Laboratório de Sistemas Operacionais (CC)
Trabalho Prático 1

Roteiro

1. Objetivos

1. Expandir o conhecimento sobre construção e suporte de distribuições Linux.
2. Implementação e instalação de um servidor Web.
3. Aprendizado sobre o conteúdo do diretório /proc.

2. Descrição

A implementação deste trabalho consiste na geração de uma distribuição Linux que possua um servidor WEB escrito em Python ou C/C++. Para tanto, será necessário adicionar um interpretador python na distribuição gerada (caso seja usado Python), implementar um servidor WEB e escrever uma página HTML simples.

O suporte para linguagem Python pode ser adicionado através do menuconfig do Buildroot (submenu *Interpreter languages and scripting*). Contudo, o Python exige uma toolchain que suporte WCHAR (um tipo de variável usado para codificação de strings UTF-16). Esse suporte também pode ser adicionado através do menuconfig. Será necessário a recompilação total da distribuição (*make clean*, seguido de *make*). Caso seja utilizado C/C++, o próprio compilador cruzado criado pelo Buildroot poderá ser utilizado para compilar a aplicação.

O objetivo da página HTML é fornecer informações básicas sobre o funcionamento do sistema (*target*). Abaixo, segue a lista de informações que devem ser apresentadas na página de maneira dinâmica:

- Data e hora do sistema;
- Uptime (tempo de funcionamento sem reinicialização do sistema) em segundos;
- Modelo do processador e velocidade;
- Capacidade ocupada do processador (%);
- Quantidade de memória RAM total e usada (MB);
- Versão do sistema;
- Lista de processos em execução (pid e nome).

Observe que algumas das informações acima são dinâmicas (mudam constantemente), assim, a cada vez que o usuário acessar a página, o servidor deve atualizar as informações. Ainda, pode-se obter tais informações através do pseudo-filesystem /proc ou comandos como ps e date.

3. Python Web Server

Existem diferentes técnicas para implementação de um servidor WEB em linguagem Python. Os alunos podem utilizar a versão abaixo como passo inicial. Observe que o arquivo index.html pode ser modificado antes de ser enviado ao browser.

```
import SimpleHTTPServer
import SocketServer
import os.path
import sys

class MyRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
                self.path = '/index.html'
                return SimpleHTTPServer.SimpleHTTPRequestHandler.do_GET(self)

Handler = MyRequestHandler

server = SocketServer.TCPServer(('0.0.0.0', 8080), Handler)
server.serve_forever()
```

4. C Web Server

Uma alternativa seria utilizar o servidor abaixo. Observe que é necessário modificar o servidor exemplo para que o conteúdo da página HTML seja lido de um arquivo.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define BUFLEN 1024    //Max length of buffer
#define PORT   8000    //The port on which to listen for incoming data


char http_ok[] =
 "HTTP/1.0 200 OK\r\nContent-type: text/html\r\nServer: Test\r\n\r\n";
char http_error[] =
 "HTTP/1.0 400 Bad Request\r\nContent-type: text/html\r\nServer: Test\r\n\r\n";
char page[] =
 "<html>\n<head>\n<title>\nTest                          page\n</title>\n</head>\n<body>\n<p>Hello
World!</p>\n</body>\n</html>\n";


void die(char *s)
{
        perror(s);
```

```
        exit(1);
}

int main(void)
{
        struct sockaddr_in si_me, si_other;

        int s, i, slen = sizeof(si_other) , recv_len, conn, child = 0;
        char buf[BUFLEN];
        pid_t pid;

        /* create a TCP socket */
        if ((s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
                die("socket");

        /* zero out the structure */
        memset((char *) &si_me, 0, sizeof(si_me));

        si_me.sin_family = AF_INET;
        si_me.sin_port = htons(PORT);
        si_me.sin_addr.s_addr = htonl(INADDR_ANY);

        /* bind socket to port */
        if (bind(s, (struct sockaddr*)&si_me, sizeof(si_me)) == -1)
                die("bind");

        /* allow 10 requests to queue up */
        if (listen(s, 10) == -1)
                die("listen");

        /* keep listening for data */
        while (1) {
                memset(buf, 0, sizeof(buf));
                printf("Waiting a connection...");
                fflush(stdout);

                conn = accept(s, (struct sockaddr *) &si_other, &slen);
                if (conn < 0)
                        die("accept");

                if ((pid = fork()) < 0)
                        die("fork");
                else if (pid == 0) {
                        close(s);

                        printf("Client    connected:    %s:%d\n",    inet_ntoa(si_other.sin_addr),
ntohs(si_other.sin_port));

                        /* try to receive some data, this is a blocking call */
                        recv_len = read(conn, buf, BUFLEN);
                        if (recv_len < 0)
                                die("read");

                        /* print details of the client/peer and the data received */
                        printf("Data: %s\n" , buf);

                        if (strstr(buf, "GET")) {
                                /* now reply the client with the same data */
                                if (write(conn, http_ok, strlen(http_ok)) < 0)
                                        die("write");
                                if (write(conn, page, strlen(page)) < 0)
                                        die("write");
                        } else {
                                if (write(conn, http_error, strlen(http_error)) < 0)
                                        die("write");
                        }

                        exit(0);
                }
```

```
                /* close the connection */
                close(conn);

                child++;
                while (child) {
                        pid = waitpid((pid_t) -1, NULL, WNOHANG);
                        if (pid < 0)
                                die("?");
                        else if (pid == 0) break;
                        else child--;
                }
        }
        close(s);

        return 0;
}
```

5. Entrega

Os alunos podem realizar esta atividade em duplas ou individualmente. Como entrega, será solicitado um tutorial de implementação descrevendo todos os passos necessários para a implementação do trabalho, de forma que outras pessoas possam reproduzi-lo (assim como é feito com os tutoriais de aula). Além disso, todo o desenvolvimento deve ser adicionado ao repositório de fontes e passar a fazer parte da distribuição.

Bom trabalho!