UNIX folyamatok kommunikációja

kiegészítő fóliák az előadásokhoz

Mészáros Tamás
http://home.mit.bme.hu/~meszaros/

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Az előző részekben történt...

- A kernel
  - felügyeli a folyamatokat, menedzseli az erőforrásokat
  - elkülöníti egymástól a folyamatokat (mindenkinek saját virtuális gép)
  - réteges, moduláris felépítésű
  - a folyamatok a kernellel rendszerhívásokon keresztül kommunikálnak
- A folyamatok
  - a felhasználói programok futás alatt álló példányai
  - UNIX alatt szülő-gyerek kapcsolatban állnak egymással
- A kommunikáció elmélete
  - közös memórián keresztül, üzenetekkel, távoli eljáráshívás (RPC)
  - erőforrások védelme, kritikus szakasz, szemafor
  - blokkoló (szinkron), nem blokkoló (aszinkron)

A kommunikáció (alap)csatornái – az előadás témája

- Jelzések
  - események keltése és kezelése
- Csővezetékek
  - FIFO kommunikáció a „rokonságon” belül és kívül
- Szemaforok
- Üzenetsorok
  - diszkrét, típusos üzenetek folyamatok között
- Osztott memória
  - azonos fizikai memóriaterület használata több folyamatban
- „hálózati” (socket) kommunikáció
- Távoli eljáráshívás UNIX módra

UNIX jelzések

- Cél: egy folyamat (vagy folyamatcsoport)
  - értesítése a kernelben, más folyamatokban, valamint önmagában bekövetkezett eseményekről
  - szinkronizálása más folyamatokhoz (ma már van jobb megoldás)

- Jelzés típusa (SIGINT, SIGCHLD, SIGKILL, … lásd: `kill -l`)
  - rendszer: kivételek (pl. hibák), kvóta, riasztás, értesítés (egy gyerek leállt)
  - felhasználói: emberek (ctrl + c, ctrl + z), folyamatok (tetszőleges céllal)

- A működés áttekintése
  - jelzést keltése (rendszerhívás vagy valamilyen esemény bekövetkezése)
  - a kernel értesíti a címzetteket a jelzésről
  - a címzett egy jelzéskezelő eljárásában fogadja a jelzést

- Problémák a megvalósítással
  - a keltés és a kézbesítés időben szétválik (akár elég messze is)
  - sokféle implementáció, némelyik nem túl megbízható (elveszhet jelzés)

Jelzések keltése és kezelése

- Jelzések keltése (folyamat által)

```
#include <signal.h>        /* kill() */
kill(pid, SIGUSR1);        /* jelzés küldése */
```

- Jelzések kezelése
  - többféle kezelési eljárás lehetséges, bizonyos keretek között állítható
    - Core: core dump és leállítás (`exit()`)
    - Term: leállítás (`exit()`)
    - Ign: figyelmen kívül hagyás
    - Stop: felfüggesztés
    - Cont: visszatérés a felfüggesztett állapotból (vagy ignore)
  - az alkalmazás saját kezelőfüggvényt is megadhat

```
signal(SIGALRM, alarm);   /* kezelőfüggvény beállítása */
void alarm(int signum) { ... }  /* a kezelőfüggvény */
```

  - a jelzés típusától függ, hogy a fentiek közül mi az alapértelmezett, illetve minek a beállítása engedélyezett
    - pl. a SIGKILL nem hagyható figyelmen kívül és nem definiálható rá jelzéskezelő függvény

UNIX jelzések: példák

```
#include <signal.h>          /* signal(), kill() */
#include <unistd.h>          /* getpid() */
#include <sys/types.h>       /* pid_t */
pid_t pid = getpid();        /* saját PID */

kill(pid, SIGSTOP);          /* STOP jelzés küldése */
```

Ekvivalens parancssori utasítás: `kill -STOP <PID>`

```
signal(SIGCLD, SIG_IGN);  /* nem foglalkozunk a gyerekekkel */

signal(SIGINT, SIG_IGN);  /* nem foglalkozunk a ctrl+c jelzéssel */

signal(SIGINT, SIG_DFL);  /* alapértelmezett kezelő beállítása */

signal(SIGALRM, alarm);   /* jelzéskezelő függvény beállítása */
void alarm(int signum) { ... }  /* az eljárás */
alarm(30);                /* alkalmazás: ALARM jelzés 30mp múlva */
```

`man -s 7 signal` (részlet)

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

UNIX csővezetékek: `pipe()`

- Cél: folyamatok közötti adatátvitel (`ls -la | more`)

- Jellemzők
  - csak szülő-gyerek (leszármazott, testvér) viszonylatban
  - adatfolyam (nincs üzenethatár, tipizálás)
  - egyirányú adatfolyam (író → olvasó) (több író és olvasó is lehet!)
  - limitált kapacitás: pl. 4k (Linux < 2.6.11, 65k (Linux >= 2.6.11)

- A megvalósítás
  - egy folyamat létrehoz egy csővezetéket (`pipe()`)
  - a kernel létrehozza az adatstruktúrákat és olvasási/írási leírókat ad vissza
  - (a folyamat továbbadja a leírókat a gyerekeinek)
  - a leírók segítségével kommunikálnak a folyamatok (`read(), write()`)

- Korlátok, problémák
  - nincs címzés, tipizálás, üzenethatár
  - csak a „rokonságban” működik

UNIX elnevezett csővezetékek (named pipe, FIFO)

- Az egyszerű csővezetékek legkomolyabb problémájának megoldása
  - független folyamatok kommunikációja
  - avagy egy már létező csővezeték elérése egy másik folyamat által
- Jellemzők
  - nem csak szülő-gyerek viszonylatban alkalmazható
  - ugyanúgy működik, mint a csővezeték
  - a létrehozás a fájlrendszer segítségével történik
  - lehetséges a kétirányú kommunikáció (megnyitás olvasásra és írásra)
- Példa: kommunikáció az init folyamattal
  - a fájlrendszerben látható a csővezeték:

```
prw-------  1 root root 0 Jan  1 12:38 /dev/initctl
```

(a fájlrendszerrel kapcsolatos részletek később)

UNIX System V IPC

- Cél: folyamatok közötti „szabványos”, egységes kommunikáció
  - adatátvitel
  - szinkronizáció
- Közös alapok (fogalmak)
  - erőforrás: a kommunikáció eszköze (l. lentebb)
  - kulcs: azonosító az erőforrás eléréséhez (egy 32 bites szám)
  - közös kezelőfüggvények: `*ctl(), *get( … kulcs …)`
  - jogosultsági rendszer:
    - létrehozó, tulajdonos és csoportjaik
    - a szokásos hozzáférés-szabályozási rendszer (felhasználó, csoport, mások)
- Erőforrások
  - szemaforok
  - üzenetsorok
  - osztott memória

bővebb infó: `man svipc` `man ipc`

UNIX System V IPC: szemaforok

- Cél: folyamatok közötti szinkronizáció
  - P() és V() operátorok
  - szemaforcsoportok kezelése

- Használat

```
sem_id = semget(kulcs, szám, opciók);
```

  - adott számú szemaforhoz nyújt hozzáférést (adott kulccsal és opciókkal)
    (a tényleges létrehozás és az egyszerű hivatkozás az opciókban válik szét)

  - az ops struktúrában leírt műveletek végrehajtása (részletek: man semop):

```
status = semop(sem_id, ops, ops_méret);
```

  - egyszerre több művelet, több szemaforon is végrehajtható
  - blokkoló és nem blokkoló P() operáció is lehetséges
  - egyszerű tranzakciókezelésre is van lehetőség

UNIX System V IPC: üzenetsorok

- Cél: folyamatok közötti adatátvitel
  - diszkrét, tipizált üzenetek
  - nincs címzés, üzenetszórás

- Használat

```
msgq_id = msgget(kulcs, opciók);
```

  - adott kulcsú üzenetsorhoz nyújt hozzáférést (adott opciókkal)
    (a tényleges létrehozás és az egyszerű hivatkozás az opciókban válik szét)

  - üzenetküldés (az msg tartalmaz egy típus azonosítót is):

```
msgsnd(msgq_id, msg, méret, opciók);
```

  - vétel: `msgrcv(msgq_id, msg, méret, típus, opciók);`

  - a `típus` (egész szám) beállításával szűrést valósíthatunk meg
    - `= 0` a következő üzenet (tetszőleges típusú)
    - `> 0` a következő adott típusú üzenet
    - `< 0` a következő üzenet, amelynek a típusa kisebb vagy egyenlő

UNIX System V IPC: osztott memória

- Cél: folyamatok közötti egyszerű és gyors adatátvitel
  - a kernel helyett közvetlen adatátviteli csatorna (osztott memória régió)
  - a fizikai memória elkülönített része, amely közösen használható

- Használat

```
shm_id = shmget(kulcs, méret, opciók);
```

  - adott kulcsú osztott nyújt hozzáférést (adott opciókkal)
    (a tényleges létrehozás és az egyszerű hivatkozás az opciókban válik szét)

  - hozzárendelés saját virtuális címtartományhoz:

```
változó = (típus) shmat(...);
```

    az adott változót hozzákötjük a visszakapott címhez

  - lecsatolás: `shmdt(cím);`

  - a kölcsönös kizárást meg kell valósítani (pl. szemaforokkal)

UNIX „hálózati” (socket) kommunikáció

- Cél: címzéssel és protokollokkal támogatott adatátvitel
  - tetszőleges folyamatok között kliens – szerver architektúrában
  - többféle célra (folyamatok közötti és gépek közötti hálózati komm.)
  - sokféle protokollt támogat
  - többféle címzés

- Fogalmak
  - hálózati csatoló avagy azonosító (socket): a kommunikáció végpontja
  - IP cím és portszám (l. hálózatok)

- Használat

```
sfd = socket(domén, típus, protokoll);
szerver: bind(sfd, cím, ...);
kliens: connect(sfd, cím, ...);
szerver: listen(sfd, sor_mérete);
szerver: accept(sfd, cím, ...);
send(sfd, üzenet, ...);
recv(sfd, üzenet, ...);
shutdown(sfd);
```

Kliens és szerver programok váza

Kliens program

```
socket()

connect()

send()
recv()
close()
```

Szerver program

```
sfd1 = socket()
bind(sfd1)
listen(sfd1)
while
    sfd2 = accept(sfd1)
    fork()
```

szülő: vissza a ciklusba

gyerek:

```
recv(sfd2)
send(sfd2)
close(sfd2)
exit()
```

(Sun) RPC (távoli eljáráshívás)

- A socket kommunikációra épülő „elosztott rendszer” infrastruktúra
- Cél:
  - folyamatok közötti magas szintű kommunikáció
  - távoli eljárások meghívása (másik folyamatban, akár másik gépen)
  - programozói támogatás: interfész leírás + programgenerálás

- Fogalmak
  - RPC nyelv: a hívható eljárások és típusaik (interfész) leírása
  - azonosítók: a leírásban megadott egyedi számok (program, eljárás)
  - portmapper: a programazonosítók és a hálózati portok összerendelése
  - rpcgen: a leírásból C programkódot generáló program

- A Sun RPC technológia részei
  - interfész leírás
  - programgenerátor
  - kommunikációs infrastruktúra

RPC interfész leírás és programgenerátor

- RPC nyelv (példa: date.x)

```
program DATE_PROG {
    version DATE_VERS {
        long BIN_DATE(void) = 1;    /* eljárás azon. = 1 */
        string STR_DATE(long) = 2;  /* eljárás azon. = 2 */
    } = 1;                          /* verziószám = 1 */
} = 0x31234567;                     /* program azon. = 0x31234567 */
```

- Kódgenerátor: `rpcgen`
  - `rpcgen date.x` eredményei
    - `date.h`: adattípusok deklarációja
    - `date_clnt.c`: a kliens kódjában felhasználható date_... függvények
    - `date_srv.c`: a szerver date implementációját meghívó függvények
    - (…)

Összefoglalás: a folyamatok közötti kommunikáció

- Klasszikus kommunikációs formák:
  - Jelzések (signal)
  - Csővezetékek (pipe)
  - FIFO (elnevezett csővezeték, named pipe)

- System V IPC
  - Szemaforok
  - Üzenetsorok
  - Osztott memória

- (nem csak) Hálózati kommunikáció
  - hálózati csatoló (socket) kommunikáció
  - Sun RPC – távoli eljáráshívás UNIX módra