Prog1, C befejezés

Magasszintű programozási nyelvek 1
BSc előadás

Dr. Bátfai Norbert
egyetemi adjunktus
http://www.inf.unideb.hu/~nbatfai/
Debreceni Egyetem, Informatikai Kar,
Információ Technológia Tanszék
batfai.norbert@inf.unideb.hu
Skype: batfai.norbert

Prog1_3.ppt, v.: 0.0.9, 2011. 04. 17.
http://www.inf.unideb.hu/~nbatfai/#p1

Az óra blogja: http://progpater.blog.hu/
A Nokia Ovi store-ban is elérhető: http://store.ovi.com/content/100794

Felhasználási engedély

Bátfai Norbert
Debreceni Egyetem, Informatikai Kar, Információ Technológia Tanszék
<nbatfai@inf.unideb.hu, nbatfai gmail com>

Célok és tartalom

Előadás

a) Struktúrák, önhivatkozó struktúrák. Állománykezelés.
b) Bináris fák kezelése, Lempel-Ziv-Welch (LZW) algoritmus
c) GNU/Linux PCB, listakezelés

Labor

a) Binárisból karakteres „dump” írása
b) http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
c) LZW fa építése
   http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
   a Humán Genom Projekt kapcsán (a jó mérnöknél mindig van egy 2. kromoszóma)
   http://progpater.blog.hu/2011/02/27/a_human_genom_projekt
   http://progpater.blog.hu/2011/03/06/az_otodik_labor
d) „Saját top” parancs megírása, PP 89-
   http://www.inf.unideb.hu/~nbatfai/ProgramozoPaternoszter.pdf
e) A PP 173- oldal példáinak megbeszélése a laborvezetővel (kernel modulok, rendszerhívások).
f) Szálak, jelek bevezetése

Célok és tartalom

Laborkártyák

a) Struktúrás kártyák

Otthoni opcionális feladat

a) A japán világbajnok HELIOS csapat szoftvereinek otthoni tanulmányozása.

Kapcsoldó videók, videómagyarázatok és blogok

1) http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
2) http://progpater.blog.hu/2011/02/27/a_human_genom_projekt
3) http://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
4) http://progpater.blog.hu/2011/03/06/az_otodik_labor
5) http://progpater.blog.hu/2011/03/05/there_is_no_spoon

2012: http://progpater.blog.hu/2012/03/04/gyonyor_a_tomor_ujratoltve

Az írásbeli és a szóbeli vizsgán bármi (jegyzet, könyv, forráskód, számítógép mobiltelefon stb.) használható! (Az írásbeli vizsgán beszélni viszont tilos.) Hiszen az én feladatom az lesz, hogy eldöntsem, jól felkészült programozóval, vagy mennyire felkészült programozóval állok szemben.

Minimális gyakorlati cél

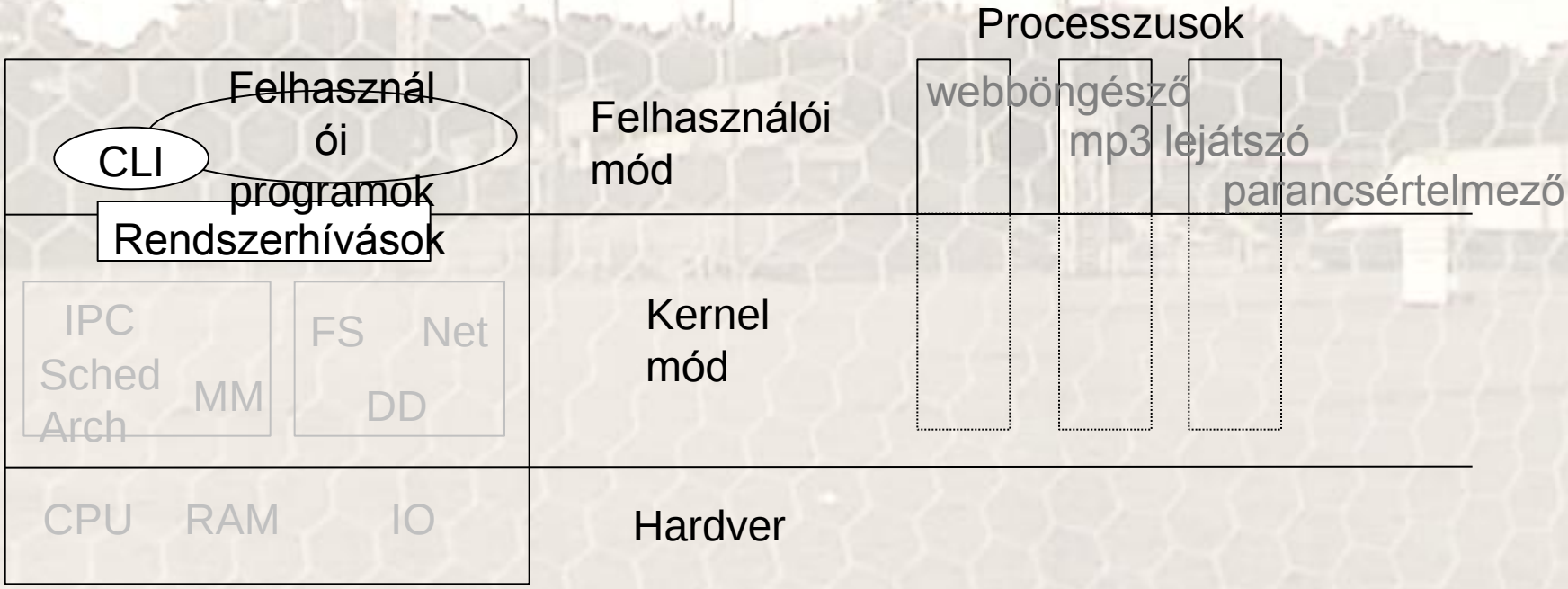
A hallgató meg tudjon írni (másolás alapján) egy egyszerű kernel modult, a rekurzív bináris fa nyomtatóba bele tudjon nyúlni (ki tudja például számolni az LZW fa águhosszainak szórását). Tudjon állományokat kezelni, konkrétan a PP alapján megírni a top parancs saját implementációját!

Minimális elméleti cél

1) C nyelv kapcsán: struktúrák, önhivatkozó struktúrák.
2) Bináris fa (pre, in, posztorder) bejárása
3) Állománykezelés kapcsán bevezetve: a GNU/Linux PCB néhány tagjának bemutatása (pl. task struct/pid, /files stb.)
4) Programkönyvtárak(statikus, megosztott), LD_LIBRARY_PATH, LD_PRELOAD
5) Rendszerhívások (néhány példával)
6) Splint kimenetének részleges ismerete (pl. a poloskak.c-re engedve)

Néhány szó az operációs rendszerről

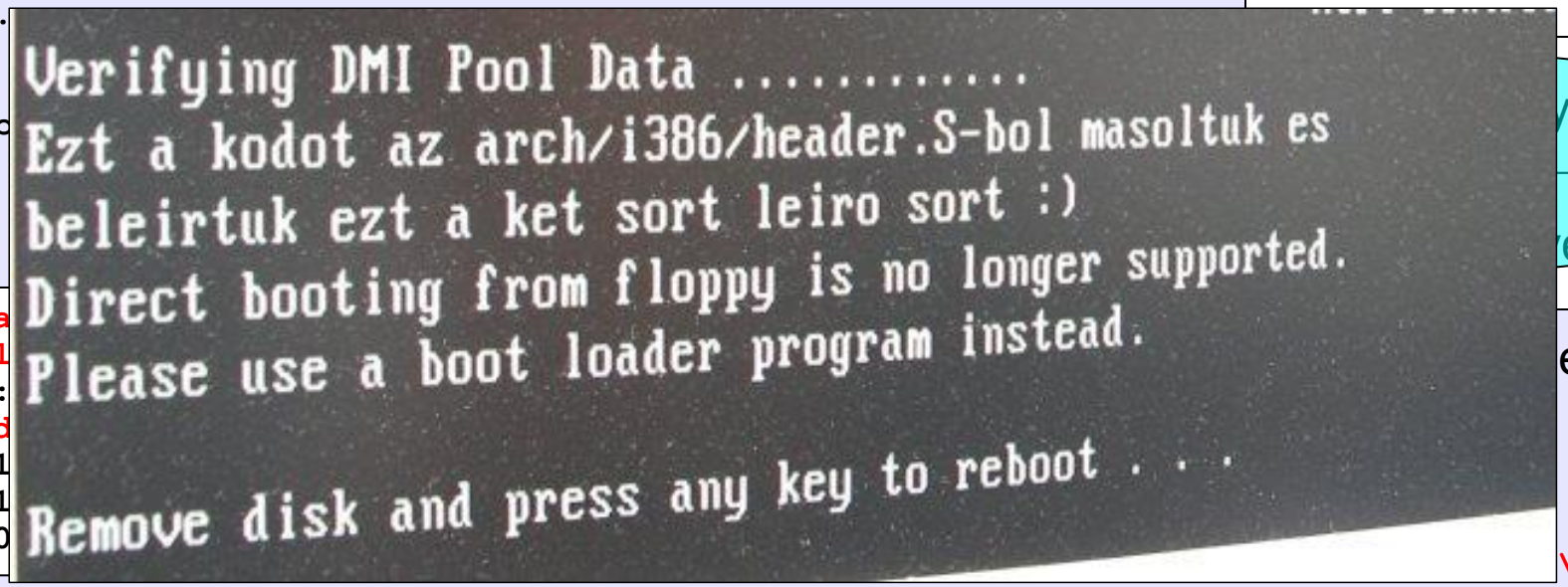
Az operációs rendszer - mint Mátrix - megteremti azt az álomvilágot, melyben vannak fájljaink, programjaink. Nélküle a *„The desert of the real.”* – a valóság sivatagában találjuk magunkat – ahogy Morpheus mondaná.

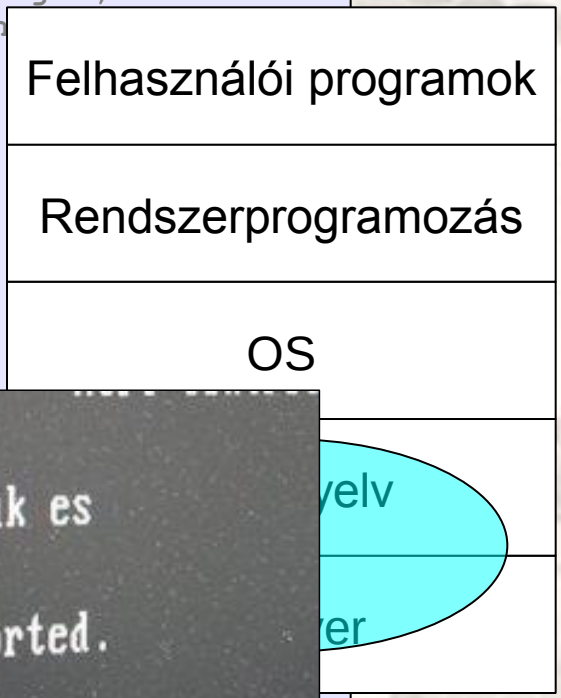


A programozó általánosításai: *„Javában minden objektum, UNIX-ban minden fájl, ...”*

Ha maradnál mégis a valóság sivatagában

```
arch/i386/boot/header.S-ból: (ez volt valóban, a 2.6.23.13 idején, amikor
készítettem az előadást, de alig két hét múlva a 2.6.24.2-ben
már az arch/x86/boot/header.S)
        .code16
        .global bootsect_start
bootsect_start:
        # Normalize the start address
        ljmp    $0x07c0, $start2
start2:
        movw    %cs, %ax
```





es rendszer

```
$ a
$ l
ld:
$ d
0+1
0+1
280
```

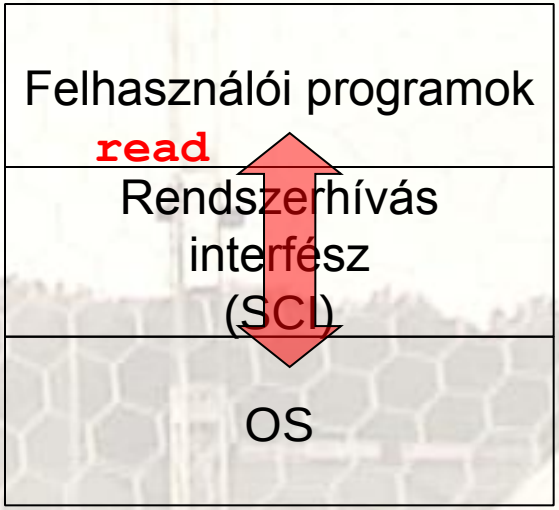
```
\r\n"
        .ascii      "beleirtuk ezt a ket sort leiro sort :)\r\n"
        .ascii      "Direct booting from floppy is no longer supported.\r\n"
        .ascii      "Please use a boot loader program instead.\r\n"
        .ascii      "\n"
        .ascii      "Remove disk and press any key to reboot . . .\r\n"
        .byte       0
```

Rendszerhívások

Számítógépes rendszer

Felhasználói mód

Kernel mód




```
olvasva = read (kapu, buffer, BUFFER_MERET)
```

billentyűzet fájl TCP socket stb.

(UNIX-ban minden fájl)

```
mennyit sikerült = olvasni (honnan, hova, mennyit kéne)
```

„Minden operációs rendszer lelke a megvalósított rendszerhívások készlete. Ezek határozzák meg az operációs rendszer tényleges tevékenységeit.„
Tanenbaum könyv, OR 63. oldal

OR 63

System Call Interface (SCI)

A kernel által nyújtott szolgáltatásokat rendszerhívásokkal vehetjük igénybe.

Programkönyvtárak

1) Statikus
2) Megosztott

   LD_LIBRARY_PATH, LD_PRELOAD

Glibc trófeák

Kisbajnokság: definiáld felül a glibc egy (változó argumentumszámú) függvényét!

```
$ gcc -shared -Wl,-soname,libsajat.so.1 -o libsajat.so.1.0 libsajat.c
$ ln -s libsajat.so.1.0 libsajat.so.1
$ ln -s libsajat.so.1 libsajat.so
$ export LD_PRELOAD=libsajat.so
$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
$ ./teszt
H 1 e 2 l 3 l 4 o
$
```

Ha gond lenne a fordítással:
http://tldp.fsf.hu/HOWTO/Program-Library-HOWTO-hu/shared-libraries.html

Glibc trófeák

libsajat.c

```
#include <errno.h>
#include <stdarg.h>

int
printf (char *fmt, ...)
{
  va_list ap;
  errno = 1;

  va_start (ap, fmt);
  putchar (0x1B);
  putchar ('[');
  putchar ('4');
  putchar ('7');
  putchar (';');
  putchar ('3');
  putchar ('1');
  putchar ('m');
  vprintf (fmt, ap);
  putchar (0x1B);
  putchar ('[');
  putchar ('0');
  putchar ('m');
  va_end (ap);

  return 0;
}
```

teszt.c

```
#include <stdio.h>

int
main (void)
{
  printf ("H %d e %d l %d l %d o\n", 1, 2, 3, 4);
}
```

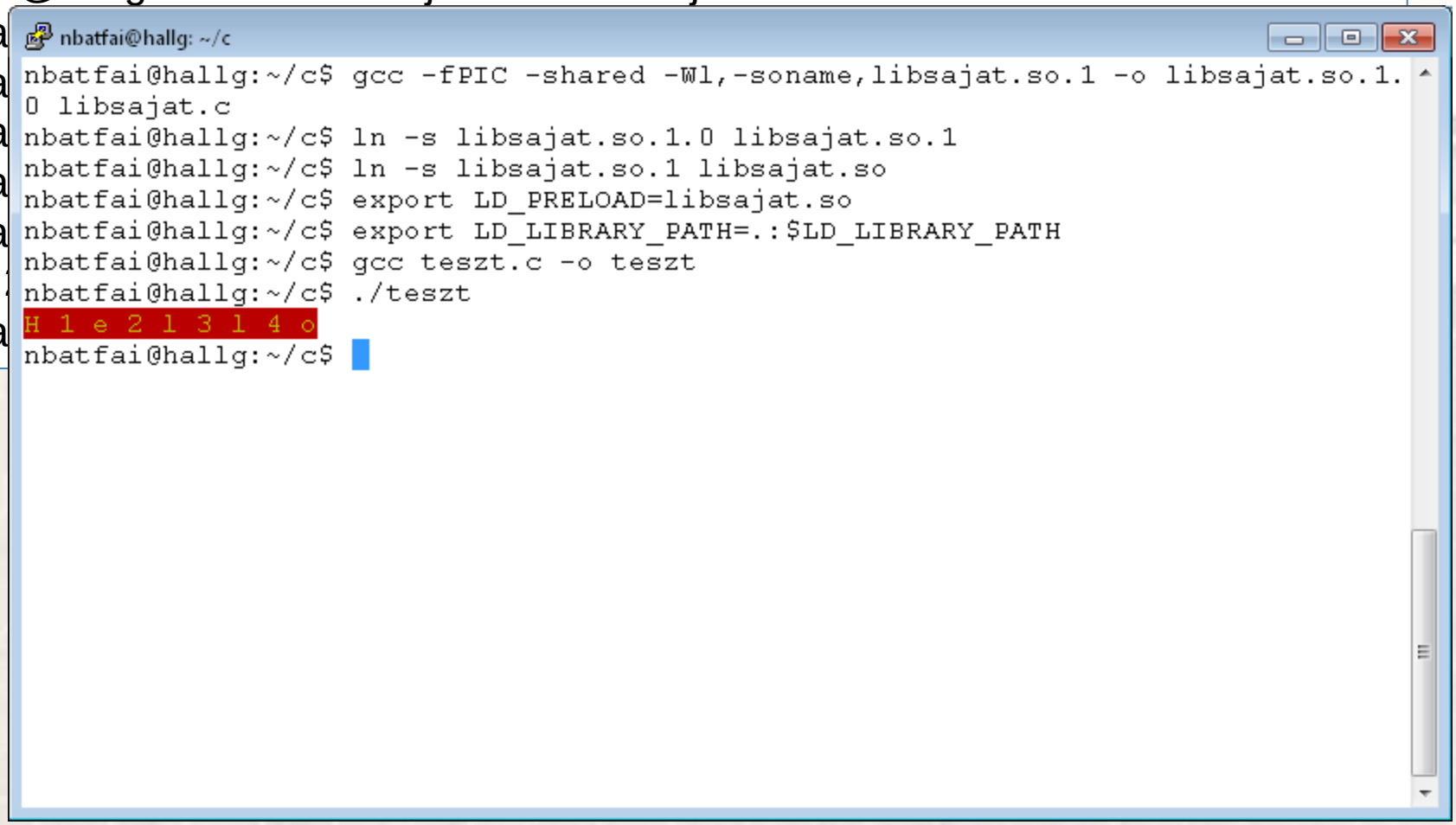
háttér- és szövegszín

ANSI Escape szekvenciák
http://en.wikipedia.org/wiki/ANSI_escape_code

visszaállítás

Glibc trófeák

nbatfai@hallg:~/c$ gcc -fPIC -shared -Wl,-soname,libsajat.so.1 -o libsajat.so.1.0 libsajat.c
nbatfai@hallg:~/c$ ln -s libsajat.so.1.0 libsajat.so.1
nbatfa
nbatfa
nbatfa
nbatfa
nbatfa
H 1 e
nbatfa



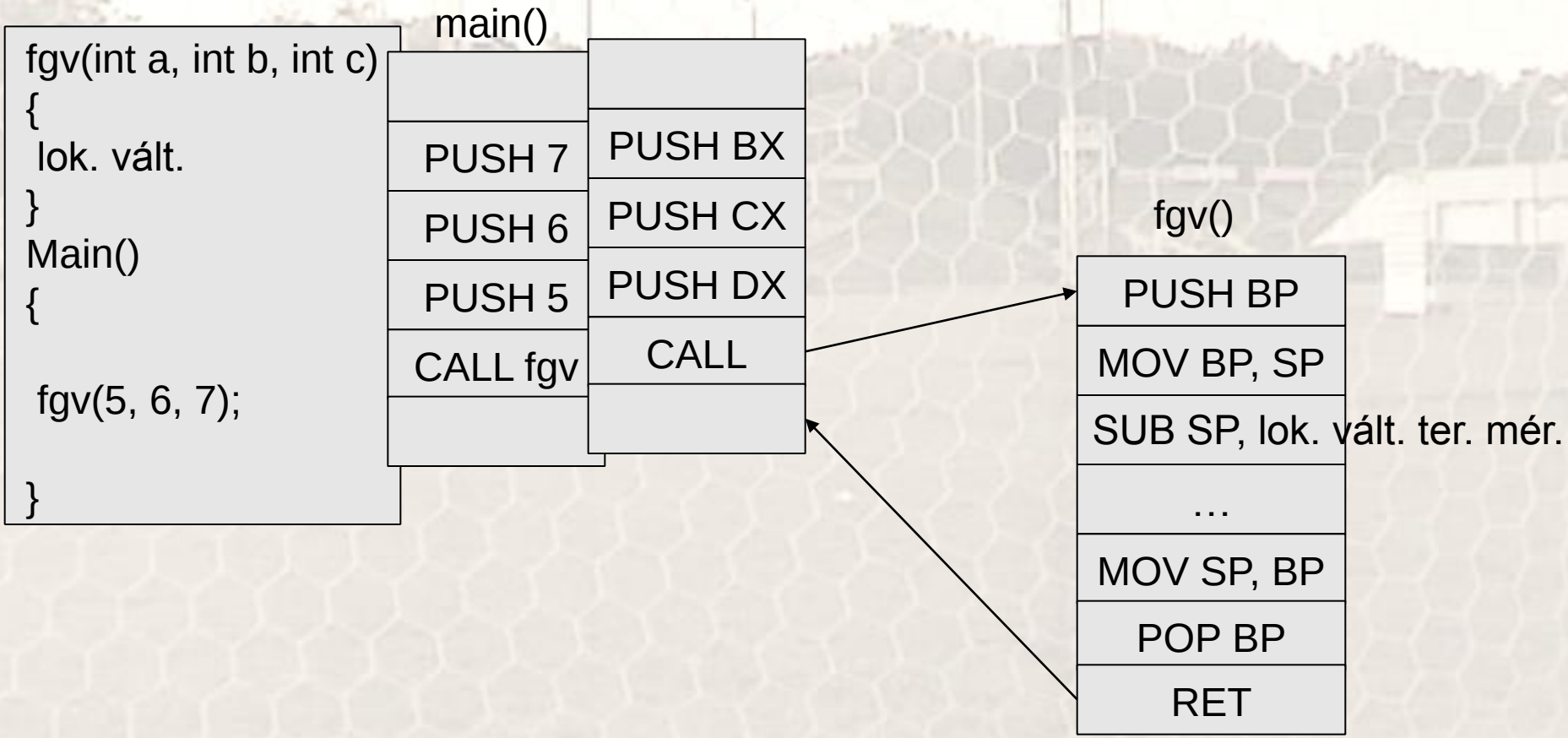
Változó argumentumszámú fgv-ek

```c
#include <stdio.h>
#include <stdarg.h>

void
sajatprintf (char *formatumsztring, .
{
  va_list arg_pointer;
  char *p = formatumsztring, c;

  va_start (arg_pointer, formatumsztr

  for (c=*p; *p; ++p)
    {
      if (*p == '%')
        {
          c = *++p;
          if (c == 'd')
            printf ("%d", va_arg (arg
          else if (c == 'f')
            printf ("%f", va_arg (arg
          else if (c == 'c')
            printf ("%c", va_arg (arg
        }
      else
        printf ("%c", *p);
    }
  va_end (arg_pointer);
}

int
main (void)
{
  int a = 5;
  double b = 5.5;
  char c = '*';

  sajatprintf ("%d, %c, %c, %f, %d\n", a, c, c, b, a);

  return 0;
}
```

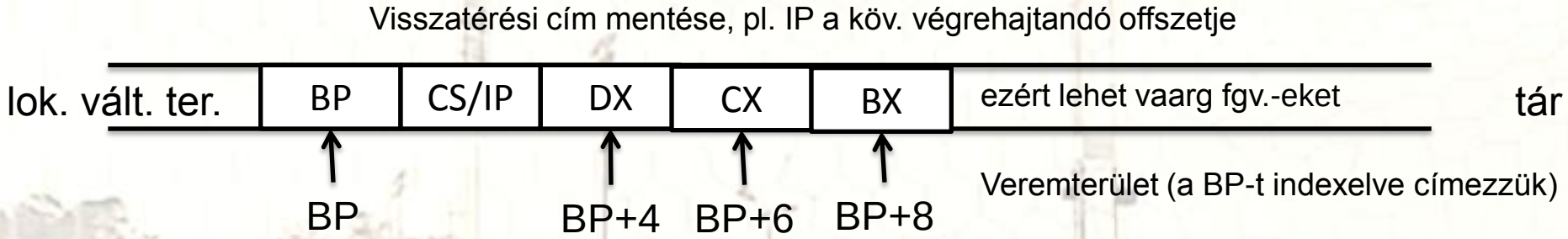
```c
sajatprintf (char *formatumsztring, ...)
{
  va_list arg_pointer;
  char *p = formatumsztring, c;

  va_start (arg_pointer, formatumsztring);

  for (c=*p; *p; ++p)
    {
      if (*p == '%')
        {
          c = *++p;
          if (c == 'd')
            printf ("%d", va_arg (arg_pointer, int));
          else if (c == 'f')
            printf ("%f", va_arg (arg_pointer, double));
          else if (c == 'c')
            printf ("%c", va_arg (arg_pointer, int));
        }
      else
        printf ("%c", *p);
    }
  va_end (arg_pointer);
```

http://progpater.blog.hu/2011/02/14/kisbajnoksagok

Fgv. hívás, paraméterátadás, lokális változók



Fgv. hívás, paraméterátadás, lokális változók




```
fgv(int a, int b, int c)
{
 lok. vált.
}
Main()
{

 fgv(5, 6, 7);

}
```

main()

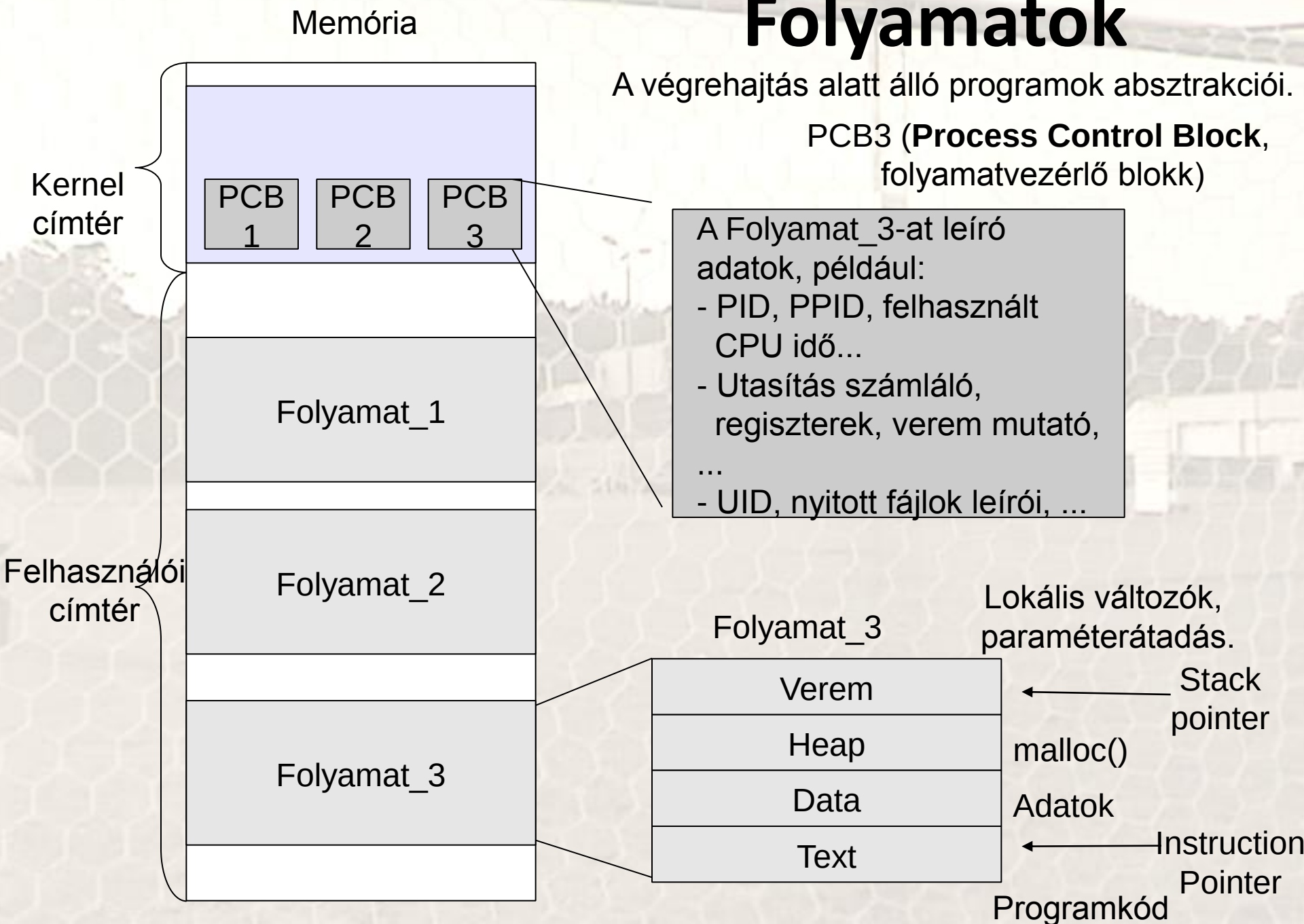
PUSH 7	PUSH BX
PUSH 6	PUSH CX
PUSH 5	PUSH DX
CALL fgv	CALL
	ADD SP, 3*(2)

fgv()

PUSH BP
MOV BP, SP
SUB SP, lok. vált. ter. mér.
…
MOV SP, BP
POP BP
RET

Folyamatok

A végrehajtás alatt álló programok absztrakciói.



A Linux kernel

Linus Torvalds, a Helsinki Egyetem 22 éves hallgatója 1991-ben az iskolában (Helsinki Egyetem) az Opreációs rendszerekből tanult MINIX-el való "elégedetlenségében" kezdett a Linux fejlesztésébe.

A Linus-Tanenbaum vita: LINUX is obsolete: (The Tanenbaum Torvalds Debate, Open Sources: Voices from the Open Source Revolution) http://www.oreilly.com/catalog/opensources/book/appa.html

„I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design :-)”

Letöltése

The Linux Kernel Archives: http://www.kernel.org/ (tipikusan az utolsó stabilt az „F” betű alól. Kb. 40 megás fájlt, de kitömörítve 200-300 mega, lefordítva pedig a gigát is meghaladhatja!)

A Linux kernel

„Linux is a clone of the operating system Unix, written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX and Single UNIX Specification compliance.

It has all the features you would expect in a modern fully-fledged Unix, including true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, proper memory management, and multistack networking inclu

It is distributed under the GNU

Linux kernel release 2.6.xx <http://kernel.org/>

Pillantsunk bele!

```
/*
 *  linux/kernel/printk.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 *
 * Modified to make sys_syslog() more flexible: added commands to
 * return the last 4k of kernel messages, regardless of whether
 * they've been read or not.  Added option to suppress kernel printk's
 * to the console.  Added hook for sending the console messages
                                    e console (someday).

                                    s Horn.
                                    Manfred Spraul


                                    wm@uow.edu.au>
```

„...Linux powers everything from supercomputers to mobile phones around the world, and Torvalds has achieved fame as the godfather of the open-source movement, ...”

http://www.time.com/time/europe/hero2006/torvalds.html
Linus Torvalds a TIME Forradalmárok és Vezetők ketegóriájának **hőse**.

Hol használják?

Motorola A780, E680 mobiltelefonok kernel/pkgs:
https://opensource.motorola.com/sf/frs/do/viewSummary/projects.a780e680/frs
Download E680 / E680i / A780 Kernel sources: http://sourceforge.net/projects/e680/
opensource.motorola.com: https://opensource.motorola.com

Linuxok a Google keresés mögött álló szerverei.

Google Android - An Open Handset Alliance Project:
http://code.google.com/android

maemo.org: Maemo is the application development platform for Internet Tablets:
http://maemo.org/
Pl.: Nokia N810: http://www.forum.n

```
/*
 *  linux/kernel/printk.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 *              2004 Motorola
 *
 * Modified to make sys_syslog() more flexible: added commands to
 * return the last 4k of kernel messages, regardless of whether
 * they've been read or not.  Added option to suppress kernel printk's
 * to the console.  Added hook for sending the console messages
 * elsewhere, in preparation for a serial line console (someday).
 * Ted Ts'o, 2/11/93.
 * Modified for sysctl support, 1/8/97, Chris Horn.
 * Fixed SMP synchronization, 08/08/99, Manfred Spraul
 *     manfreds@colorfullife.com
 * Rewrote bits to get rid of console_lock
 *           01Mar01 Andrew Morton <andrewm@uow.edu.au>
 *
 * 2004-Aug-4 - (Motorola) Added changes to enable phone logging
 */

#include <linux/kernel.h>
...
```

Kitekintés-visszatekintés

Table 2
Worldwide Smartphone Sales to End Users by Operating System in 2010 (Thousands of Units)

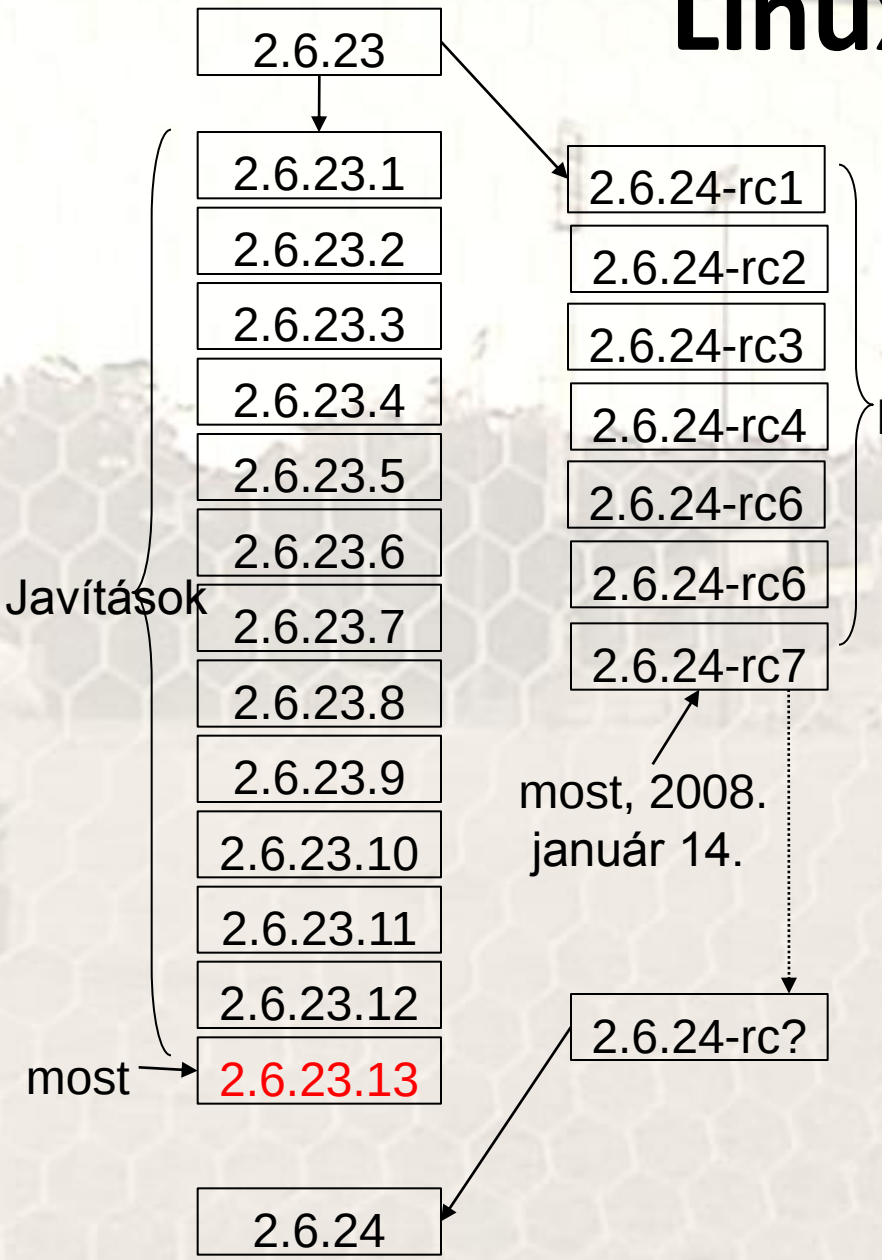
Company	2010 Units	2010 Market Share (%)	2009 Units	2009 Market Share (%)
Symbian	111,576.7	37.6	80,878.3	46.9
Android	67,224.5	22.7	6,798.4	3.9
Research In Motion	47,451.6	16.0	34,346.6	19.9
iOS	46,598.3	15.7	24,889.7	14.4
Microsoft	12,378.2	4.2	15,031.0	8.7
Other Oss	11417.4	3.8	10432.1	6.1
Total	**296,646.6**	**100.0**	**172,376.1**	**100.0**

Source: Gartner (February 2011)

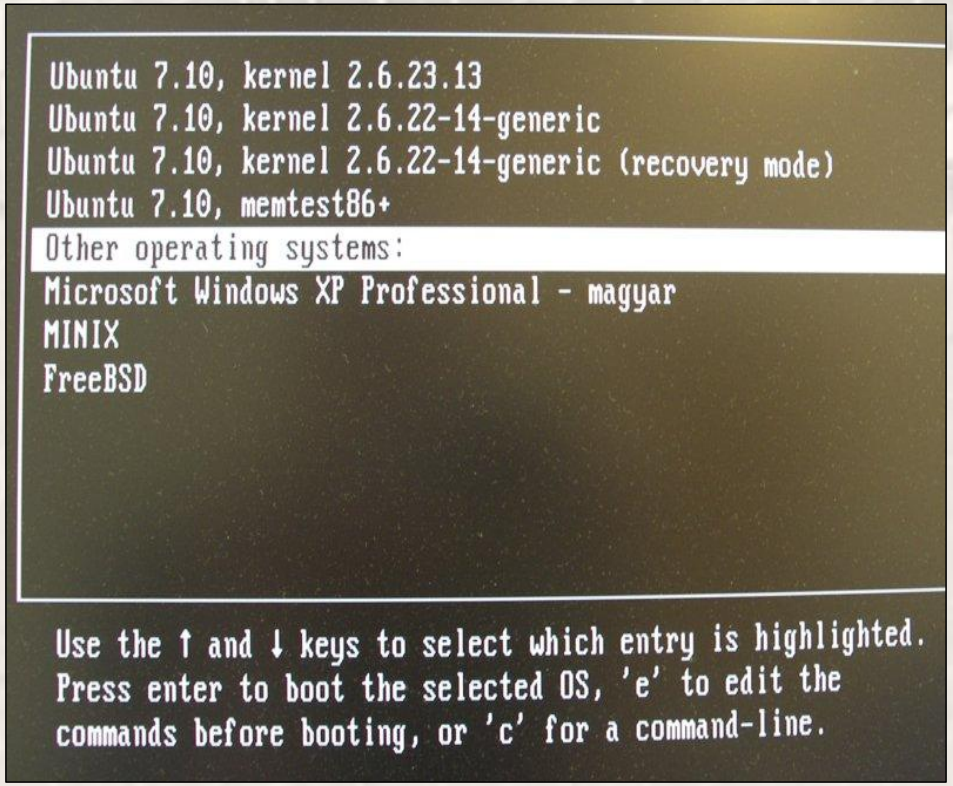
Ábra forrása:
http://www.gartner.com/it/page.jsp?id=1543014

http://progpater.blog.hu/2011/02/13/az_ero_0x333333_oldalan

Linux kernel verziók

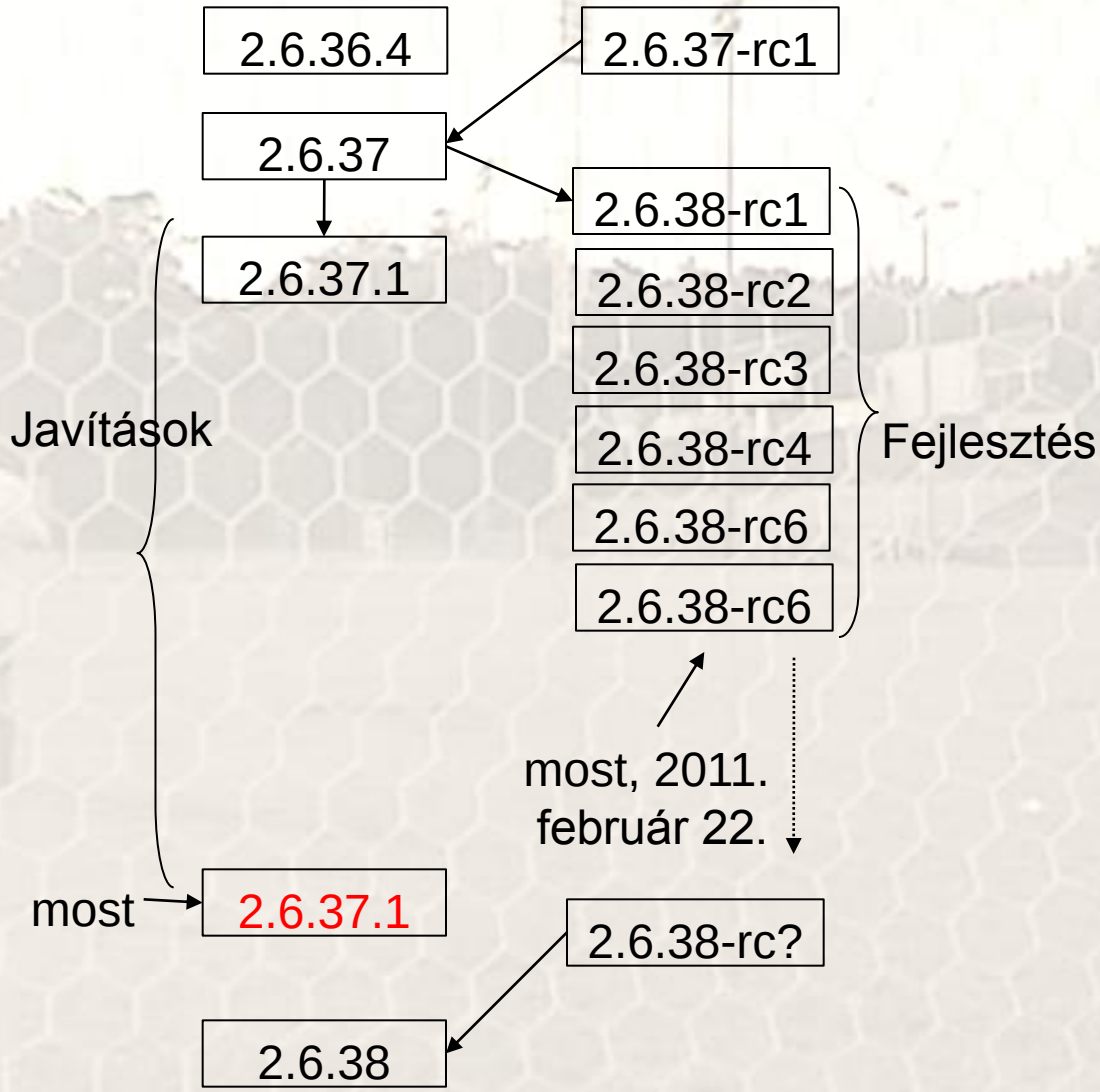


The Linux Kernel Archives:
http://www.kernel.org/
(a stabil akkor éppen 2.6.23.13)



Major.Minor.Revision.Patchlevel

Linux kernel verziók



The Linux Kernel Archives:
http://www.kernel.org/
(a stabil most 2.6.36.4, 2.6.37.1)

Fejlesztési ciklus (nap)

2.6.23	2007-10-09	94
2.6.24	2008-01-24	108
2.6.25	2008-04-16	83
2.6.26	2008-07-13	88
2.6.27	2008-10-09	88
2.6.28	2008-12-24	76
2.6.29	2009-03-23	89
2.6.30	2009-06-09	78
2.6.31	2009-09-09	92
2.6.32	2009-12-02	84
2.6.33	2010-02-24	84
2.6.34	2010-05-15	81
2.6.35	2010-08-01	77

http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf

Fejlesztési ciklus (fájl, sor)

2.6.23	22,530	8,566,606
2.6.24	23,062	8,859,683
2.6.25	23,813	9,232,592
2.6.26	24,273	9,411,841
2.6.27	24,356	9,630,074
2.6.28	25,276	10,118,757
2.6.29	26,702	10,934,554
2.6.30	27,911	11,560,971
2.6.31	29,143	11,970,124
2.6.32	30,504	12,532,677
2.6.33	31,584	12,912,684
2.6.34	32,316	13,243,582
2.6.35	33,335	13,468,253

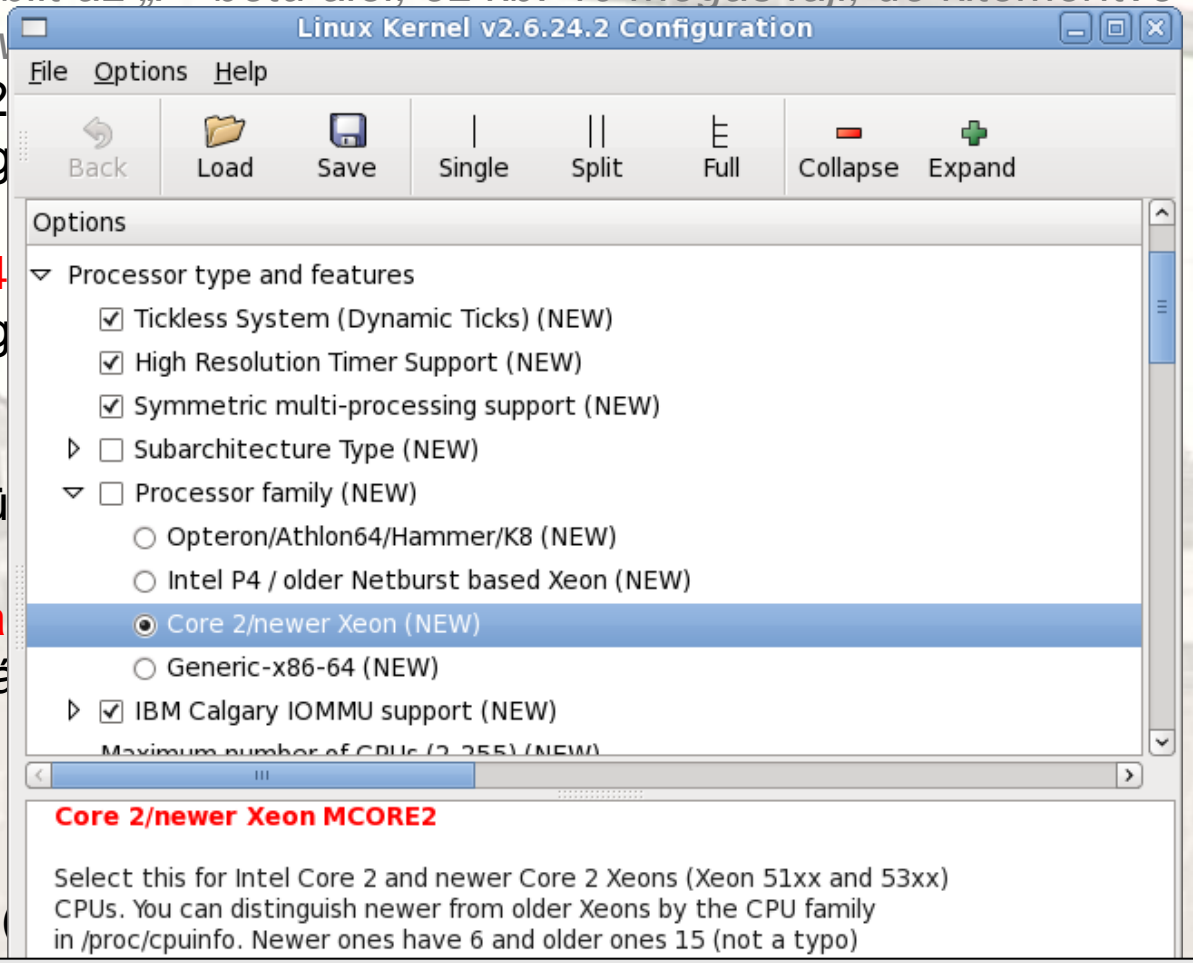
http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf

Fejlesztési ciklus (naponta hozzáadott, törölt, módosított sorok)

2.6.23	3,747	3,034	1,343
2.6.24	6,893	4,181	1,563
2.6.25	7,980	3,488	2,430
2.6.26	5,698	3,662	1,815
2.6.27	12,270	9,791	2,102
2.6.28	12,105	5,707	1,850
2.6.29	14,678	5,516	2,454
2.6.30	12,993	4,958	2,830
2.6.31	9,408	4,962	1,635
2.6.32	12,086	5,388	2,387
2.6.33	8,925	4,379	2,841
2.6.34	6,667	2,580	1,568
2.6.35	7,896	5,037	1,802

http://www.linuxfoundation.org/docs/lf_linux_kernel_development_2010.pdf

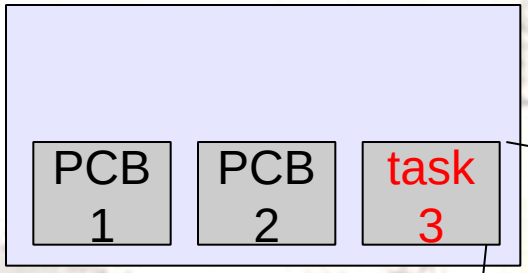
- 1.) A kernel forrásainak letöltése: *The Linux Kernel Archives:* http://www.kernel.org/ *(tipikusan az utolsó stabilt az „F” betű alól, ez kb. 40 megás fájl, de kitömörítve 200-300 mega, lefordít*
  /home/norbi/Kernelek/2
- 2.) A források kicsomag
  $ cd Kernelek/2.6
  $ bzip2 -cd linux-2.6.24
- 3.) Belépek a kicsomag
  $ cd linux-2.6.24.2
  $ make mrproper
- 4.) Kiindulásnak szerzü
  könyvtárba:
  $ cp /usr/src/kernels/va
- 5.) A kernel bekonfigurá
  $ make gconfig
- 6.) A kernel lefordítása:
  $ make
  … dolgozik a C fordító
- 7.)
  $ s
  Pas
  # m
- 8.)

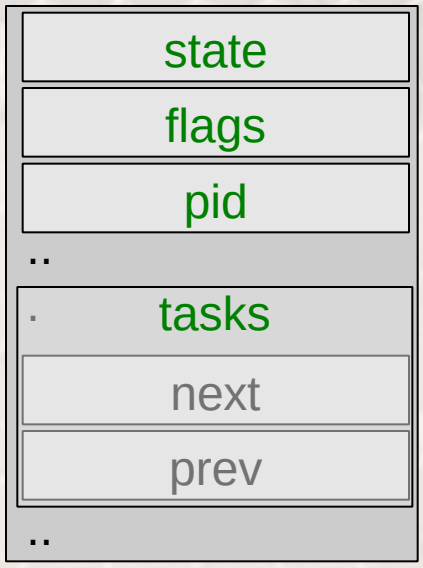


Kernelfordítás: a Programozó Páternoszter 173. oldalán találsz segítséget, illetve a Linux Kernel in a Nutshell című könyvet ajánlhatjuk: http://www.kroah.com/lkn/, http://www.kernel.org/pub/linux/kernel/people/gregkh/lkn/lkn_pdf/

Részletesebb infók: /home/norbi/Kernelek/2.6/linux-2.6.24.2/README

Linux terminológiában: task





include/linux/sched.h

```
struct task_struct {
        volatile long state;/* -1 unrunnable, 0 runnable, >0 stopped *
        ...
        unsigned int flags; /* per process flags, defined below */
...
        int prio, static_prio, normal_prio;
...
        pid_t pid;
...
        struct list_head tasks;
...
        struct files_struct *files;
...
```

A Linux PCB: task_struct

include/linux/sched.h

```
struct task_struct {
        volatile long state;/* -1 unrunnable, 0 runnable, >0 stopped */
        ...
        unsigned int flags; /* per proc
...
        int prio, static_prio, normal_p
...
        pid_t pid;
...
        struct files_struct *files;
...
```

```
/*
 * Task state bitmask. ...
 */
#define TASK_RUNNING                    0
#define TASK_INTERRUPTIBLE 1
#define TASK_UNINTERRUPTIBLE      2
#define TASK_STOPPED                    4
#define TASK_TRACED                     8
/* in tsk->exit_state */
#define EXIT_ZOMBIE                     16
#define EXIT_DEAD                       32
                                        gain */
```

include/linux/file.h

```
/*
 * Open file table structure
 */
struct files_struct {
...
        ... struct file * fd_array[NR_OPEN_DEFAULT];
```

(ennek a file*-okból álló tömbnek az indexei a fájl leírók: a 0, az 1 és a 2, továbbá azok a ***misztikus kis egész számok a fájlkezelésnél***)

A Linux processz állapotai

```
fs/proc/array.c
/*
 * The task state array is a strange "bitmap" of
 * reasons to sleep. Thus "running" is zero, and
 * you can test for combinations of others with
 * simple bit tests.
 */
static const char *task_state_array[] = {
        "R (running)",              /*  0 */
        "S (sleeping)",             /*  1 */
        "D (disk sleep)",   /*  2 */
        "T (stopped)",              /*  4 */
        "T (tracing stop)", /*  8 */
        "Z (zombie)",       /* 16 */
        "X (dead)"                  /* 32 */
};

static inline const char *get_task_state(struct task_struct *tsk)
{
```

```
$ ps
  PID TTY          TIME CMD
 6222 pts/1    00:00:00 bash
 6238 pts/1    00:00:00 ps
$ more /proc/6222/stat
6222 (bash) S 5921 6222 6222 34817 6239 4194304 1295 3477 1 6 38 4 23 5 15 0 1 0
 2139215 5783552 762 4294967295 134512640 135193680 3217577904 3217576728 429496
0144 0 65536 3686404 1266761467 3222449910 0 0 17 0 0 0 0
```

```
        while (state) {
                p++;
                state >>= 1;
        }
        return *p;
}
```

Az OpenSolaris PCB: proc

http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/sys/proc.h

```
19  * CDDL HEADER END
20  */
21
22 /*
23  * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
24  */
25
26 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
27 /*        All Rights Reserved   */
```

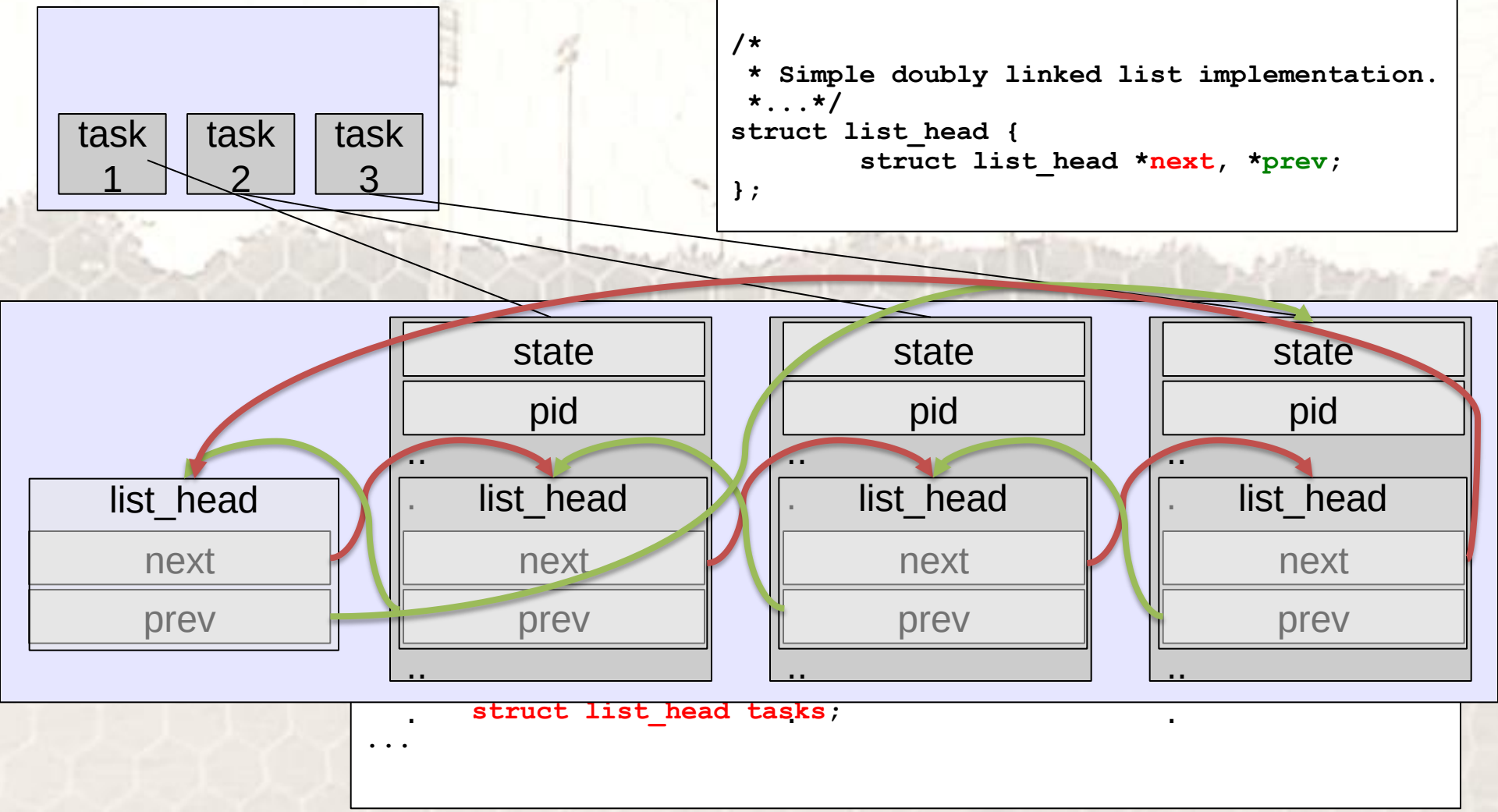
```
/*
 * One structure allocated per active process.  It contains all
 * data needed about the process while the process may be swapped
 * out.  Other per-process data (user.h) is also inside the proc structure.
 * Lightweight-process data (lwp.h) and the kernel stack may be swapped out.
 */
typedef struct proc {
        /*
```

A MINIX PCB: proc

```
File Edit Options Buffers Tools C Help
#ifndef PROC_H
#define PROC_H

/* Here is the declaration of the process table.  It contains all process
 * data, including registers, flags, scheduling priority, memory map,
 * accounting, message passing (IPC) information, and so on.
 *
 * Many assembly code routines reference fields in it.  The offsets to these
 * fields are defined in the assembler include file sconst.h.  When changing
 * struct proc, be sure to change sconst.h to match.
 */
#include <minix/com.h>
#include "protect.h"
#include "const.h"
#include "priv.h"

struct proc {

  struct stackframe_s p_reg;    /* process' registers saved in stack frame */

#if (CHIP == INTEL)
  reg_t p_ldt_sel;              /* selector in gdt with ldt base and limit */
-----:---F1  proc.h          (C Abbrev)--L1--Top-------------------------------
Wrote /usr/src/kernel/proc.h
```

A Linux PCB részletesebben

```
include/linux/list.h

/*
 * Simple doubly linked list implementation.
 *...*/
struct list_head {
        struct list_head *next, *prev;
};
```




```
    .    struct list_head tasks;                    .
...
```

A Linux PCB részletesebben:

include/linux/list.h

```
/**
  * list_for_each        -       iterate over a list
  * @pos:         the &struct list_head to use as a loop cursor.
  * @head:        the head for your list.
  */
 #define list_for_each(pos, head) \
        for (pos = (head)->next; prefetch(pos->next), pos != (head); \
                pos = pos->next)
```

```
#include <linux/
#include <linux/sched.h>
#include <linux/list.h>
asmlinkage long
sys_norbi ()
{
  struct list_head *p;
  int i = 0;
  list_for_each (p, current->tasks.next)++ i;
  printk (KERN_NOTICE "norbi a kernelben: %d folyamatot szamoltam.\n", i);
  return i;
}
```

PP 180

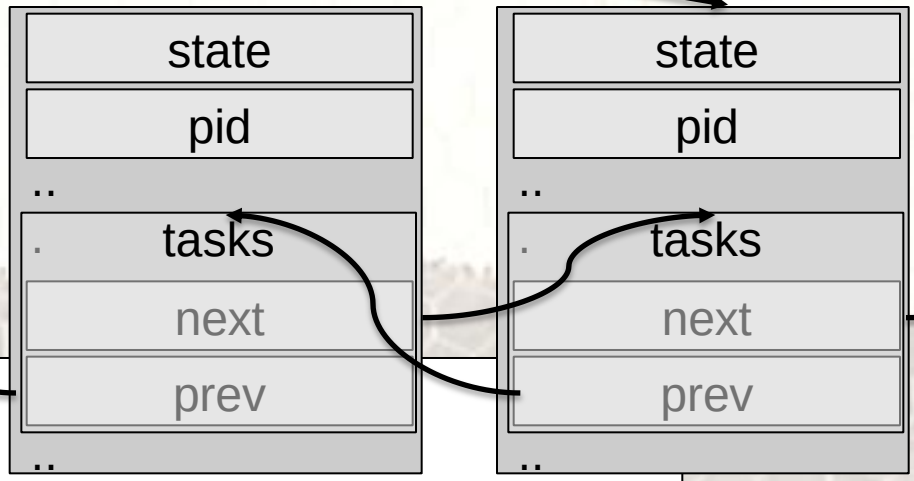
```
static int
harmadik_init_module (void)
{
 struct task_struct *task;
 struct list_head *p;
 list_for_each (p, current->tasks.next) {
   task = list_entry (p, struct task_struct, tasks);
   printk (KERN_NOTICE "%s %i %lX %li %lu %lu\n", task->comm,
           task->pid, task->flags, task->state, task->sleep_avg,
           task->policy);
 }
 return 0;
}
```

PP 177

A Linux PCB részletesebben

a current makró

```
struct task_struct * task = list_entry (p, struct task_struct,
tasks);
```



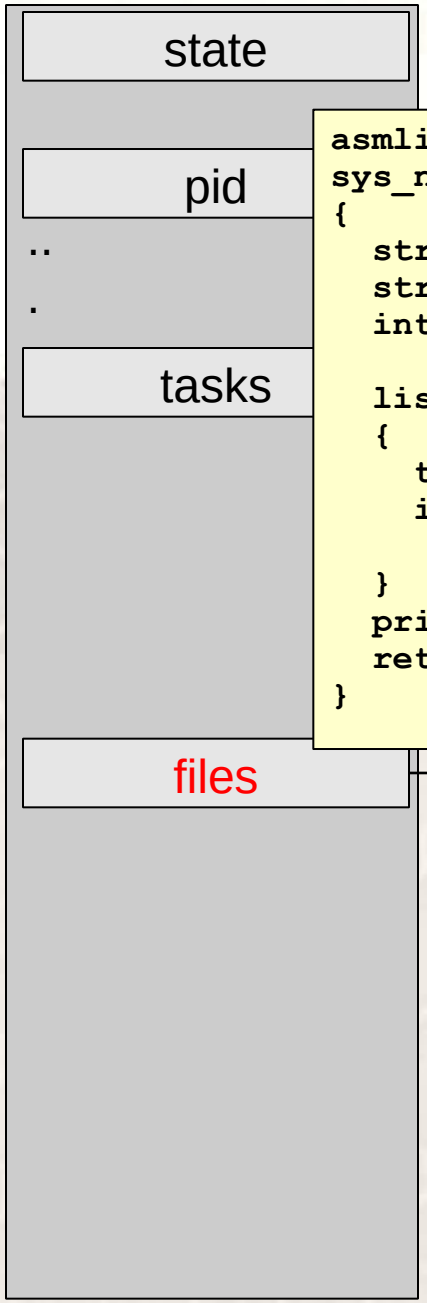
include/linux/list.h

```
/**
  * list_entry - get the struct for this entry
  * @ptr:        the &struct list_head pointer.
  * @type:       the type of the struct this is embedded in.
  * @member:     the name of the list_struct within the struct.
  */
 #define list_entry(ptr, type, member) \
         container_of(ptr, type, member)
```

PP 177

```
 struct task_struct *task;
 struct list_head *p;
 list_for_each (p, current->tasks.next) {
   task = list_entry (p, struct task_struct, tasks);
   printk (KERN_NOTICE "%s %i %lX %li %lu %lu\n", task->comm,
           task->pid, task->flags, task->state, task->sleep_avg,
           task->policy);
 }
 return 0;
}
```

norbi()-ból norbi2()



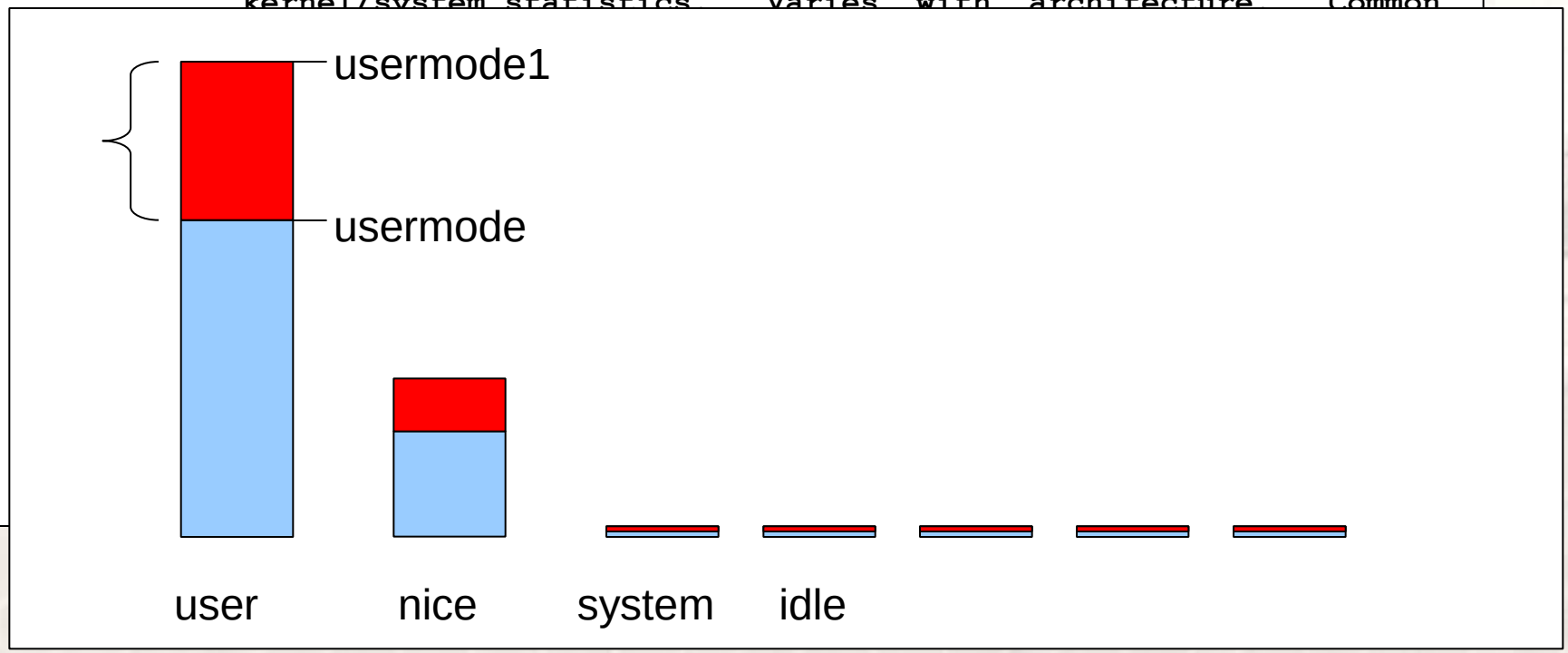

```
asmlinkage long
sys_norbi2 (int pid)
{
  struct list_head *p;
  struct task_struct *task;
  int i = 0;

  list_for_each (p, current->tasks.next)
  {
    task = list_entry (p, struct task_struct, tasks);
    if (pid == task->pid)
      i = task->files->next_fd;
  }
  printk (KERN_INFO "norbi a kernelben: nyitott fajlok szama: %i\n", i);
  return i;
}
```

http://progpater.blog.hu/2011/02/14/kisbajnoksagok

A /proc fájlrendszer

MM 343

```
PROC(5)                    Linux Programmer's Manual                    PROC(5)
...
       /proc/stat
              kernel/system statistics.  Varies with architecture.  Common
```



http://progpater.blog.hu/2011/02/14/kisbajnoksagok

A /proc fájlrendszer

```
PROC(5)                    Linux Programmer's Manual                    PROC(5)

NAME
       proc - process information pseudo-filesystem

DESCRIPTION
       The  proc  filesystem is a pseudo-filesystem which is used as an inter-
       face to kernel data structures. It is commonly mounted at /proc.  Most
       of  it  is  read-only,  but  some  files  allow  kernel variables to be
       changed.

       The following outline gives a quick tour through the /proc hierarchy.

       /proc/[number]
              There is a numerical subdirectory for each running process;  the
              subdirectory is named by the process ID.  Each such subdirectory
              contains the following pseudo-files and directories.
```

PP 177

Bővítsük saját rendszerhívással Linux rendszerünket!

norbi megvaló-sítása

```
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/list.h>
asmlinkage long
sys_norbi ()
{
  struct list_head *p;
  int i = 0;
  list_for_each (p, current->tasks.next)++ i;
  printk (KERN_NOTICE "norbi a kernelben: %d folyamatot szamoltam.\n", i);
```

Ez kernelbeli adatszerkezet!

```
$ gcc -o norbi norbi.c
$ ./norbi
[  768.235026] norbi a kernelben: 85 folyamatot szamoltam.
Norbi mondja: 85 folyamatot szamolt.
```

norbi meghí-vása

```
#include <sys/norbi.h>
int
main ()
{
  long n = norbi ();
  printf ("Norbi mondja: %ld folyamatot szamolt.\n", n);
  return 0;
}
```

PP 180

Saját rendszerhívás…

Kis segítség mert a Páternoszterbeli példa a 2.6.18 kerneltől elavult részeket is tartalmaz!

```
$ man 2 intro

INTRO(2)                  Linux Programmer's Manual                 INTRO(2)

NAME
       intro, _syscall - Introduction to system calls

DESCRIPTION
       This chapter describes the Linux system calls.  For a list of the Linux
       system calls, see syscalls(2).

   Calling Directly
       In most cases, it is unnecessary to invoke a system call directly,  but
       there  are  times when the Standard C library does not implement a nice
       function call for you.  In this  case,  the  programmer  must  manually
       invoke  the  system call using syscall(2).  Historically, this was also
       possible using one of the _syscall macros described below.
...
NOTES
       Starting around kernel 2.6.18, the _syscall macros were removed from
       header files supplied to user space.  Use  syscall(2)  instead.
```

```
$ joe norbi.c
#include <linux/unistd.h>
#include <sys/syscall.h>

static inline int
norbi ()
{
  return syscall (__NR_norbi);
}
```

```
$ gcc -o norbi norbi.c
$ ./norbi
Norbi mondja: 139 folyamatot szamolt.
$ gcc -o norbi norbi2.c
$ ./norbi
Norbi mondja: 139 folyamatot szamolt.
```

PP 180

Bővítsük saját kernelmodullal Linux rendszerünket!

PP 177

```
$ joe negyedik.c
...
MODULE_DESCRIPTION ("Ez a negyedik kernel modulom");
MODULE_AUTHOR ("Batfai Norbert (nbatfai@gmail.com)");
MODULE_LICENSE ("GPL");
static int
taszk_lista_show (struct seq_file *m, void *v)
{
  int i = 0;
  struct task_st
  struct list_he
  seq_puts (m, "
            "SZAM
  list_for_each
  {
    task = list_
    seq_printf (
                ta
  }
  return 0;
}
...
$ joe Makefile
obj-m += negyedi
all:
        make -C /
clean:
        make -C /lib/modules/`una
        rm *~
```

```
$ make
make -C /lib/modules/`uname -r`/build M=`pwd` modules
make[1]: Entering directory `/usr/src/kernels/2.6.23.14-107.fc8-x86_64'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/kernels/2.6.23.14-107.fc8-x86_64'
# /sbin/insmod negyedik.ko
# more /proc/sajat/taszk_stat
sajat taszk lista (negyedik kernel modul)
SZAMOZAS PARANCSSOR         PID     FLAGS     STATE
1        init               1       00400100  1
...
132      bash               6681    00400000  1
133      more               6904    00400000  0
# /sbin/rmmod negyedik.ko
```

linux-.../Documentation/kbuild/mobules.txt

http://progpater.blog.hu/2011/02/14/kisbajnoksagok

C állománykezelés

1) Karakteres/bináris

2) Magas/alacsony

V.1.2.1.1 C

```
#include <stdio.h>
#include <math.h>
int
main()
{
  FILE *gyok2f;
  double gy2;
  gyok2f = fopen("GyokKetto.txt", "r");
  fscanf(gyok2f, "%lf", &gy2);
  fclose(gyok2f);
  printf("%lf\n", gy2);
  return 0;
}
```

", "w");

étrehozott fájlba:

Ezt is próbáljuk ki:

```
$ gcc -o gyokbe gyokbe.c
$ ./gyokbe
1.414214
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#inclu
#inclu
int
main()
{
   int
   doub
   gyok
   writ
   clos
   retu
}
```

Fordítsuk

```
$ gcc -
$ ./gyo
$ more
Íf ö?
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
main()
{
   int gyok2f;
   double gy2;
   gyok2f = open("GyokKetto", O_RDONLY);
   read(gyok2f, (void *)&gy2, sizeof(double));
   close(gyok2f);
   printf("%lf\n", gy2);
   return 0;
}
```

Sikerül visszaolvasni?

```
$ gcc -o gyokbeb gyokbeb.c
$ ./gyokbeb
1.414214
```

C struktúrák

Már nem ismeretlenek…

include/linux/sched.h

```
struct task_struct {
        volatile long state;/* -1 unrunnable, 0 runnable, >0 stopped */
        ...
        unsigned int flags; /* per process flags, defined below */
...
        int prio, static_prio, normal_prio;
...
        pid_t pid;
...
        struct list_head tasks;
...
        struct files_struct *files;
...
```

C struktúrák

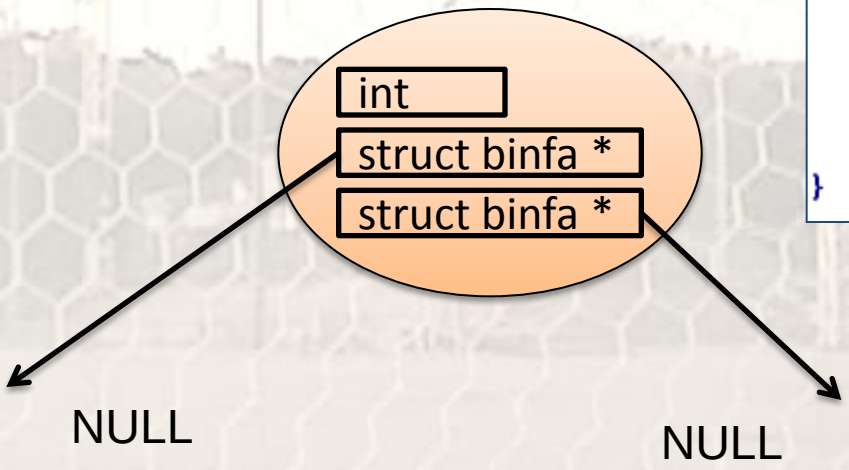
struct címke{…} a; <-> int a;
struct címke a; <-> int a;

a.

struct címke *ap; <-> int *ap;

(*ap). mert a * gyengébb precijű
ap->

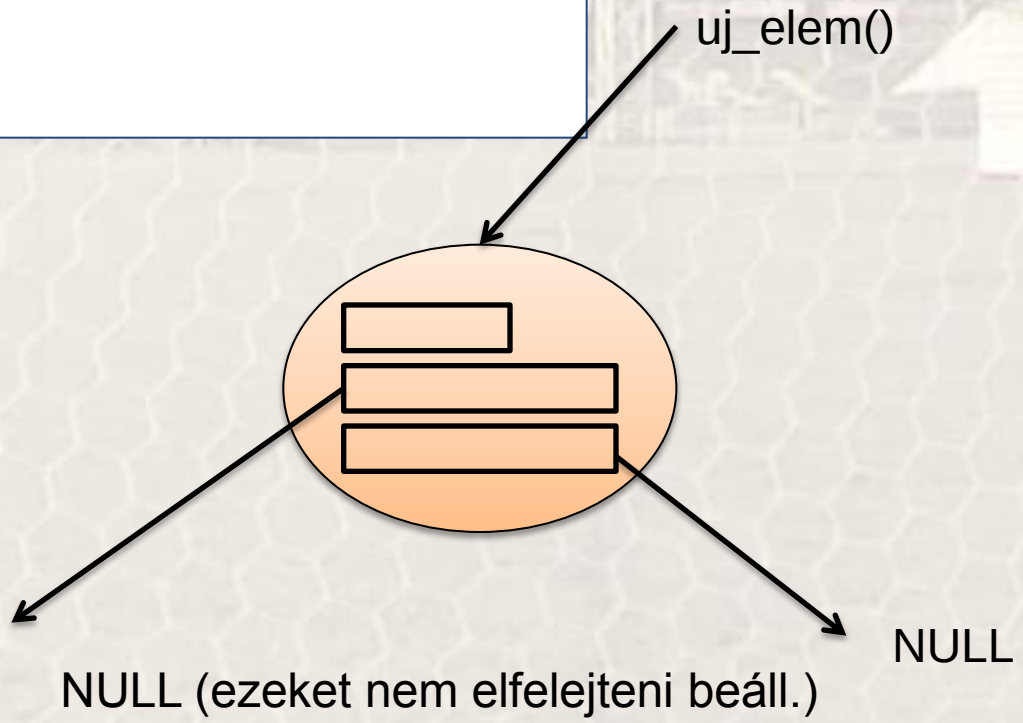
C önhivatkozó struktúrák




```
typedef struct binfa
{
  int ertek;
  struct binfa *bal_nulla;
  struct binfa *jobb_egy;

} BINFA, *BINFA_PTR;
```

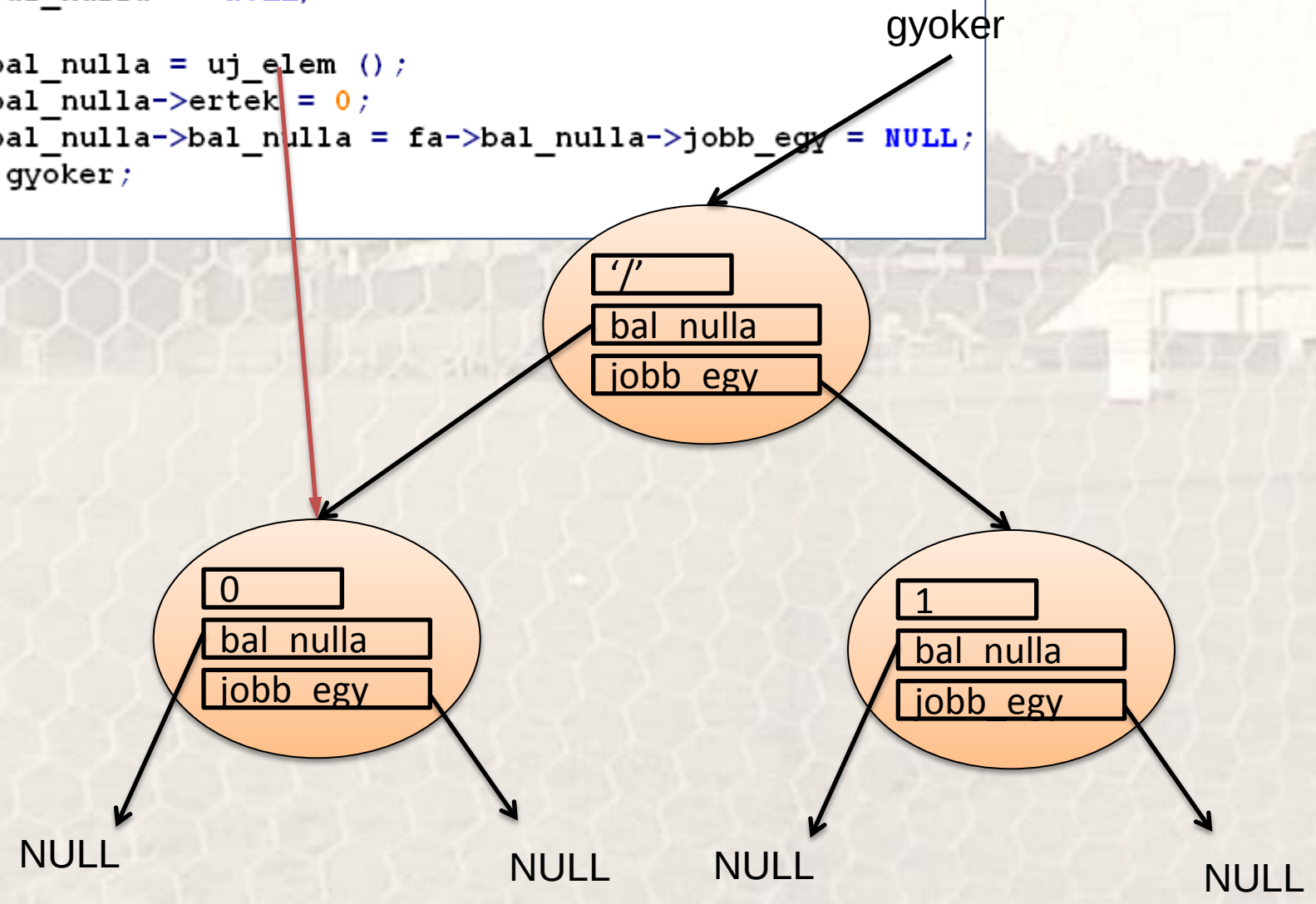
C önhivatkozó struktúrák

```
BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

  if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
      perror ("memoria");
      exit (EXIT_FAILURE);
    }
  return p;
}
```



C önhivatkozó struktúrák

```
if (b == '0')
  {
    if (fa->bal_nulla == NULL)
      {
        fa->bal_nulla = uj_elem ();
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
      }
```

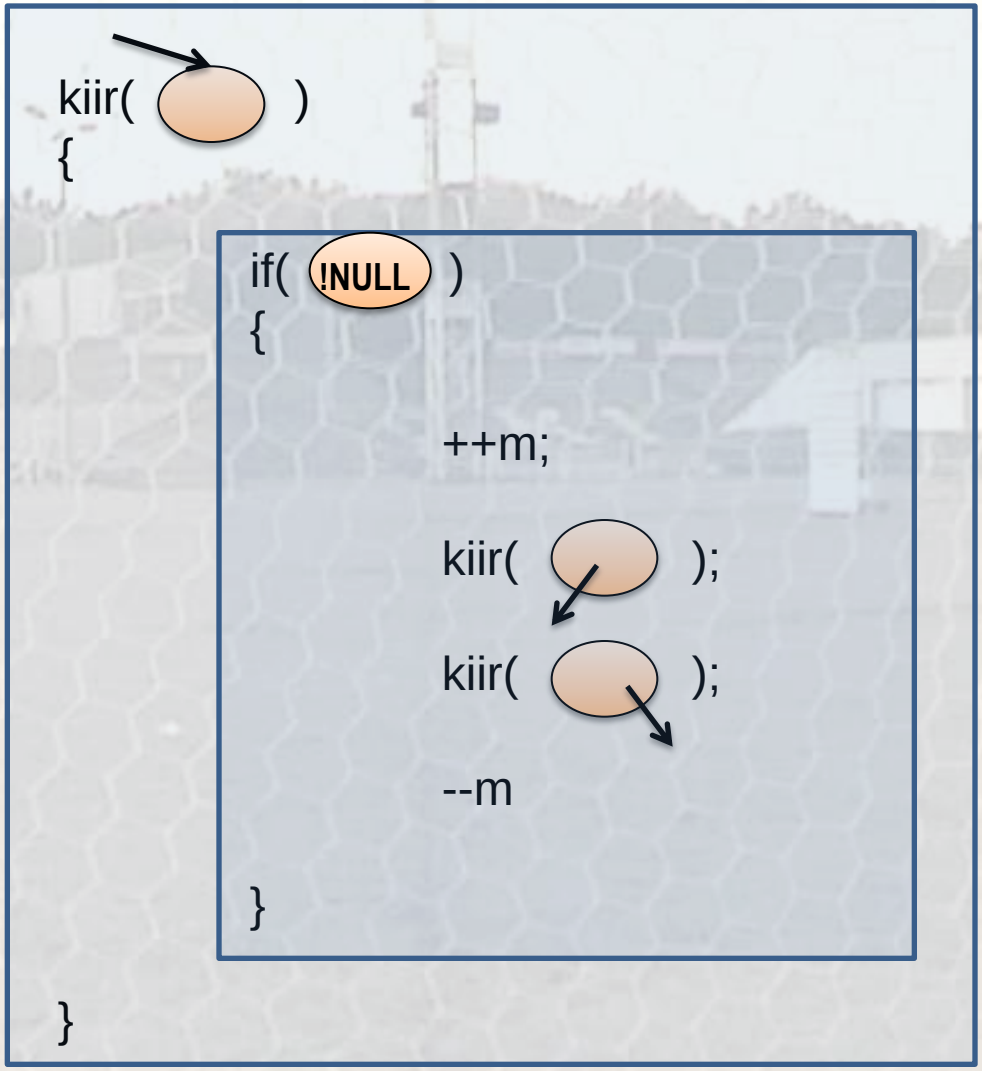


Fabejárás

(Lásd a K&R könyvben is! Ezért is választottam ilyen alapú példát a laboron védendőnek.)

++mélység heurisztikusan: a gyerek csomópontok feldolgozásához lejjebb kell lépni

-- feljebb lépünk, ha a gyerek csomópontokat feldolgoztuk.



Fabejárás

```
 // a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
 // http://sourceforge.net/projects/javacska/
 // az atlag() hivasakor is inicializalni kell oket, a
 // a rekurziv bejaras hasznalja
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{

  if (fa != NULL)
    {
      ++melyseg;
      ratlag (fa->jobb_egy);
      ratlag (fa->bal_nulla);
      --melyseg;

      if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {

          ++atlagdb;
          atlagosszeg += melyseg;

        }

    }

}
```

posztorder: a két részfa feldolgozása után foglalkozunk a gyökérelemmel.

Labor

A negyedik labortól már semmiképpen sem gépeljünk képekről forrást! Töltsük le a blog megfelelő posztjairól, rántsuk le a CVS-ből, vagy kérjük el valakitől! A PP javasolta manuál lapokat továbbra is nyissuk ki, nézzük meg!

A labor sikeres teljesítésének egyik szükséges feltétele

http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

A karakterekre működő LZW fa építő a linkelt posztban szerepelt, de binárisra úgy engedtük rá, hogy azt először az szintén szereplő „dump” progival karakteres 0,1 állománnyá alakítottuk. Most olyat kell írnod, ami kapásból binárisat dolgoz fel!

Íme a specifikáció:

Állomány: BNbeadando.c (saját monogram, ékezetes betű nincs)
Parancssor: ./BNbeadando input_fájl_neve -o kimeneti_fájl_neve (ha nem így indítják, kiírja, hogy így kellene használni és leáll)
Kimenet: tartalmazza a 0,1 bináris LZW fát, majd a fa mélységét, végül a fa águhosszainak átlagát és szórását (utóbbi a negyedik laboron is téma).

(Lesz néhány teszt állomány megadva, ahol előre meg lehet nézni, hogy megfelelően működik-e a kifejlesztett progitok.)

LZW

http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor :
Az algoritmus remek leírását találjuk a Rónyai-Iványos-Szabó Algoritmusok könyvben. Alább egy naivabb implementációt adunk a Tusnády: Sztochasztikus számítástechnika című könyv alapján: jön a 0-1 sorozat, betűnként olvassuk, ha olyan rész jön, ami még "nincs a zsákban", akkor "letörjük és be a zsákba":

```
00011101110 -> 0 00 1 11 01 110
```

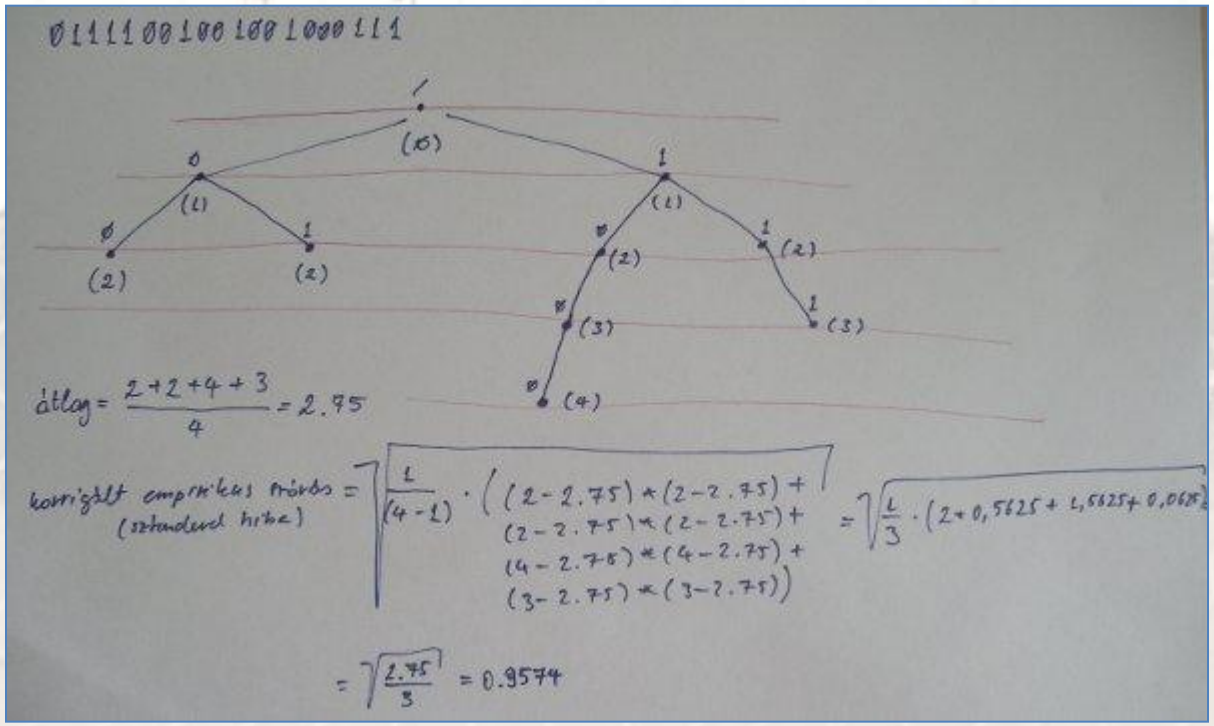
a tördelést egy bináris fával könnyen meg tudjuk valósítani:

```
              /
        0             1
    0       1               1
                        0
```

Így használd:

```
[norbi@sgu C]$ more b.txt
00011101110
[norbi@sgu C]$ gcc z.c -o z -std=c99
[norbi@sgu C]$ ./z < b.txt
00011101110
---------1
------------0
------1
---/
---------1
------0
---------0
```

LZW

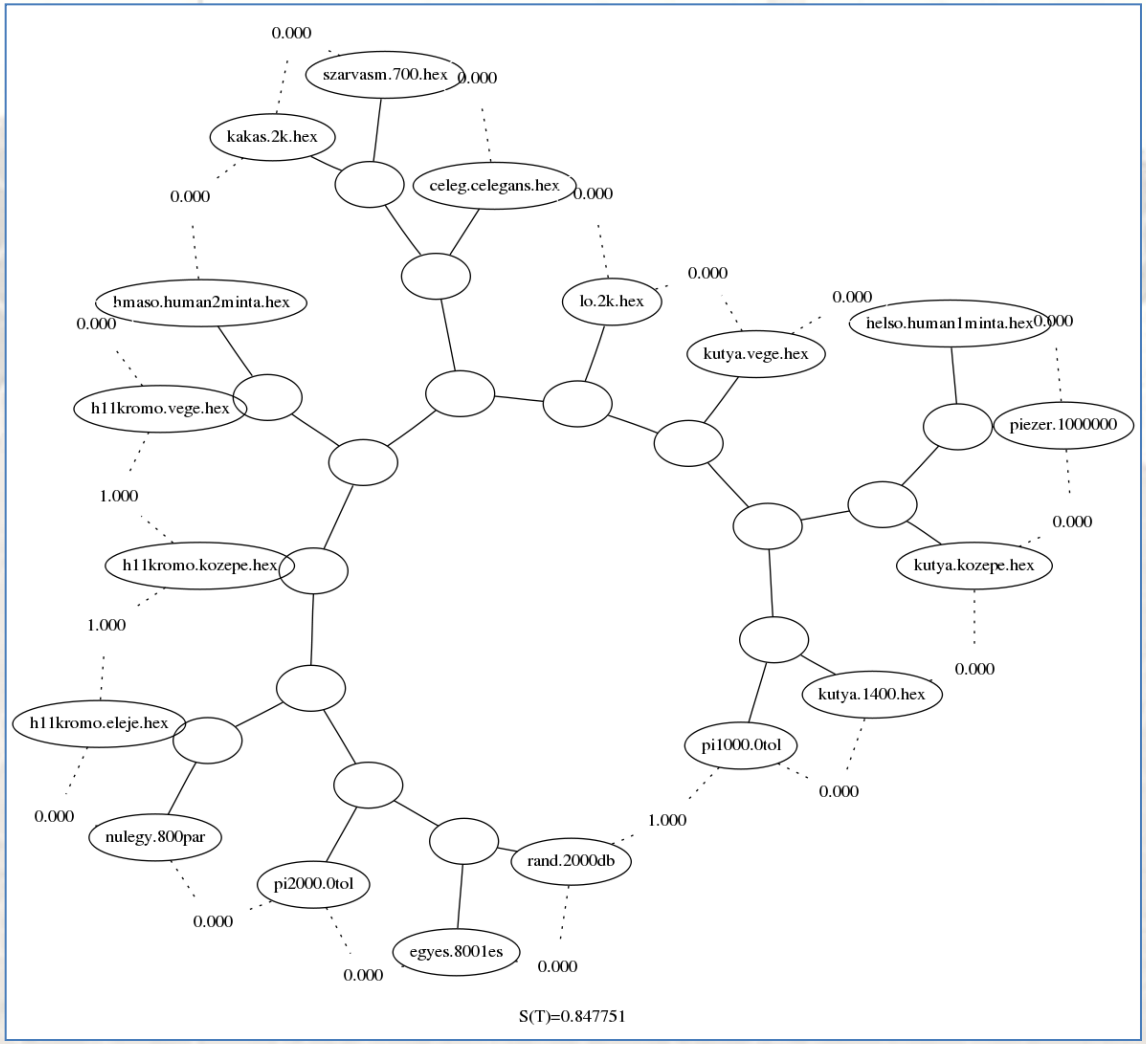


http://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat

Genetikai kód

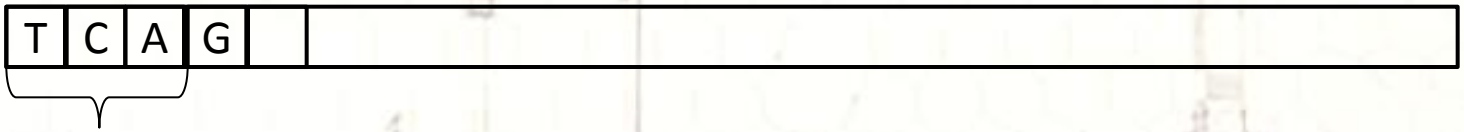
A UNIX típusú rendszerekben a fork() a mitózis.

http://progpater.blog.hu/2011/02/27/a_human_genom_projekt



Genetikai kód

A UNIX típusú rendszerekben a fork() a mitózis.



Ser=012(4) = 6(10)

http://progpater.blog.hu/2011/03/06/az_otodik_labor

```
[norbi@sgu tcag]$ ./gh <hs_alt_Hs_Celera_chr2.fa
Stop 4226021
Phe 4938666
Leu 8650548
Ile 4703612
Met 1421143
Val 3903156
Ser 6864063
Pro 4211007
Thr 3892567
Ala 3197516
Tyr 2514516
His 2541712
Gln 3049116
Asn 3230810
Lys 4968633
Asp 1735431
Glu 2846839
Cys 2645039
Trp 1450350
Arg 3795349
```

Poloskák

```
#define TOMB_MERET 10
#define ELOTTE 0
#define UTANA 1

char bsstomb[TOMB_MERET];
    A
char *
masol (const char *mit)
{
  char mibe[TOMB_MERET];
  char *p = mibe;
  char *vissza = mibe;

  for (; *p++ = *mit++;);

  // Vermen lokalisan foglalt terulet visszaadasa
  return vissza;
}

char *
csmasol (char *hova)
{
  int i = 0;
  for (i = -ELOTTE; i < TOMB_MERET + UTANA; ++i)
    *(hova + i) = '*';

  // Eggyel tulirva a sztring lezarasa
  *(hova + TOMB_MERET) = '\0';

  return hova;
}
```

Poloskák

```c
#define TOMB_MERET 10
#define ELOTTE 0
#define UTANA 1

char bsstomb[TOMB_MERET];

char *
masol (const char *mit)
{
  char mibe[TOMB_MERET];
  char *p = mibe;
  char *vissza = mibe;

  for (; *p++ = *mit++;);

  // Vermen lokalisan foglalt terulet visszaadasa
  return vissza;
}

char *
csmasol (char *hova)
{
  int i = 0;
  for (i = -ELOTTE; i < TOMB_MERET + UTANA; ++i)
    *(hova + i) = '*';

  // Eggyel tulirva a sztring lezarasa
  *(hova + TOMB_MERET) = '\0';

  return hova;
}
```

Poloskák

```
int
main (void)
{
  char veremtomb[TOMB_MERET];
  char *halomtomb;

  printf ("Verem tomb: [%s]\n", csmasol (veremtomb));

  printf ("Masolt tomb: [%s]\n", masol (veremtomb));

  printf ("BSS tomb: [%s]\n", csmasol (bsstomb));

  halomtomb = (char *) malloc (TOMB_MERET * sizeof (char));

  printf ("Heap tomb: [%s]\n", csmasol (halomtomb));

  free (halomtomb);

  // Felszabaditott terulet hasznalata
  printf ("Heap tomb: [%s]\n", csmasol (halomtomb));

  // Elfolyik a memoria
  halomtomb = (char *) malloc (TOMB_MERET * sizeof (char));

  return 0;
}
```

Poloskák

```
[morpheus@zion morpheus]$ gcc poloskak.c -o poloskak
[morpheus@zion morpheus]$ ./poloskak
Verem tomb: [**********]
Masolt tomb: []
BSS tomb: [**********]
Heap tomb: [**********]
Heap tomb: [**********]
```

```
nbatfai@hpserver:~> gcc poloskak.c -o poloskak
nbatfai@hpserver:~> ./poloskak
Verem tomb: [**********]
Masolt tomb: []
BSS tomb: [**********]
Heap tomb: [**********]
Heap tomb: [**********]
```

```
nbatfai@hallg:~/c$ gcc poloskak.c -o poloskak
nbatfai@hallg:~/c$ ./poloskak
Verem tomb: [**********]
Masolt tomb: [**********]
BSS tomb: [**********]
Heap tomb: [**********]
Heap tomb: [**********]
```

Poloskák - splint - A tool for statically checking C programs

```c
char *
masol (const char *mit)
{
  char mibe[TOMB_MERET];
  char *p = mibe;
  char *vissza = mibe;

  for (; *p++ = *mit++;);

  // Vermen lokalisan foglalt terulet visszaadasa
  return vissza;
}
```

```
poloskak.c:20:10: Stack-allocated storage vissza reachable from return value:
                    vissza
  A stack reference is pointed to by an external reference when the function
  returns. The stack-allocated storage is destroyed after the call, leaving a
  dangling reference. (Use -stackref to inhibit warning)
```

Poloskák - splint - A tool for statically checking C programs

```
free (halomtomb);

// Felszabaditott terulet hasznalata
printf ("Heap tomb: [%s]\n", csmasol (halomtomb));
```

```
poloskak.c:55:32: New fresh storage (type char *) passed as implicitly temp
                    (not released): csmasol(halomtomb)
```

```
// Elfolyik a memoria
halomtomb = (char *) malloc (TOMB_MERET * sizeof (char));
```

```
poloskak.c:60:12: Fresh storage halomtomb not released before return
   poloskak.c:58:3: Fresh storage halomtomb allocated
```

Poloskák

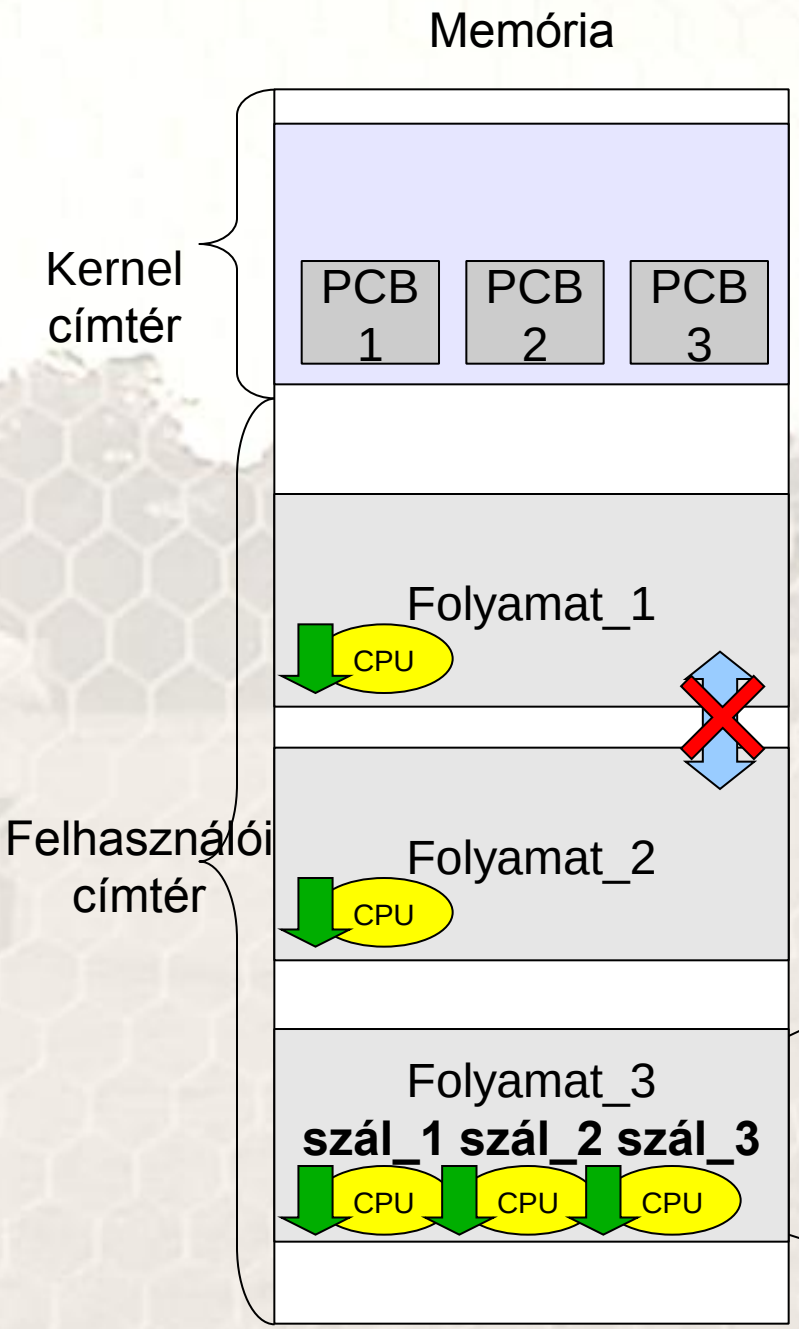
$ gcc poloskak.c -o poloskak
$ valgrind -v --leak-check=full ./poloskak

```
#define ELOTTE 0
#define UTANA 1

==5687== IN SUMMARY: 68 errors from 16 contexts (suppressed: 8 from 1)
==5687==
==5687== malloc/free: in use at exit: 10 bytes in 1 blocks.
==5687== malloc/free: 2 allocs, 1 frees, 20 bytes allocated.
```

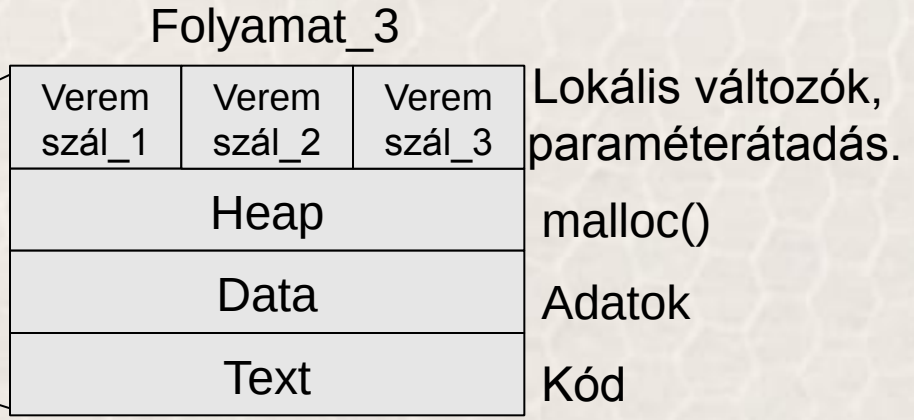
$ gcc -lefence poloskak.c -o poloskak
$ ./poloskak

(stb. kísérletezz magad is a poloskak.c-vel)

Szálak




```
PTHREADS(7)                    Linux Programmer's Manual
NAME
       pthreads - POSIX threads

DESCRIPTION
A single process can contain multiple threads,
all of which are executing the same program.
These threads share the same global emory (data
and heap  segments), but  each  thread  has its
own stack (automatic variables).
...
```



A (POSIX threads) pthreads könyvtár, pthreads_

```
PTHREADS(7)                Linux Programmer's Manual                PTHREADS(7)

NAME
       pthreads - POSIX threads

DESCRIPTION
       POSIX.1  specifies  a  set  of interfaces (functions, header files) for
       threaded programming commonly known as POSIX threads, or  Pthreads.   A
       single process can contain multiple threads, all of which are executing
       the same program.  These threads share the same global memory (data and
       heap  segments),  but  each  thread  has its own stack (automatic vari-
       ables).
```

```
PTHREAD_CREATE(P)          POSIX Programmer's Manual          PTHREAD_CREATE(P)

NAME
       pthread_create - thread creation

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *restrict thread,
              const pthread_attr_t *restrict attr,
              void *(*start_routine)(void*), void *restrict arg);

DESCRIPTION
       The   pthread_create()   function  shall  create  a  new  thread,  with
```

pthreads_

```
PTHREAD_CREATE(P)

NAME
       pthread_create - thread creation

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *restrict thread,
              const pthread_attr_t *restrict attr,
              void *(*start_routine)(void*), void *restrict ar
```

4.

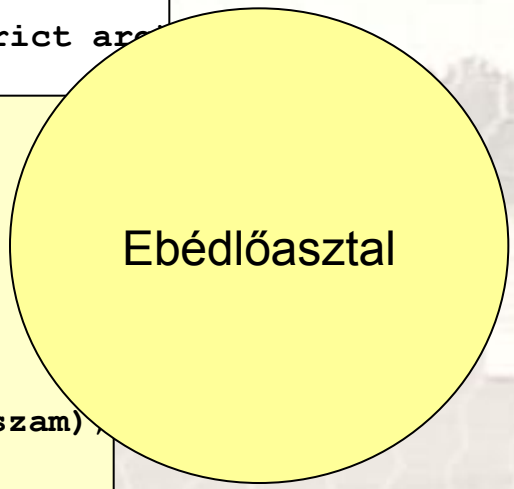
```
#define FILOSZOK_SZAMA 5
sem_t villa[FILOSZOK_SZAMA];

void *egy_filosz(void *id)
{
  int sorszam = *(int *)id;
  printf("%d. filosz jelen.\n", sorszam);
  fflush(stdout);
  for(;;)
  {
       // villákat felveszem
       // (sem_wait)
       // eszek
       // villákat leteszem
       // (sem_post)
  }
  return id;
}
```

PP 70

4.

3.

3. szál fut



0. villa

0. filozófus

1. villa

1.

2.

3.

2.

A pthreads könyvtár, mutex zárak, pthreads_

```
PTHREAD_MUTEX_LOCK(P)       POSIX Programmer's Manual      PTHREAD_MUTEX_LOCK(P)

NAME
       pthread mutex lock,  pthread mutex trylock, pthread mutex unlock - lock
       and

SYNOPSIS
       #in

       int
       int
       int

DESCRIPTIO
       The                                                    by  calling
       pth                                                   the calling
       thr                                                    operation
       sha
       state with the cal
```

```
void *
novel_szal(void *id)
{
  int i;
  for(i=0; i<100; ++i)
  {
    printf("Szal: %d, %d\n", *(int *)id, pthread_self());
    fflush(stdout);
    var();
    szamlalo = szamlalo + 1;
  }
  return id;
}
```

```
void *
csokkent_szal(void *id)
{
  int i;
  for(i=0; i<100; ++i) {
    printf("Szal: %d, %d\n", *(int *)id, pthread_self());
    fflush(stdout);
    var();
    szamlalo = szamlalo - 1;
  }
  return id;
}
```

PP 67

```
.
.
.
Szal: 98, 1622116
Szal: 96, 1589346
Szal: 98, 1622116
Szal: 96, 1589346
Szal: 96, 1589346
A szamlalo vegul: -2
```

A pthreads könyvtár, szemaforok, sem_

```
SEM_OVERVIEW(7)            Linux Programmer's Manual           SEM_OVERVIEW(7)

NAME
       sem_overview - Overview of POSIX semaphores

DESCRIPTION
       POSIX  semaphores  allow  processes  and  threads  to synchronise their
       actions.

       A semaphore is an integer whose value is never allowed  to  fall  below
       zero.   Two  operations  can  be performed on semaphores: increment the
       semaphore value by one (sem_post(3)); and decrement the semaphore value
       by  one  (sem_wait(3)).  If the value of a semaphore is currently zero,
       then a sem_wait(3) operation will block until the value becomes greater
       than zero.

       POSIX  semaphores  come  in  two  forms:  named  semaphores and unnamed
       semaphores.
```

A pthread könyvtár és a 2.4, 2.6 Linux kernel

```
$uname -r
2.4.19

$ ps axHo comm,pid,ppid,stat,tid,nlwp
COMMAND            PID  PPID STAT   TID NLWP
szerver          22265 22226 S+   22265    1
szerver          22266 22265 S+   22266    1
szerver          22267 22266 S+   22267    1
szerver          22268 22266 S+   22268    1
szerver          22269 22266 S+   22269    1
szerver          22270 22266 S+   22270    1
szerver          22271 22266 S+   22271    1
```

PP 119

```
$uname -r
2.6.11

$ ps axHo comm,pid,ppid,stat,tid,nlwp
COMMAND            PID  PPID STAT   TID NLWP
szerver          20115 20077 Sl+  20115    6
szerver          20115 20077 Sl+  20116    6
szerver          20115 20077 Sl+  20117    6
szerver          20115 20077 Sl+  20118    6
szerver          20115 20077 Sl+  20119    6
szerver          20115 20077 Sl+  20120    6
```

NPTL, Native POSIX Threads Library

```
$ getconf GNU_LIBPTHREAD_VERSION
NPTL 2.5
```

```
PTHREADS(7)                Linux Programmer's Manual                PTHREADS(7)

NAME
       pthreads - POSIX threads

DESCRIPTION
       POSIX.1 specifies a set of interfaces (functions, header files) for
       threaded programming commonly known as POSIX threads, or Pthreads.  A
       single process can contain multiple threads, all of which are executing
       the same program.  These threads share the same global memory (data and
       heap segments), but each thread has its own stack (automatic vari-
       ables).
...

 NPTL
       With NPTL, all of the threads in a process are placed in the same
       thread group; all members of a thread groups share the same PID.  NPTL
       does not employ a manager thread.  NPTL makes internal use of the first
       two real-time signals; these signals cannot be used in applications.
```

Jelek

```
SIGNAL(7)                  Linux Programmer's Manual                  SIGNAL(7)

NAME
       signal - list of available signals
...
 Standard Signals
       Linux supports the standard signals listed below. Several  signal  num-
       bers  are  architecture  dependent, as indicated in the "Value" column.
       (Where three values are given, the first one is usually valid for alpha
       and  sparc,  the  middle one for i386, ppc and sh, and the last one for
       mips.  A - denotes that a signal is absent on the corresponding  archi-
       tecture.)

       First the signals described in the original POSIX.1-1990 standard.

       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
       S ...
       S       SIGCHLD   20,17,18    Ign     Child stopped or terminated
       S       SIGCONT   19,18,25    Cont    Continue if stopped
       S       SIGSTOP   17,19,23    Stop    Stop process
       S       SIGTSTP   18,20,24    Stop    Stop typed at tty
       S       SIGTTIN   21,21,26    Stop    tty input for background process
               SIGTTOU   22,22,27    Stop    tty output for background process

       The  signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
```

System V, BSD, POSIX jelkezelés

```
SIGNAL(2)                  Linux Programmer's Manual                 SIGNAL(2)

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);

DESCRIPTION
       The  signal()  system call installs a new signal handler for the signal
       with number signum.  The signal handler is set to sighandler which  may
       be a user specified function, or either SIG_IGN or SIG_DFL.
```

```
#include <stdio.h>
#include <signal.h>
int
main(void)
{
   signal(SIGINT, SIG_IGN);
   sleep(5);
   signal(SIGINT, SIG_DFL);
   for(;;)
     sleep(5);
   return 0;
}
```

PP 41

MM 382

System V, **BSD**, POSIX jelkezelés

```
SIGNAL(2)                  Linux Programmer's Manual                 SIGNAL(2)

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);
...

RETURN VALUE
       The signal() function returns the previous value of the signal handler,
       or SIG_ERR on error.
```

```
#include <stdio.h>
#include <signal.h>
void utolso_tennivalo(int sig)
{
  printf("Utolso tennivalo kesz, immar kilephetek\a\n");
  exit(0);
}
int
main(void)
{
  if(signal(SIGINT, utolso_tennivalo) == SIG_IGN)
    signal(SIGINT, SIG_IGN);
  for(;;)
    putchar(getchar());
  return 0;
}
```

System V, BSD, POSIX jelkezelés

```
SIGNAL(2)                  Linux Programmer's Manual                 SIGNAL(2)

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);

DESCRIPTION
       The  signal()  system call installs a new signal handler for the signal
       with number signum.  The signal handler is set to sighandler which  may
       be a user specified function, or either SIG_IGN or SIG_DFL.
```

```
#include <stdio.h>
#include <signal.h>
void ctrlc_kezelo(int sig)
{
  signal(SIGINT, ctrlc_kezelo);
  printf("Megprobaltal megallitani?\a\n");
}
int
main(void)
{
  if(signal(SIGINT, ctrlc_kezelo) == SIG_IGN)
    signal(SIGINT, SIG_IGN);
  for(;;)
    putchar(getchar());
  return 0;
}
```

PP 42

System V, BSD, **POSIX** jelkezelés

```
SIGACTION(2)                Linux Programmer's Manual                SIGACTION(2)

NAME
       sigaction - examine and change a signal action

SYNOPSIS
       #include <signal.h>

...
      The sigaction structure is defined as something like

              struct sigaction {
                  void (*sa_handler)(int);
                  void (*sa_sigaction)(int, siginfo_t *, void *);
                  sigset_t sa_mask;
                  int sa_flags;
                  void (*sa_restorer)(void);
              }
```

```
#include <stdio.h>
#include <signal.h>
void ctrlc_kezelo(int sig)
{
  printf("Megprobaltal megallitani?\a\n");
}
int
main(void)
{
  struct sigaction sa;
  sa.sa_handler = ctrlc_kezelo;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = SA_RESTART;
  sigaction(SIGINT, &sa, NULL);
  for(;;)
    putchar(getchar());
  return 0;
}
```

Hol találkozol vele pl.?

crashme.c „újabb” verziók

agent2d-3.0.0/src/main_player.cpp

Nem lokális ugrások

```
SETJMP(3)                    Library functions                    SETJMP(3)

NAME

SYNOPS
```

```
LONGJMP(3)                   Library functions                   LONGJMP(3)

NAME
       longjmp, siglongjmp - non-local jump to a saved stack context

SYNOPSIS
       #include <setjmp.h>
```

```
DE                                                    val);

                                                      for dealing with errors and inter-
..
```

```
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
sigjmp_buf jmpbuf;
void
kezdjuk_ujra (int sig)
{
  signal (SIGINT, kezdjuk_ujra);
  printf ("Megzavartal, ujra kezdjuk\a\n");
  siglongjmp (jmpbuf, 0);
}

int
main (void)
{
  if (signal (SIGINT, kezdjuk_ujra) == SIG_IGN)
    signal (SIGINT, SIG_IGN);
  sigsetjmp (jmpbuf, 1);
  printf("Kezdjuk!");
  for (;;)
    putchar (getchar ());
  return 0;
}
```

PP 42

```
$ gcc ugras.c -o ugras
$ ./ugras
Kezdjuk!alma
alma  Ctrl+C
Megzavartal, ujra kezdjuk
Kezdjuk!korte
korte Ctrl+C
Megzavartal, ujra kezdjuk
Kezdjuk!
[1]+  Stopped                 ./ugras
$ kill %1
$
[1]+  Terminated              ./ugras
```

Crashme

Letöltés: http://packages.debian.org/stable/source/crashme

PP 45

```
CRASHME(1)                                                    CRASHME(1)

NAME
       crashme - test operating environment software robustness

SYNOPSIS
       crashme [NBYTES] [SRAND] [NTRYS] [NSUB] [VERBOSE]

DESCRIPTION
       crashme is a very simple program that tests the operating environment's
       robustness by invoking random data as if  it  were  a  procedure.   The
       standard  signals  are  caught and handled with a setjmp back to a loop
       which will try again to produce a fault by executing random data.  Some
       people call this stress testing.
```

Laborkártyák

```c
#include <stdio.h>
int
main (void)
{

  struct {
    int alma;
  } p = {55}, *pp;

  pp = &p;

  printf ("%d\n", (*pp).alma);

 (*pp).alma = 65;

  printf ("%d\n", pp->alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

```c
#include <stdio.h>
int
main (void)
{

  struct aaa
  {
    int alma;
  } p = {55}, *pp;

  pp = &p;

  printf ("%d\n", (*pp).alma);

  (*pp).alma = 65;

  printf ("%d\n", pp->alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

Laborkártyák

```c
#include <stdio.h>
int
main (void)
{

  struct
  {
    int alma;
  } p =  {55}, *pp;

  pp = &p;

  printf ("%d\n", (*pp).alma);

  *(pp.)alma = 65;

  printf ("%d\n", pp->alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

```c
#include <stdio.h>
int
main (void)
{

  struct
  {
    int alma;
  } p =  {55}, *pp;

  pp = &p;

  printf ("%d\n", (*pp).alma);

  pp->alma = 65;

  printf ("%d\n", (*pp).alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

Laborkártyák

```c
#include <stdio.h>
int
main (void)
{

  struct
  {
    int alma;
  } p =  {55}, *pp;

  pp = &p;

  printf ("%d\n", ++pp->alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

```c
#include <stdio.h>
int
main (void)
{

  struct
  {
    int alma;
  } p =  {55}, *pp;

  pp = &p;

  printf ("%d\n", ++(pp->alma));

  return 0;
}
```

Ha lefordul, mit ír ki?

Laborkártyák

```c
#include <stdio.h>
int
main (void)
{
  int korte = 54;

  struct
  {
    int *alma;
  } p =  {&korte}, *pp;

  pp = &p;

  printf ("%d\n", *pp->alma);

  return 0;
}
```

Ha lefordul, mit ír ki?

```c
#include <stdio.h>
int
main (void)
{
  int korte = 54;

  struct
  {
    int *alma;
  } p =  {&korte}, *pp;

  pp = &p;

  printf ("%d\n", (*pp->alma)++);
  printf ("%d\n", (*pp->alma)++);

  return 0;
}
```

Ha lefordul, mit ír ki?

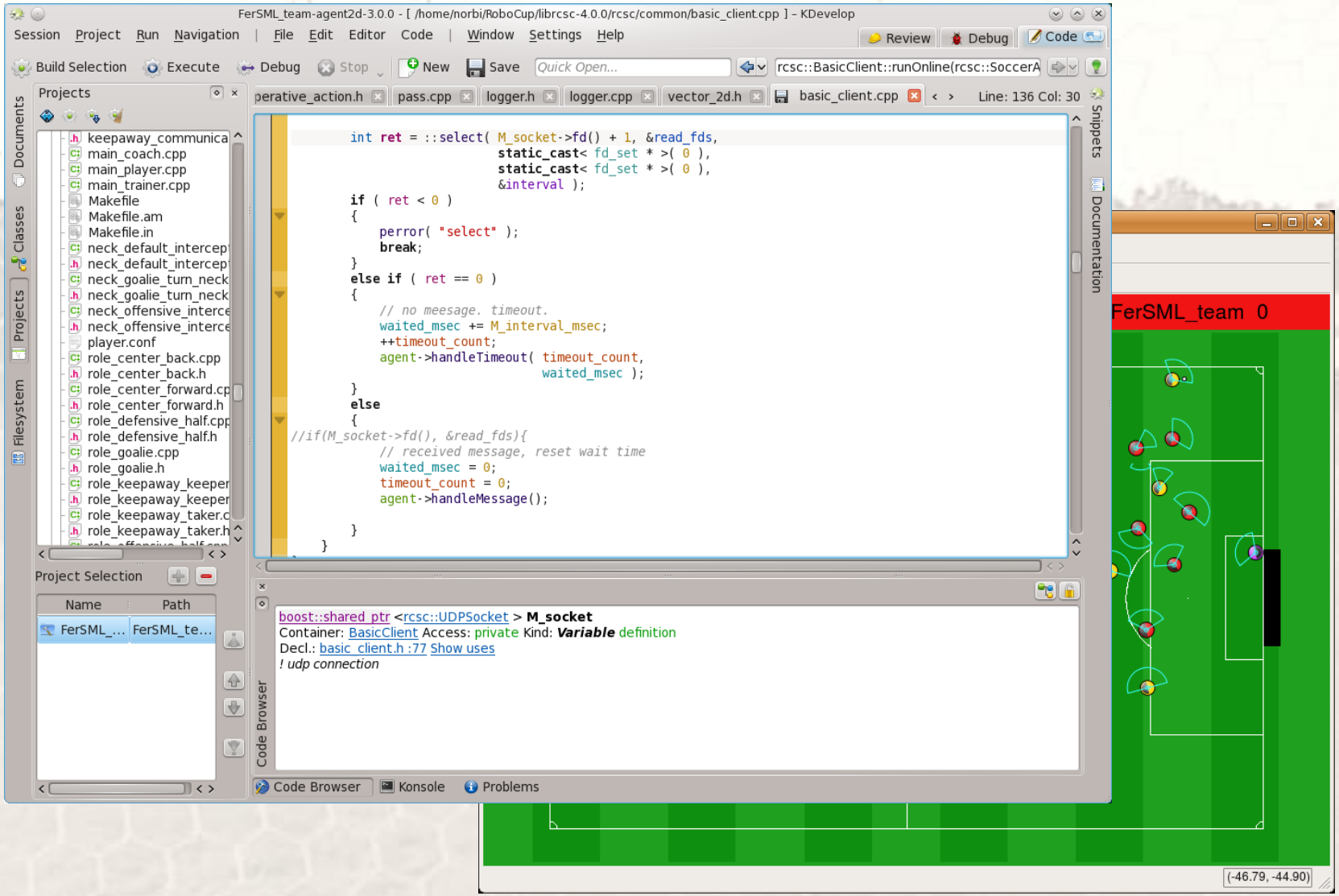
Laborkártyák

```
char *
masol (const char *mit)
{
  char mibe[TOMB_MERET];
  char *p = mibe;
  char *vissza = mibe;

  for (; *p++ = *mit++;);

  // Vermen lokalisan foglalt terulet visszaadasa
  return vissza;
}
```

Mi a véleményed erről a függvényről?

Otthoni opcionális feladat

A robotfoci japán szoftvereinek (librcsc, agent2d) tanulmányozása a KDevelop-ban.



Kötelező olvasmány

K&R könyvből olvassuk el (többször!) a hatodik, hetedik és nyolcadik fejezetet:

a) Struktúrák
b) Bevitel és kivitel
c) Csatlakozás a UNIX operációs rendszerhez