Systémové programovanie

I., 16. 2. 2012

POSIX

- Štandard unixovského systému
  - + štandard písania UNIX aplikácií
  - „čo musí systém dodržať, aby sa mohol nazývať unixom“
  - Podpora prenositeľnosti
- Vznikol neskôr než UNIX
- Rozvetvovanie unixovských systémov, rozličné systémové volania, rozličné návratové hodnoty
- Potreba zjednotiť systémy a zmenšiť počet #ifdefov

PROCESY A SIGNÁLY

Štart systému

- Bootloader načíta jadro do pamäte
- Jadro rozbalí inicializačný RAMdisk
  - Initdrd
  - Dvojnásobne komprimované
    - Cpio + gz
  - Nepovinné, všetko (ovládače...) možno zakompilovať do jadra
- RAMdisk nastaví ako základný filesystém
  - / (root)
  - Doň sa domontujú ostatné

- Sanity checks
- Spustia sa základné procesy
- Spustí sa úplne prvý proces
  - Init (skript)
  - Dostane identifikátor PID=1
- Inicializácia ďalších procesov: závislá od Distra

Debian: štýl SystemV

- Vytvorí sa dávka skriptov pre spúšťanie jednotlivých procesov
  - Obvykle shellskripty
- Sekvenčné spúšťanie
- Závislosti medzi démonmi možno pekne zobraziť
  - **démon**: analógia rezidentného programu z MS-DOS
  - Beží na pozadí, nepotrebuje interakciu s používateľom
- Obvykle v /etc/rc.2/@sXXXXXYYYYY
  - S – štartuj: skript pre štart
  - K – ukonči sa: skript pre ukončenie
  - XXX – číslo, priorita spúšťania
  - YYY - názov
  - 2 - runlevel

Ubuntu: upstart

- Paralelné
- Možnosť definovať pravidlá pre závislosti
- Ak nie sú žiadne závislosti, všetko sa spustí naraz

- Ďalšie procesy spustené z init sa stanú dcérskymi procesmi
- Vzniká hierarchia procesov
- PIDy inkrementálne narastajú
  - do 32768
  - Medzery sa nevypĺňajú
  - ...až po pretečení SIGINT

Procesy vs program

- Program = zdrojový kód
- Proces = bežiaca inštancia programu
  - program zavedený do pamäte
  - má pridelené ID: PID
- Stavy procesu:
  - Sleeping: čaká na vstup, proces sa aktívne zastaví a čaký
  - Running: bežiaci
  - Closed: ukončený
  - Stopped: zastavený
    - Zvonku, napr. z gdb
  - Zombie

Multitasking

- Starý multitasking
  - Aplikácia sa sama musela zrieknuť procesorového času
  - Neexistovala moc, ktorá zbavila proces času
- Preemptívny multitasking
  - Riadenie na úrovni jadra
  - Jadro obsahuje plánovač (scheduler)
  - Rozličné spôsoby:
    - Time-slicing, dokola sa prideľuje procesorový čas
    - Systém prioritných front

Systém prioritných front

- Dobré procesy sú odmeňované, zlé trestané
- Procesy, ktoré žerú veľa CPU, budú mať zníženú prioritu
- Opačne: proces, ktorý čaká na vstup: čím dlhšie spí, tým lepšiu prioritu dostane
  - a nepoužíva while-sleep(), ale je kultúrny

Životný cyklus

- Sleeping: čaká na vstup, proces sa aktívne zastaví a čaký
- Running: bežiaci
  - Hneď ako dostane cpu time, rozbehne sa
  - Viacero procesov môže byť v tomto stave
- Closed: ukončený
- Stopped: zastavený
  - Zvonku, napr. z gdb alebo signálom SIGQUIT
    - Ctrl-Z
  - Čaká na pokračovanie
- Zombie
  - Dobehnutý, ale neupratali sa všetky zdroje
  - Po forku() sa alokujú zdroje pre dcérsky proces
  - Ak dcéra dobehne, zdroje ostávajú alokované
    - Ten treba upratať
  - Na starých systémoch: veľa alokovaných procesov mohlo zabrániť súvislej alokácií pamäte
  - Plus stále existuje limit na počet procesov (maximálne limit na PID)

Signály

- Prvý a najstarší spôsob medziprocesovej komunikácie (IPC)
- Delenie
  - Základné: 28
    - SIGINT, SIGKILL, SIGTERM, ...
    - Súčasť POSIX (signal.h)
  - Realtime (bezpečné)
- Ovplyvniteľné/neovplyvniteľné
  - Chytiť a ovplyvniť
    - SIGTERM (defaultný pre kill): vieme odchytiť a ignorovať, alebo odchytiť a upratať zdroje
    - SIGKILL (kill -9): nevieme ignorovať
  - Slúži na odstrelenie procesov: kultúrne/nekultúrne

Delenie signálov

- Synchrónne / asynchrónne
- SIGALRM (alarm): synchrónne: získame a vyriešime
  - Vieme pozastaviť beh programu, kým nenastane signál
- SIGTERM: asynchrónne: v ľubovoľnom momente behu

Starý spôsob

- Signal()
  - http://pubs.opengroup.org/onlinepubs/009695399/functions/signal.html
  - Číslo signálu
  - Pointer na funkciu, ktorá obslúži signál
- Použitie vo forku
  - Dieťa odosiela rodičovi SIGCHLD, ak dobehne
  - Defaultne sa v rodičovi neobsluhuje
  - Ak sa neobsluhuje, môžu vznikať zombíci

- Staré volanie
- Namiesto obsluhy dve konštanty pointrov pre obslužné funkcie:
  - SIG_IGN: ignorovaný signál
    - Vieme odignorovať Ctrl-C (SIGINT)
    - Alebo SIGTERM
  - SIG_DFL: obnovuje pôvodnú obsluhu signálu
    - Vieme nahodiť SIG_IGN
    - Vykonať veci
    - Obnoviť pôvodnú obsluhu cez SIG_DFL
- Ďalšie funkcie: raise(), pause(), wait()

Nový spôsob pre signály

- Princíp ostáva, iné volanie
- sigaction()
  - Číslo signálu
  - Struct s maskou signálov
    - Jedným volaním vieme vyplniť obsluhu viacerých signálov naraz
    - Sigemptyset(), sigfillset()
  - Výstupný parameter: výstupná maska

Čo ak pri obsluhe signálu nastane iný signál?

- Funkcia obsluhy signálu musí byť reentrantná
  - Do funkcie môže vbehnúť viacero obslúh naraz
  - Existuje zoznam funkcií, ktoré možno bezpečne volať
- Štruktúry v jadre musia byť reentratné
- Systémové volania tiež

- Kernel space: oddelený priestor v pamäti, v ktorom operuje len jadro
  - Dokonca odlišné veľkosti než v userspace
  - Odlišné veľkosti dátových typov
  - Len on má prístup k hardvéru
  - Zverejňuje systémové volania
    - Volania rutiny v jadre
  - Pri výpise **ps** sú procesy v [..] v kernel space
  - Jadro = hypervisor, forma virtualizácie, jadro pridelí pamäť procesu a spravuje ho.
  - Proces ani nevie, že nejaký kernel space existuje
- User space
  - Priestor pre procesy, knižnice...

- Program beží v kernelspace, lebo číta spbor
  - Používa napr. Sys_read()
- Nastane signál
- Obslúži sa na rozhraní KS a US
- Môže nastať chyba EINTR, že čítanie nebolo dokončené
- Signál totiž prerušuje typický beh
- Možno nakonfigurovať stav, keď po EINTR sa čítanie resetne
- Vie to byť veľmi komplexné, niektorí radia sa vyhnúť signálom úplne ;-)

Prerušenia (interrupt)

- Signáloidná komunikácia najnižšej úrovne (hardvér)
- Systém ma tabuľku prerušení
- Ovládač registruje prerušenia a na ich základe komunikuje s OS cez signály
- Dnes: dve inštrukcie v CPU Intelov
  - Sysenter, sysexit v inštrukčnej sade procesora
  - Po vyvolaní prerušenia treba uložiť stav procesora (inštrukčný čítač, stack pointer...)
  - Už to nemusí riešiť jadro, ako kedysi