:5000`.

- Abaixo, quatro casos de teste via web que foram bem sucedidos:


localhost:5007/dir2
WELCOME.HTML !
THIS IS ANOTHER GENERIC HTML PAGE
INSIDE ./dir2 (there's no index here)


localhost:5008/dir1
INDEX.HTML !
THIS IS A GENERIC HTML INDEX PAGE
INSIDE ./dir1


localhost:5003/dir1/texto2.html
TEXTO2.HTML !
THIS IS A GENERIC HTML TEXTO2 PAGE
INSIDE ./dir1

- Abaixo um exemplo de erro de “recurso não existente”:

Not Found

The requested URL /OI was not found on this server.

- Abaixo um caso de erro de “acesso negado”:

Forbidden

You do not have the permission to access this resource