[[si:lab:lab4]]

Configurare HTTP. Dezvoltare sistem embedded

Table of Contents
Configurare HTTP. Dezvoltare sistem embedded
Server HTTP
Introducere
Configurarea unui nou sit
Generarea dinamică de pagini
Controlul sistemului
Dezvoltare sistem embedded

Server HTTP

Introducere

NGW100 permite rularea unui server HTTP şi găzduirea unui mic sit web. Serverul se numeşte httpd (abrevierea standard de la HTTP Daemon) şi este pornit din scriptul de iniţializare /etc/init.d/S42httpd (în cazul versiunii de sistem ce foloseşte kernelul 2.6.25.10). Se observă, analizând linia de comandă din script, că paginile servite se află în directorul /www. Fişierul de configurare /etc/httpd.conf specifică următoarele:

- A: * – se permite conectarea clienţilor de pe orice interfaţă de reţea
- .sufix:MIME-type – specifică tipurile de fişier servite, după sufixul acestora
  - exemplu .jpg:image/jpg
- /cgi-bin/webif:root:root – permite accesul în directorul respectiv numai pe baza autentificării

Sistemul standard instalat pe NGW100 conţine o unealtă pentru administrarea prin interfaţă web numită webif. Aceasta va fi înlocuită cu paginile create de noi.

Configurarea unui nou sit

Creaţi un script /etc/init.d/S40webdir care va monta un sistem de fişiere tip tmpfs în directorul /www, înlocuind astfel conţinutul preinstalat al acestuia. Fişierele servite de httpd se vor afla astfel în RAM şi vor fi şterse la fiecare resetare a sistemului. Pentru a exista fişiere în /www la pornirea sistemului, creaţi un script /etc/init.d/S41webupd care va copia conţinutul directorului /home/www în /www, ştergând în prealabil conţinutul directorului /www pentru a permite actualizarea manuală, fără restartarea sistemului.

1. Intraţi în directorul /www
2. Creaţi un fişier index.html de probă, cu un titlu şi 2-3 cuvinte. Conectaţi-vă din browser la 10.0.0.1, observând funcţionarea serverului.

Generarea dinamică de pagini

Serverul web poate genera dinamic pagini. Aceasta se realizează prin execuţia de scripturi aflate în directorul cgibin din rădăcina serverului, folosindu-se o metodă standard numită Common Gateway Interface. CGI presupune execuţia unui script specificat în cererea HTTP (identificat ca orice alt fişier de pe server), care poate primi parametri dintr-o formă web pe stdin şi prin variabilele de mediu, respectiv poate genera pagini dinamice scriind pe stdout.

Mergeţi în directorul cgi-bin în /www şi creeaţi un fişier 1.sh în acesta. *Faceţi-l executabil* şi scrieţi în el următoarele linii (de ex. copy-paste în terminal în vi insert-mode):

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<HTML><HEAD><TITLE>Sample CGI Output</TITLE></HEAD>"
echo "<BODY><p>"
echo -n "Generated "
date
echo "</p><pre>"
env
echo "</pre>"
echo "</BODY></HTML>"
```

Accesaţi http://10.0.0.1/cgi-bin/1.sh şi observaţi rezultatul. Scriptul executat trimite ieşirea pe stdout, aceasta fiind trimisă browserului. În cazul de faţă sunt afişate variabilele de mediu.

Controlul sistemului

Metoda CGI permite execuţia oricărui script pe server, putându-se astfel efectua orice operaţie. În mod normal s-ar restricţiona accesul la sistem spre exemplu rulând serverul web sub un utilizator fără privilegii. Creaţi un fişier on.sh, faceţi-l executabil şi aprindeţi din el un LED de pe placă. Creaţi un fişier similar off.sh pentru stingerea respectivului LED. Creaţi în index.html (în rădăcina ierarhiei web) două forme cu metoda POST, conţinând un singur buton submit fiecare, pentru a aprinde respectiv stinge LEDul din interfaţa web.

Exemplu de cod HTML:

```
<form method="post" action="/cgi-bin/on.sh">
<input type="submit" name="on" value="LED On" />
</form>
```

Chiar dacă scripturile nu generează ieşire, browserul le va interpreta ca downloaduri goale (pentru că a emis o cerere şi aşteaptă un fişier ca răspuns, mecanismul CGI se desfăşoară doar pe server, fără ştirea sau interesul browserului). Trebuie ca ele să trimită un răspuns de tip redirect la pagina iniţială pentru a obţine o interfaţă rezonabilă. Aceasta se face adăugând în scripturi următorul cod:

```
echo "HTTP/1.0 302 OK"
echo "Location: /"
```

Pentru a aprinde respectiv stinge LEDul se poate folosi metoda cunoscută cu GPIO sau o metodă de nivel mai înalt ce beneficiază de faptul că la compilarea kernelului s-a specificat în fişierul de descriere a plăcii că anumite GPIO-uri au LEDuri conectate pe ele.

Astfel, pentru a aprinde LEDul A, se poate folosi comanda

```
echo 255 > /sys/class/leds/a/brightness
```

respectiv 0 pentru a îl stinge.

Dezvoltare sistem embedded

Până acum s-au tratat următoarele subiecte:

- compilarea unei aplicații Hello World pentru o altă arhitectură
- configurarea unui bootloader
- descrierea sistemului de fișiere
- configurare scripturilor de inițializare a sistemului
- folosire module de kernel
- configurarea sistemului pentru a funcționa în rețea.

Cum se leagă toate acestea cu dezvoltarea sistemelor embedded? Toate aceste subiecte sunt un subset al etapelor necesare construirii unui sistem embedded.

Dorim să construim un sistem "de la zero". Vom presupune avem partea hardware gata făcută și mai mult, avem și suport pentru toate perifericele. (Presupunem că vrem să construim un sistem bazat pe Linux. Sunt situații în care Linux nu este de preferat ca sistem de operare, sisteme la care nu sunt aplicabili toți pașii de mai jos). Pașii în cazul acesta ar fi următorii:

Alegem un bootloader

În funcție de hardware-ul pe care îl avem la dispoziție, s-ar putea să avem de ales sau nu. Este posibil să nu fie platforma hardware suportată de niciun bootloader în mod implicit, caz în care va trebui portat un bootloader de la o platformă asemănătoare. Odată ales, bootloader-ul se scrie prin JTAG în memoria sistemului, la o locație care să-i permită rularea la resetarea procesorului.

Alegem sistemele de fișiere/partițiile

Aici depinde foarte mult de ce avem la dispoziție. De exemplu, pe NGW avem două flash-uri și o interfață mmc și avem mai multe posibilități, dintre care enumerăm câteva:

1. kernel pe flash paralel, root pe mmc
2. root pe flash paralel, /usr pe flash serial (default)
3. kernel pe prima partiție de pe mmc, root pe a doua partiție de pe mmc

Fiecare variantă are avantaje/dezavantaje, în momentul în care alegem va trebui sa ținem cont atât de ele cât și de specificul aplicației, condițiile de exploatare ale dispozitivului ș.a.m. De exemplu:

- ext2 nu oferă jurnalizare, dar bootloader-ul UBoot nu suportă ext3
- memoria flash se uzează în timp (număr limitat de scrieri pe bloc), părțile sistemului de fișiere unde au loc scrieri pot fi ținute într-o memorie separată (de exemplu, directoare cu multe scrieri precum /var și /tmp sunt montate în RAM pe NGW)
- viteza flash-ului paralel este mai mare decât cea a flash-ului serial și decât a cardului mmc
- memoria flash va avea probabil dimensiuni mai mici decât un card mmc (deși bootloader-ul nu poate încărca un kernel de Linux de pe o partiție cu mai mult de 500MB, odată încărcat Linux se va putea folosi și restul cardului)

În acest pas se configurează și bootloader-ul pentru a încărca de unde trebuie kernel-ul și pentru a-i da argumentele potrivite.

Alegerea programelor și instalarea lor

Acest pas nu a fost tratat la laborator, constă în cross-compilarea tuturor pachetelor necesare sistemului și instalarea lor în viitoarea rădăcină a sistemului embedded.

- Știți care este un sistem (aproape) minim de la care se poate porni?
  - /bin/busybox (compilat cu uClibc static, deci nu are nevoie de uclibc.so)
  - /bin/sh - link la /bin/busybox
  - /bin/ls (for convenience)
  - /dev/console
  - /dev/null
  - /tmp
  - /var

Construirea sistemului de fișiere se poate face manual sau cu un sistem de scripturi precum Buildroot.

Scripturi de inițializare

Aplicația va avea nevoie de anumite servicii, acestea trebuiesc configurate și pornite odată cu sistemul. Aici intră configurările de rețea, montarea celorlalte partiții, și restul de scripturi de inițializare.

Creearea aplicației

Odată ce este pus în funcțiune întreg sistemul, există driveri pentru toate perifericele necesare, aplicația poate fi total independentă de platformă.

Administrare, reconfigurare

Odată pus în funcțiune sistemul, el mai poate fi accesat fie printr-o consolă sau prin rețea (ssh). Ce se întâmplă însă cu update-urile? Soluția cea mai bună ar fi update-ul remote, dar sistemul trebuie gândit de la început pentru a permite acest lucru:

- Varianta cea mai simplă o reprezintă sincronizarea sistemului de fișiere:

```
rsync -e "ssh -l andrei" -r -l -p -t -D -v --progress <ip>:/home/andrei/target/* /
```

sincronizează cele două foldere. (*Exercițiu* Căutați cu man rsync ce înseamnă fiecare parametru) Comanda face un update al sistemului. De ce ar trebui însă o altă metodă? Ce se întâmplă dacă sistemul este resetat în timpul acestui update

- Sistemul de fișiere este împachetat odată cu kernelul (cpio sau initramfs). Tot sistemul este practic un singur fișier cu CRC.
  1. Se transferă pe sistem update-ul (se scrie într-o altă locație pe disc, nu se suprascrie sistemul existent)
  2. Se verifică update-ul
  3. Se reconfigurează bootloader-ul pentru a folosi cealaltă locație de memorie.
- Mediu extern pe care este pus sistemul este rescris cu noul sistem. (update pe card SD)

Logging

Pentru a face analiza unui crash în sistem sau pentru a ține evidența unor parametri este nevoie de logging.

- Este foarte important mediumul pe care este scris logul. O memorie flash de exemplu va fi uzată foarte mult de logging.
  - Ce împiedică folosirea /var/log pentru logging pe NGW?
- Se poate face logging remote pe o altă mașină, lucru care elimină problema memoriei.
- Trebuie limitată dimensiunea maximă a log-ului (logrotate)
  - În cât timp credeți ca se umple o memorie de 8MB cu mesaje la interval de 30 de secunde?