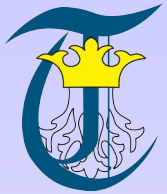

Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare distribuite și în timp real

– curs –
master SECI anul I

ș.l. dr. ing. Kertész Csaba-Zoltán

Embedded Linux

- oferă suport pentru dispozitive embedded
- kernel foarte modularizat
- compatibil POSIX
- ușor de portat aplicații din domeniul desktop
- suportă a gamă mare de dispozitive
- poate fi configurat pentru procesare real-time (în anumite condiții)

Mic istoric

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Date: 25 Aug 91 20:57:08 GMT
Local: Sun, Aug 25 1991 10:57 pm
Subject: What would you like to see most in minix?

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Istoric embedded Linux

- 1996 – RTLinux
  - nucleu mic RTOS care rulează Linux ca un proces
- 1997 – uCLinux
  - Linux pentru sisteme fără MMU
- 1999 – caracteristici soft real-time în kernel
  - Montavista Linux (distribuție cu suport real-time)
- 2000 – Qt/Embedded
  - PDA-uri cu Linux
- 2001 – uClibc

Avantaje embedded Linux

- independență de furnizori (de SO)
  - schimbarea între furnizori de distribuții ușoară
- time-to-market
  - existența multitudinii de soluții open-source (pentru desktop și embedded)
- costuri mici
  - royalty free, costuri mici de training prin documentație vastă pe Internet
- compatibil POSIX
  - aplicațiile sunt ușor portabile

Distribuții embedded Linux

- există distribuții dedicate embedded
- configurații de kernel orientate embedded
  - posibilități mai mari de eliminare componente nefolosite din kernel (MMU, FS, NET)
- includ toolchain-uri dedicate arhitecturii
- portate pe diferite arhitecturi şi plăci de evaluare
  - Board Support Packages

Distribuţii

- BlueCat Linux
- Cadenux
- Denx
- Embedded Debian / Embedded Ubuntu
- ElinOS
- Metrowerks
- MontaVista Linux
- RTLinux
- Timesys Linux

Arhitectura Linux

- nucleul Linux conține modulele principale:
  - HAL (Hardware Abstraction Layer)
  - MM (Memory Manager)
  - Scheduler
  - FS (File System)
  - IO subsytem
  - Networking subsystem
  - IPC (InterProcess Communication)

Hardware Abstraction Layer

- HAL virtualizează platforma hardware pentru a permite o portare ușoară a diferitelor drivere
- conține practic BSP (Board Support Package) pentru arhitectura hardware țintă
  - nu oferă însă un API standard din motive tradiționale (Linux dezvoltat prima dată pentru 386 fără gândul la portări ulterioare)
  - există însă tentative de standardizare a interfeței (mai ales în cazul ARM și PowerPC)

Componente HAL

- HAL suportă următoarele componente principale:
  - procesor, cache, MMU
  - interrupt controller
  - DMA
  - timere
  - consolă sistem
  - controler de magistrală
  - power management

Planificatorul (scheduler)

- planificatorul permite funcționarea multitasking a sistemului
  - aplicațiile rulează pseudoparalel
- firele executate de scheduler pot fi:
  - kernel threads (fire de execuție în interiorul kernelului fără context)
  - user process (fire de execuție, împreună cu context, date, stivă, heap, text, bss + stivă kernel)
  - user threads (fire de execuție, cu instanță de execuție separată, dar heap, code, bss şi data partajată cu alte fire din cadrul aceluiași proces)

Planificatorul pentru real-time

- începând cu apariția nevoii de portare a Linuxului pentru sisteme embedded, planificatorul a suferit îmbunătățiri continue pentru a permite a rulare cât mai rapidă
- începând cu versiunea 2.6.8, planificatorul din Linux este unul care rulează în O(1), permițând astfel o operare deterministă a sistemului (cerință importantă pentru a satisface criteriul de real-time)
  - de la versiunea 2.6.23 kernelul Linux a trecut la un planificator O(log n) Completely Fair Scheduler, distribuțiile embedded rămânând pe O(1)

Managerul de memorie

- realizează alocarea dinamică a memoriei pentru componentele SO (scheduler, FS, networking, drivere)
- implementează memoria virtuală pentru aplicații (paginare)
  - memoria este împărțită în pagini de 4kB (de obicei) și pot fi alocate independent kernelului sau aplicațiilor
- separă zonele de memorie alocate diferitelor procese pentru protecție între ele (segmentare)

Managerul de memorie

- Linux oferă a separare între memoria alocată kernelului și memoria alocată proceselor
  - kernelul poate vede toate paginile, procesele numai paginile alocate procesului respectiv
  - paginile alocate proceselor pot fi supuse paginării (între memorie și storage) reducând dimensiunea necesară a memoriei
- Pe sistemele fără MMU, dezvoltatorul trebuie să aibă grijă de această separare

Sistemul de fișiere

- Linux combină diferitele sisteme de fișiere într-un strat comun Virtual File System
- VFS oferă o interfață comună pentru toate dispozitivele din sistem
- Linux are nevoie întotdeauna de un FS (cât de cât minimal):
  - programele sunt stocate într-o memorie nevolatilă (HDD, Flash) în forma unor fișiere accesibile prin intermediul FS
  - toate dispozitivele de intrare/ieșire existente în sistem sunt tratate ca fișiere

Sistemul de fișiere

- sistemul Linux oferă un root file system, care trebuie montat la pornire (dacă nu există, kernelul nu va porni)
- pe acest root vor fi montate celelalte sisteme de fișiere:
  - ext2, ext3, reiserfs (pentru harddiskuri)
  - romfs (memorie ROM)
  - ramfs (memorie RAM)
  - jffs2 (flash)
  - procfs (informații despre sistem /proc)
  - devfs, udev (dispozitive IO /dev)

Subsistemul IO

- oferă o interfață uniformă pentru dispozitivele din sistem
  - character devices (dispozitive secvențiale)
  - block devices (dispozitive accesibile aleator)
  - network devices (dispozitive pe rețea)
- accesul dispozitivelor se realizează prin intermediul FS
- controlul dispozitivelor se realizează prin interfața IO (ioctl)

Subsistemul de rețea

- una din principalele atuuri al sistemului Linux este orientarea spre operare în rețea
- nucleul Linux include implementări complete de stive de rețea, cea mai importantă fiind stiva TCP/IP
- foarte multe subsisteme Linux se folosesc de modularitatea oferită de stivele de rețea pentru realizarea separării și comunicării între ele

IPC

- pentru comunicații între procese Linux include
  - semnale (comunicație asincronă = întreruperi soft)
  - pipe (prin intermediul FS)
  - socket (prin intermediul networking)
  - System V IPC (metodele IPC definite de POSIX):
    - semafoare
    - shared memory
    - message passing

Pornirea unui sistem Linux

- sistemul Linux fiind în sine un sistem complex, are nevoie de o secvență de pornire prin intermediul căreia se vor încărca diferitele componente ale sistemului
- pe sistemele embedded această secvență de pornire trebuie utilizată pentru a se realiza într-un timp cât mai scurt
- fazele pornirii sunt:
  - boot loader
  - inițializare kernel
  - inițializare user-space

Boot loader

- prima fază la pornirea sistemului
- presupune o inițializare a hardwareului:
  - inițializare procesor (frecvență, cache, interfață cu memoria)
  - inițializare port serial pentru consola Linux
- încarcă kernelul din spațiul de stocare nevolatil în memorie
- pasează argumente pentru kernel (încarcă o zonă de memorie cu valori prestabilite)
- salt la punctul de început al kernelului

Inițializare kernel

- începe cu o rutină de configurare a mediului pentru rutinele C (scris în asm)
  - inițializează MMU (dacă există)
  - resetează zonele de memorie alocate bss la 0
  - setează stiva pentru prima rutină C
    - `start_kernel()` execută a serie de rutine şi termină într-o buclă infinită (idle)
- setup_arch()
  - restul iniţializării platformei
- trap_init()
  - inițializează excepțiile (întreruperi software)

Inițializare kernel

- init_IRQ()
  - inițializează controlerul de întreruperi și descriptorii de întreruperi (care vor prelua întreruperile și pasa la rutinele utilizatorilor)
- time_init()
  - inițializează un timer pentru a realiza un tick periodic (folosit de scheduler)
- console_init()
  - inițializează portul serial pentru consolă kernel
  - de acum încolo toate mesajele de pornire vor fi afișate

Inițializare kernel

- calibrate_delay()
  - inițializează rutinele de întârziere oferite de sistem
- inițializare subsisteme
  - se inițializează schedulerul, managerul de memorie și VFS
- creare un nou proces init, după care start_kernel() va continua ca un proces idle
- init va inițializa driverele și va monta sistemul de fișiere root, după care se va trece în user-space

Inițializare user-space

- este dependent de distribuție
- de obicei se realizează de /sbin/init, care citește configurațiile găsite în /etc/inittab și le execută
- fișierele de configurație pot conține scripturi shell ce vor executa și alte programe de inițializare
- se inițializează procesele care nu interacționează cu utilizatorul (daemon-ii)
- se oferă o consolă de login pentru utilizator

Compilarea cross-platform

- de obicei în platformele embedded nu există resurse suficiente pentru a permite compilarea sistemului
- pentru compilarea sistemului pentru o nouă platformă (embedded) se folosește o suită de compilare cross-platform
- exemplu gnu-toolchain
- acest toolchain permite compilarea pe o mașină host a sistemului destinat unei mașini target care poate fi pe o arhitectură diferită

GNU toolchain

- binutils
  - suită de programă care ajută la compilare
  - ar, as, ld, nm, objcopy, objdump, ranlib, size
- gcc
  - compilatorul C
- glibc
  - suită de biblioteci standard
  - include și toate funcțiile pentru apeluri sistem