T1 - Construção e suporte de distribuições Linux

Laboratório Sistemas Operacionais

Guilherme Hiago Costa dos Santos
Enrique Bozza Dutra

1. Objetivo

Este tutorial tem como objetivo a geração de uma distribuição Linux que possua um servidor WEB python e escrever uma página html simples, além de que esta distribuição, na máquina target, deve poder ser acessada pela máquina host.

2. Pré-requisitos

Para este tutorial é considerado que você já tenha realizado os tutoriais 1.1, 1.2 e 1.3 (compilação de sistema operacional com Buildroot, emulação com QEMU, configuração de rede com QEMU), vistos em aula.
Após seguir os passos dos tutoriais 1.1, 1.2 e 1.3, vistos em aula, é necessário adicionar suporte a linguagem python. Para isso seguem os comando:

3. Adicionando Python

Para adicionar suporte ao Python é necessário adicionar a variável WCHAR ao tool chain do sistema, para isso, no diretório “/buildroot” rode os comandos:

```
$ make menuconfig (diretório buildroot)
```

Adicione suporte a wchar na toolchain (para poder utilizar python):

tool chain - - >
(*) Enable WCHAR support

Adicione o python 3:

Target packages - ->
Interpreter languages and scripting - - >
(*) python 3

Recompile a distribuição:

```
$ make clean
```

Entre no menu de configurações do kernel Linux:

```
$ make linux-menuconfig
```

Adiciono o driver de ethernet:

```
Device Drivers  --->
    [*] Network device support  --->
        [*]   Ethernet driver support  --->
        <*>     Intel(R) PRO/1000 Gigabit Ethernet support
```

Compile o programa com:

```
$ make
```

Crie o arquivo “S51PythonServer” na pasta buildroot/output/target/etc/init.d

```
#!/bin/sh
case "$1" in
    start)
        python3 /usr/bin/simple_http_server.py
        ;;
    stop)
        exit 1
        ;;
    *)
        exit 1
        ;;
esac

exit 0
```

de privilegios root ao arquivo “chmod +x S51PythonServer”
copie os arquivos python para a pasta buildroot/output/target/user/bin

compile novamente com make

```
sudo qemu-system-i386 --device e1000,netdev=eth0,mac=aa:bb:cc:dd:ee:ff \
    --netdev tap,id=eth0,script=custom-scripts/qemu-ifup \
    --kernel output/images/bzImage \
    --hda output/images/rootfs.ext2 --nographic \
    --append "console=ttyS0 root=/dev/sda"
```

git token

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

4. Adicionando uma Aplicação (Web server)

Para que o programa possa ser executado na máquina target o arquivo (do programa) deve ser copiado para o local que será usado para construir o sistema de arquivos root em um diretório acessível, neste caso foi utilizado o caminho “ *buildroot/output/target/usr/bin/* ”.

Crie um arquivo chamado “S51PythonServer” no diretório “/etc/init.d”. Este arquivo será um script que executa o programa toda vez que iniciar o sistema. Adicione os comando a seguir no arquivo:

```
#!/bin/sh
case "$1" in
      start)
            python3 /usr/bin/simple_http_server.py
            ;;
      stop)
            exit 1
            ;;
      *)
            exit 1
            ;;
esac

exit 0
```

Não esqueça de dar privilégios root para o arquivo, executado o seguinte comando no diretório do arquivo:

```
$ chmod +x S51PythonServer

Compile o programa novamente:
$ make
```

5. Servidor Python

Segue abaixo o programa python que executa o servidor http. O método MyHandler lida com as requisições recebidas, enquanto os métodos `uptime2()`, `get_process_name(id)`, `getprocesses()` e `get_running_process()` fornecem informações sobre o sistema.

Nota: Os últimos dois processos listados são duas formas diferentes de ler os processos executando no sistema, porém o método “`get_running_process()`” pode acabar gerando erro não encontrar resposta para o método “os.getLogin()”, que é usado para ordenar a lista de uma maneira mais legível. Caso o primeiro método gere erro, substitua o método utilizado pelo servidor.

```
from base64 import decode
from calendar import prmonth
from concurrent.futures import process
import time
import http.server#BaseHTTPServer
import os, subprocess
import cpustat

HOST_NAME = '0.0.0.0' # !!!REMEMBER TO CHANGE THIS!!!
PORT_NUMBER = 8000

cpuinfo = cpustat.GetCpuLoad()

def uptime2():
    with open('/proc/uptime', 'r') as f:
        uptime_seconds = float(f.readline().split()[0])
        return uptime_seconds

def get_running_process():
    cmd = ["ps", "aux"]

    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)

    o, e = proc.communicate()

    o = o.decode('ascii').split()
    resp = []
    aux = ""

    for i in o:

        if i.lower() == "command":
            resp += [aux + " " + i]
            aux = ""
        elif i.lower() in ["root", os.getlogin()]:
            resp += [aux]
            aux = i
        else:
            aux += " " + i

    return resp

def get_process_name(id):
    p = subprocess.Popen(["ps -o cmd= {}".format(id)],
stdout=subprocess.PIPE, shell=True)
    return str(p.communicate()[0])

def getprocesses():
        pids = [int(x) for x in os.listdir('/proc') if
x.isdigit()]
        return pids

class MyHandler(http.server.BaseHTTPRequestHandler):
```

```
    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
    def do_GET(s):

        """Respond to a GET request."""
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
        s.wfile.write(bytes("<html><head><title>T1 - Sistemas
Operacionais</title></head>", 'utf-8'))
        s.wfile.write(bytes("<body><p>T1 - Author: Guilherme Hiago
Costa dos Santos</p>", 'utf-8'))

        datahora = os.popen('date').read()
        s.wfile.write(bytes("<p>Data e Hora: %s</p>" % datahora,
'utf-8'))
        s.wfile.write(bytes("<p>CPU uptime: %s in seconds</p>" %
uptime2(), 'utf-8'))
        s.wfile.write(bytes("<p>CPU %s</p>" %
cpuinfo.getcpuinfo(), 'utf-8'))
        #s.wfile.write(bytes("<p>CPU frequency: %s</p>" %
str(psutil.cpu_freq()), 'utf-8'))
        s.wfile.write(bytes("<p>CPU usage: %s</p>" %
str(cpuinfo.getcpuload()), 'utf-8'))
        s.wfile.write(bytes("<p>CPU: %s</p>" %
cpuinfo.getcputime(), 'utf-8'))

        aux = cpuinfo.getraminfo()
        s.wfile.write(bytes("<p>Ram total: %s</p>" % aux[0],
'utf-8'))

        s.wfile.write(bytes("<p>Ram available: %s</p>" % aux[1],
'utf-8'))

        s.wfile.write(bytes("<p>Running Process: </p>", 'utf-8'))

        process_list = get_running_process()

        print(process_list)

        for i in process_list:
            s.wfile.write(bytes("<p>%s</p>" % i, 'utf-8'))

        # If someone went to
"http://something.somewhere.net/foo/bar/",
        # then s.path equals "/foo/bar/".
        #s.wfile.write(bytes("<p>You accessed path: %s</p>" %
s.path, 'utf-8'))
        s.wfile.write(bytes("</body></html>", 'utf-8)'))

if __name__ == '__main__':
    server_class = http.server.HTTPServer
    httpd = server_class((HOST_NAME, PORT_NUMBER), MyHandler)
```

```
    print(time.asctime(), "Server Starts - %s:%s" % (HOST_NAME,
PORT_NUMBER))
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
    print(time.asctime(), "Server Stops - %s:%s" % (HOST_NAME,
PORT_NUMBER))
```

5.1 CPUSTAT

Segue o código do arquivo python que auxilia na extração de informações sobre o sistema operacional que está rodando o servidor.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

'''
Created on 04.12.2014
@author: plagtag
'''
from time import sleep
import sys

class GetCpuLoad(object):
    '''
    classdocs
    '''


    def __init__(self, percentage=True, sleeptime = 1):
        '''
        @parent class: GetCpuLoad
        @date: 04.12.2014
        @author: plagtag
        @info:
        @param:
        @return: CPU load in percentage
        '''
        self.percentage = percentage
        self.cpustat = '/proc/stat'
        self.cpuinfo = '/proc/cpuinfo'
        self.meminfo = '/proc/meminfo'
        self.sep = ' '
        self.sleeptime = sleeptime

    def getcputime(self):
        '''

http://stackoverflow.com/questions/23367857/accurate-calculation-o
f-cpu-usage-given-in-percentage-in-linux
        read in cpu information from file
```

```
        The meanings of the columns are as follows, from left to
right:
            0cpuid: number of cpu
            1user: normal processes executing in user mode
            2nice: niced processes executing in user mode
            3system: processes executing in kernel mode
            4idle: twiddling thumbs
            5iowait: waiting for I/O to complete
            6irq: servicing interrupts
            7softirq: servicing softirqs
        #the formulas from htop
             user    nice   system  idle      iowait irq   softirq
steal  guest  guest_nice
        cpu  74608   2520   24433   1117073   6176   4054  0
0      0      0
        Idle=idle+iowait
        NonIdle=user+nice+system+irq+softirq+steal
        Total=Idle+NonIdle # first line of file for all cpus

CPU_Percentage=((Total-PrevTotal)-(Idle-PrevIdle))/(Total-PrevTota
l)
        '''
        cpu_infos = {} #collect here the information
        with open(self.cpustat,'r') as f_stat:
            lines = [line.split(self.sep) for content in
f_stat.readlines() for line in content.split('\n') if
line.startswith('cpu')]

            #compute for every cpu
            for cpu_line in lines:
                if '' in cpu_line: cpu_line.remove('')#remove
empty elements
                cpu_line = [cpu_line[0]]+[float(i) for i in
cpu_line[1:]]#type casting

cpu_id,user,nice,system,idle,iowait,irq,softrig,steal,guest,guest_
nice = cpu_line

                Idle=idle+iowait
                NonIdle=user+nice+system+irq+softrig+steal

                Total=Idle+NonIdle
                #update dictionionary

cpu_infos.update({cpu_id:{'total':Total,'idle':Idle}})
            return cpu_infos

    def getcpuinfo(self):
        cpu_infos = {} #collect here the information
        with open(self.cpuinfo,'r') as f_info:
            content = f_info.readlines()
            return content[4]

    def getraminfo(self):
        with open(self.meminfo,'r') as f_meminfo:
            content = f_meminfo.readlines()
```

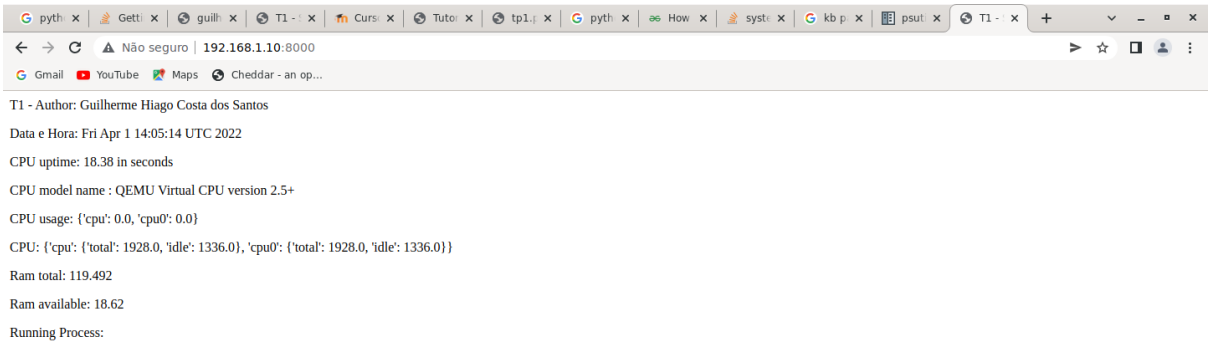
```
            memTotal = int(content[0][17:-4])
            memFree = int(content[1][16:-4])

            return (memTotal / 1000,(memTotal - memFree) / 1000)

    def getcpuload(self):
        '''

CPU_Percentage=((Total-PrevTotal)-(Idle-PrevIdle))/(Total-PrevTota
l)
        '''
        start = self.getcputime()
        #wait a second
        sleep(self.sleeptime)
        stop = self.getcputime()

        cpu_load = {}

        for cpu in start:
            Total = stop[cpu]['total']
            PrevTotal = start[cpu]['total']

            Idle = stop[cpu]['idle']
            PrevIdle = start[cpu]['idle']

CPU_Percentage=((Total-PrevTotal)-(Idle-PrevIdle))/(Total-PrevTota
l)*100
            cpu_load.update({cpu: CPU_Percentage})
        return cpu_load


if __name__=='__main__':
    x = GetCpuLoad()
    while True:
        try:
            data = x.getcpuload()
            print(data)
        except KeyboardInterrupt:

            sys.exit("Finished")
```

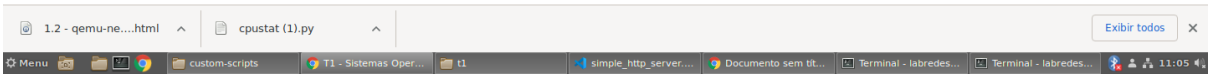
6. Executando o Servidor

Para inicializar o servidor é necessário apenas inicializar o sistema operacional emulado, pois o script que criamos antes cuidará do resto. Para isso, no diretório /buildroot, execute:

```
sudo qemu-system-i386 --device e1000,netdev=eth0,mac=aa:bb:cc:dd:ee:ff \
        --netdev tap,id=eth0,script=custom-scripts/qemu-ifup \
        --kernel output/images/bzImage \
        --hda output/images/rootfs.ext2 --nographic \
        --append "console=ttyS0 root=/dev/sda"
```

Agora para acessar o servidor na máquina host é necessário digitar o ip da máquina target no navegador, no formato <<ip-maquina-target>> : 8000 (8000 é a porta que foi configurada para para o servidor). Você terá uma tela semelhante a essa, com adição de uma lista de processos no fim da página.





7. Código no Github

O código fonte da distribuição linux criada (sem os binários) pode ser encontrado em: https://github.com/GuilhermeHiago/linuxdistro/blob/main/custom-scripts/cpustat.py