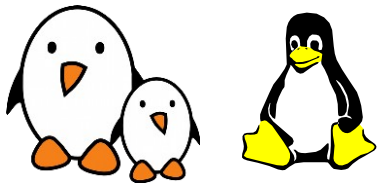
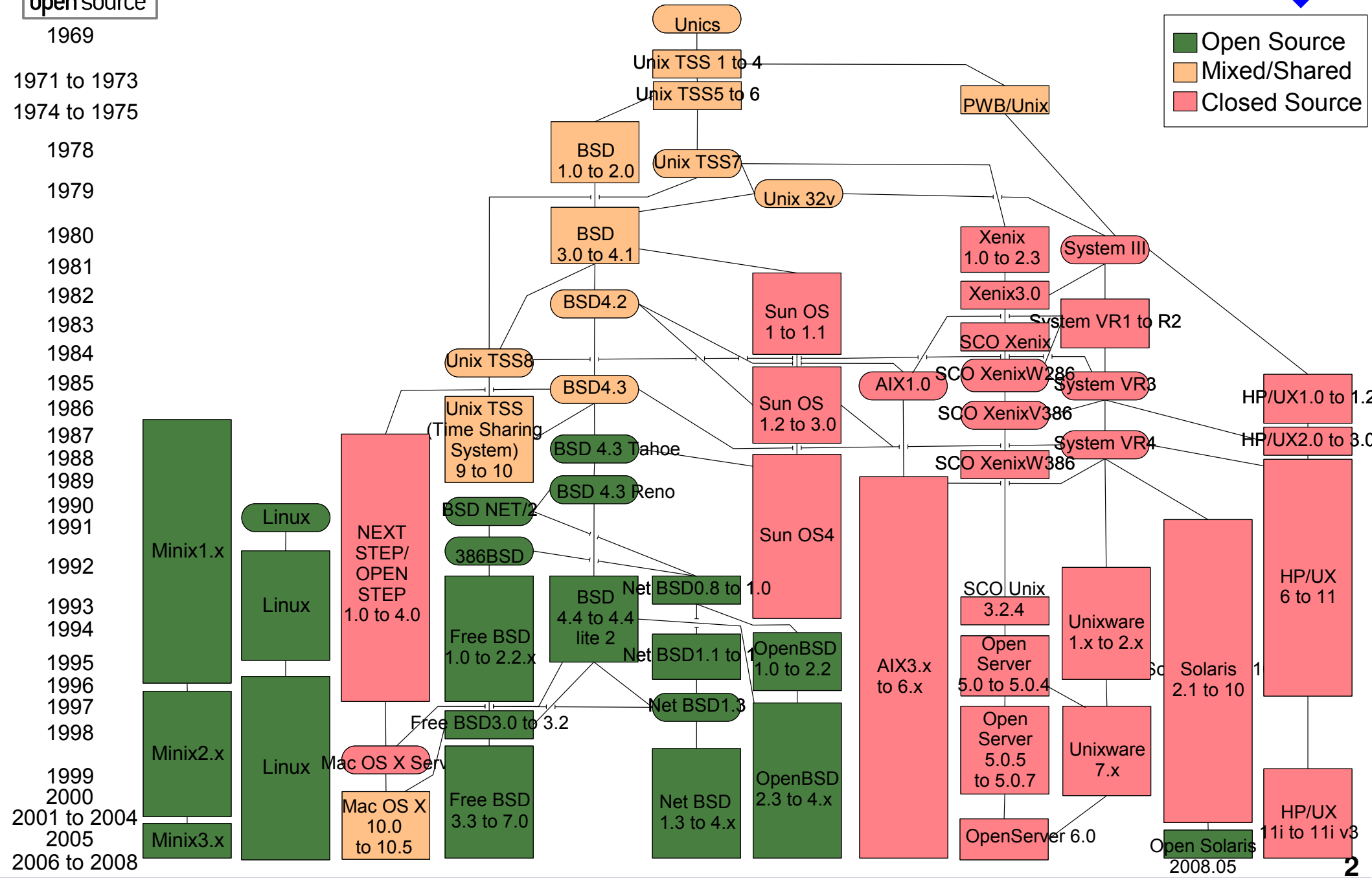
Open Source Programování

http://rtime.felk.cvut.cz/osp/

Pavel Píša
<pisa@fel.cvut.cz>
http://cmp.felk.cvut.cz/~pisa
Michal Sojka
František Vacek
DCE FEL ČVUT



A jak to bylo dál



Unix – hlavní větve

BSD family

BSD (Berkeley Software Distribution)

FreeBSD

OpenBSD

NetBSD

Bill Joy

Bill Joy

SunOS (Stanford University Network)

NextStep

MacOS X

GNU

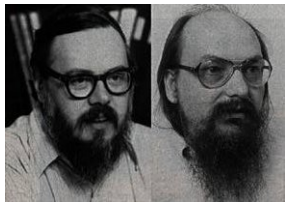
GNU / Linux

Richard Stallman

Linus Torvalds

Bell Labs (AT&T)

Ken Thompson
Dennis Ritchie (C language
created to implement a portable OS)



Ritchie, Thompson

SRV5

IRIX (SGI)

HP-UX

Sun Solaris

AIX (IBM)

System V family

1970 1980 1990 2000 Time

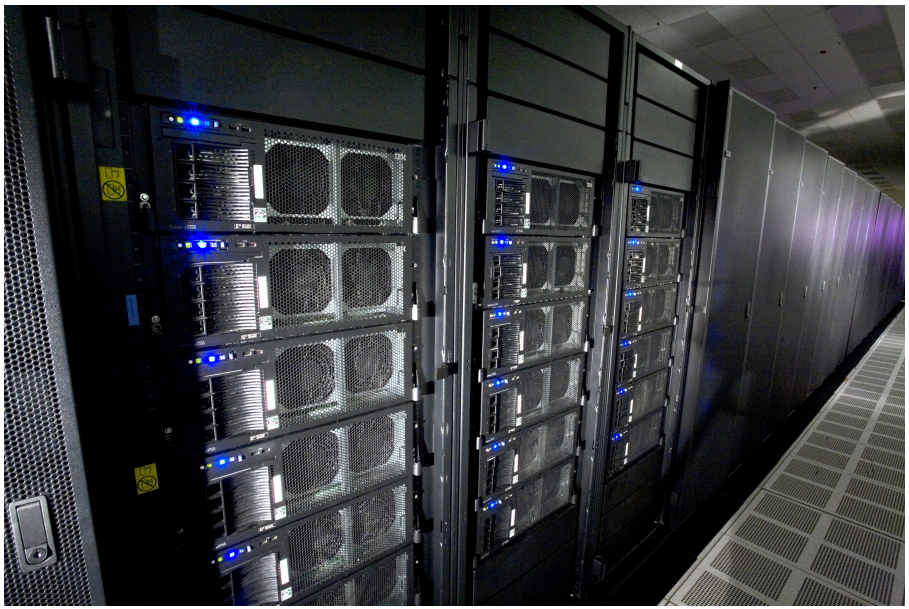
Linux kernel -Trocha historie

- 5.1. 1991 Linus Torvalds kupuje IBM PC
- V srpnu 1991 Linus informace o záměru napsat jádro
- 1991 – verze 0.01 publikována na internetu
- 1994 – verze 1.0 – jeden i386 CPU
- 1996 – verze 2.0 – SMP pro aplikace, BKL pro jádro
- 1999 – verze 2.2 – spinlock, m68k a PowerPC
- 2001 – verze 2.4 – ISA PnP, USB, PC Cards, PA-RISC, později Bluetooth, LVM, RAID, ext3
- 2003 – verze 2.6 – mainline µClinux, ARM a další, PAE, ALSA, preemption, Native POSIX Thread Library, Futex, později FUSE, JFS, XFS, ext4, robust mutex, prio inherit mutex, high resolution timers
- Okolo 2007 – CONFIG_PREEMPT_RT, spinlock→RT-mutex, rušení BKL, IRQ→thready, preemptible RCU

GNU/Linux on trpaslíků

… po superpočítače

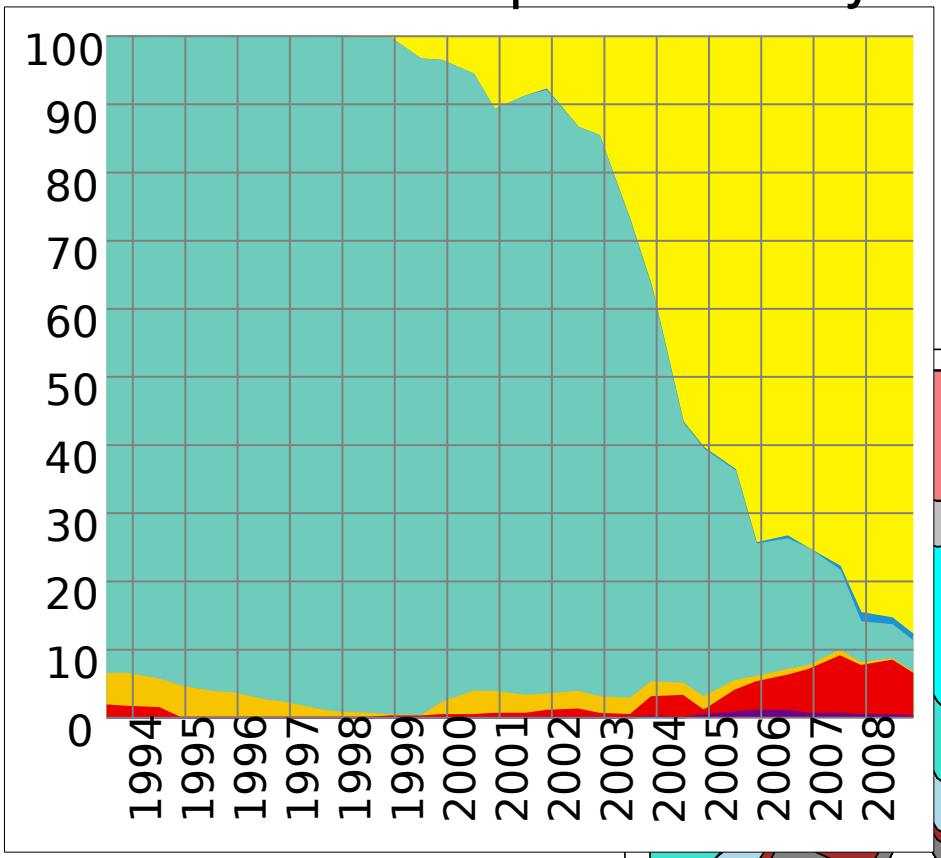
- IBM Roadrunner, Los Alamos National Laboratory
- Majitel: National Nuclear Security Administration, USA
- LINPACK/BLAS (Basic Linear Algebra Subprograms) systém
- Architectura: 12,960 IBM PowerXCell 8i CPUs,
  6,480 AMD Opteron dual-core processors, Infiniband, Linux
- Systém: Red Hat Enterprise Linux a Fedora
- Napájení: 2.35 MW
- Velikost: 296 stojanů
  560 m2
- Paměť: 103.6 TiB
- Výkon 1.042 petaflops
- Cena: USD $125M



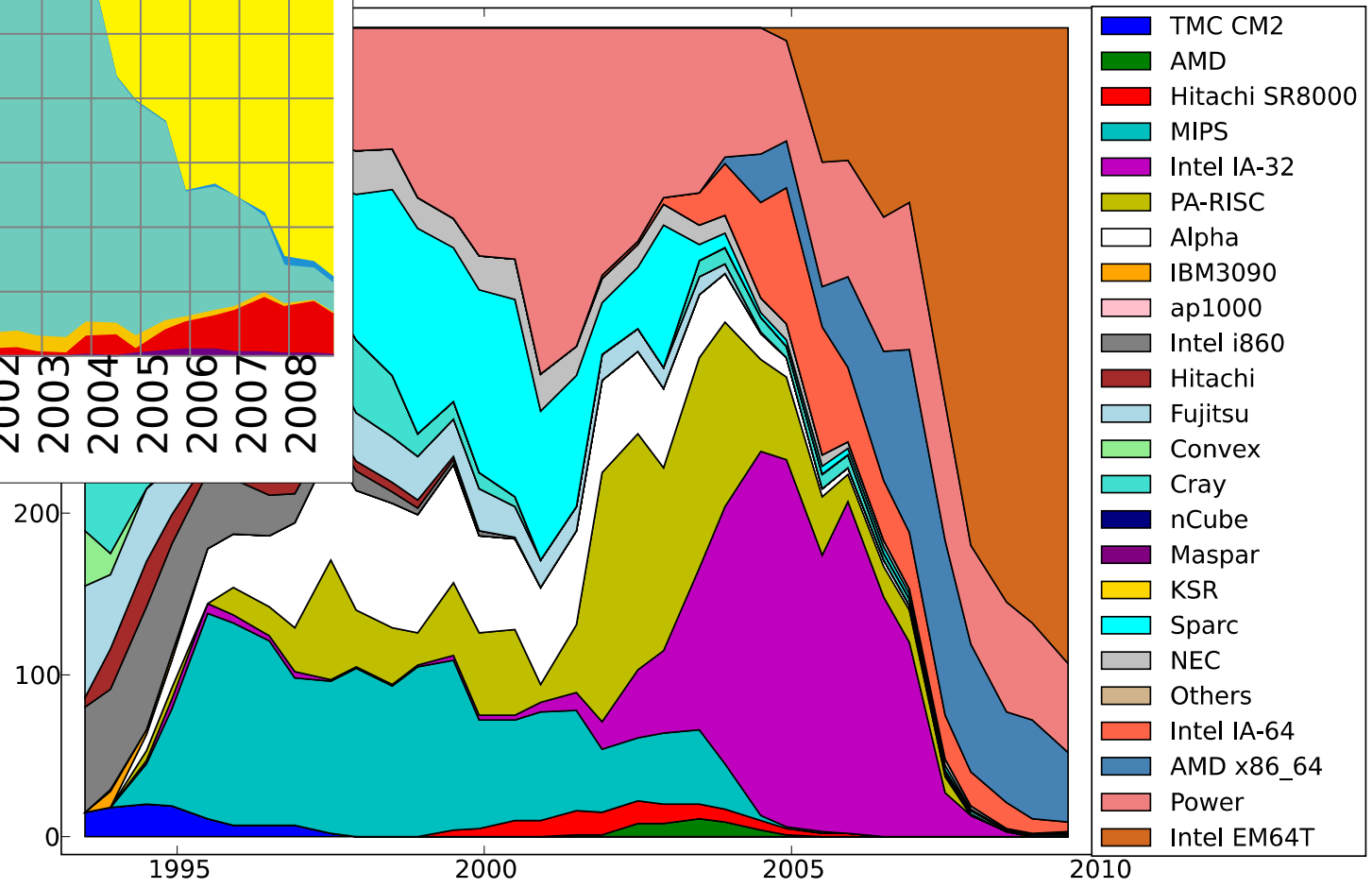
- SGI SSI (single system image) Linux, 2048 Itanium CPU a 4TiB RAM

Top 500 Supercomputers

Podle operačního systému



Podle CPU architektury

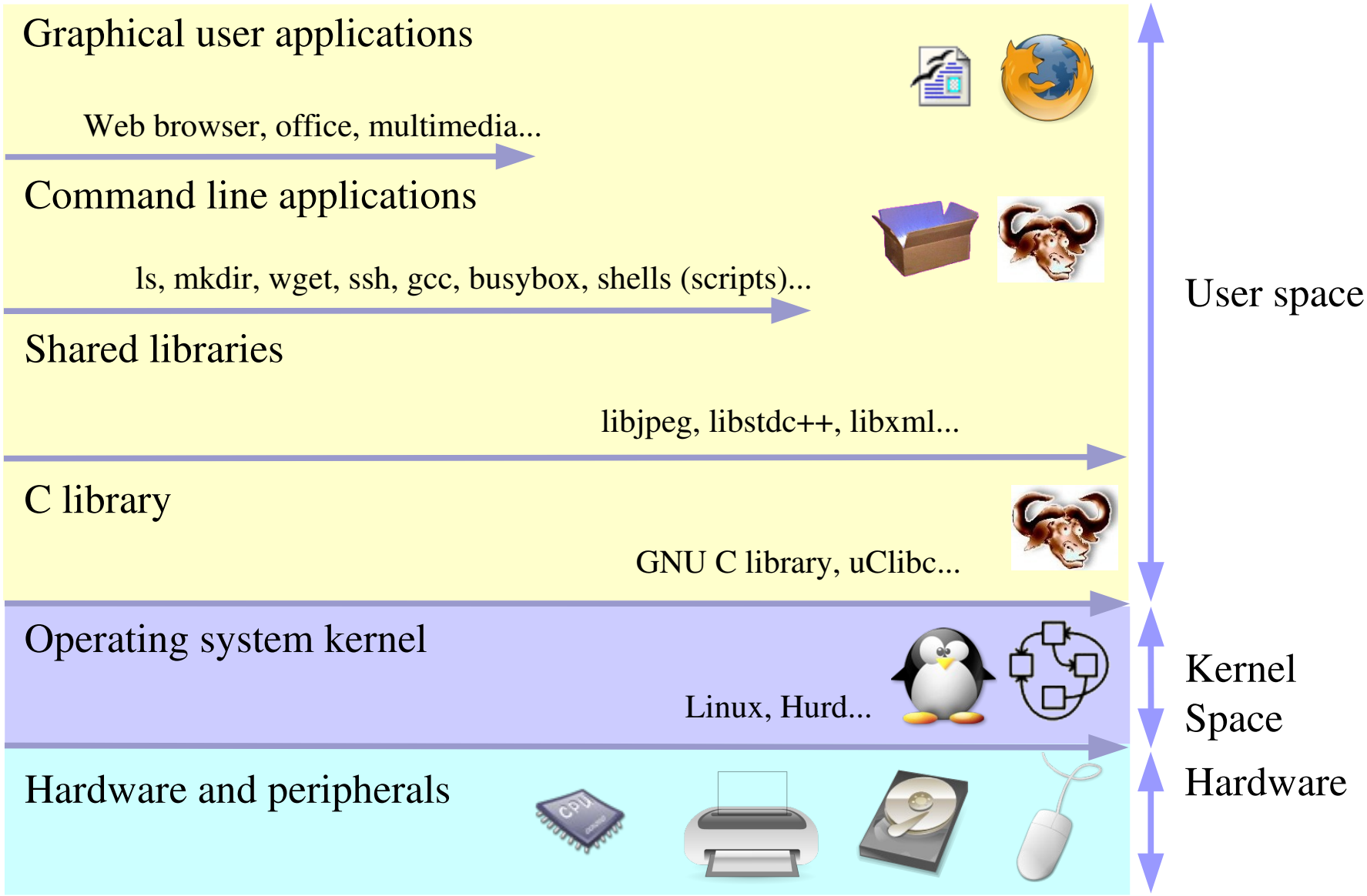


http://www.top500.org/

A prognóza pro vestavné systémy

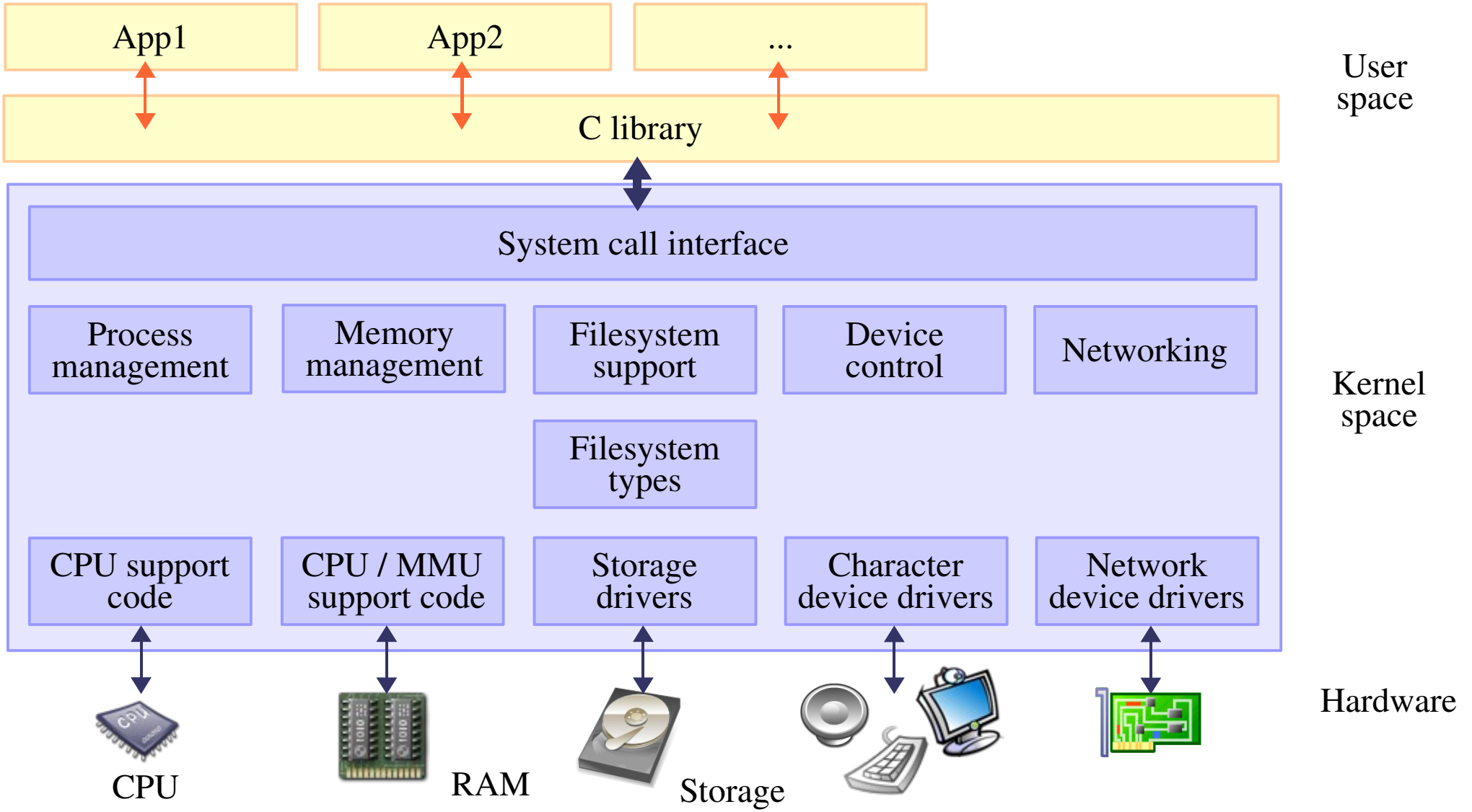
- Operating system used in the previous project
  - Proprietary OS: 39%
  - Free of cost embedded Linux: 29%
  - Embedded Linux with commercial support: 11%
  - Home grown OS: 7%
  - No OS: 11%
- Operating system planned for the next project
  - Free of cost embedded Linux: 71%
  - Embedded Linux with commercial support: 16%
  - Proprietary OS: 12%
  - Home grown OS: 1%
- Source: Venture Development Corp, October 2007
  http://www.vdc-corp.com/_documents/pressrelease/press-attachment-1394.pdf

Architektura (unixových) systémů



Převzato z http://free-electrons.com/docs/freesw (Michael Opdenacker)

Spodní vrstvy operačního systému



Skutečné programování

- Mějme jednoduchý požadavek alokovat a uvolňovat unikátní čísla (např. PID) v určitém rozsahu (alloc_pid)

```
int pid_used[MAX_PID];
int alloc_pid(void)
{ static int pid = 1;
  for(i = 1; i<MAX_PID; i++) {
    if (!pid_used[pid]) {
      pid_used[i] = 1;
      return i;
    } else {
      pid = pid<MAX_PID-1?pid+1:1;
    }
  }
  Return 0;
}
```

Kód asi není dobře
Chybí SMP, přidáme zámky
Ó to je pomalé
Zkusíme atomické operace
a bitové pole

Na jednom CPU to jde, ale co cache na 2048 SSI (MSI, MOSI, MOESI si vzalo veškerý výkon)

```
while (1) {
  if (!test_and_set_bit(...)) {
    atomic_dec(&nr_free);
    return pid;
  }
}
```

Tak jak to stím PIDem je

- Ve skutečnosti je to mnohem složitější
- Protože jádro podporuje i jmenné prostory (lehká virtualizace), potřebujeme celý řetěz PIDů pro každý name space
- Nemůžeme použít statické pole, max user processes je jen administrátorem nastavitelná konstanta (ulimit -u / -a)

  Max 4 millióny PID na 64-bit arch., horní bity pro robust futex

  na 32-bit max 32 tisíc, CONFIG_BASE_SMALL 8 tisíc
- Vše tedy musí být dynamické, nelze zabrat tolik paměti zbytečně, zároveň použití PID jako jednoznačného ID je v jádře díky NS nemožné
- Řešení: `struct pid` – zároveň slouží jako reference na a pro `struct task_struct`
- `PIDMAP_ENTRIES ((PID_MAX_LIMIT+8*PAGE_SIZE-1)/PAGE_SIZE/8)`
- Ukázka `make cscope`, `kscope` a `make htmldocs`

Jak na mutex, vezmi futex

- FUTEX – fast userspace mutex
- Pro shared převod na vma+index+offs→mezi procesy (get_futex_key)
- Základ pro pthread_mutex_init, _lock, _trylock, _unlock, _destroy
- Ale i sem_post, sem_wait a veškeré pthread_cond
- FUTEX_ + WAIT
  + WAKE
  + FD
  + REQUEUE
  + CMP_REQUEUE
  + WAKE_OP
  + LOCK_PI
  + UNLOCK_PI
  + TRYLOCK_PI
  + WAIT_BITSET
  + WAKE_BITSET
  + WAIT_REQUEUE_PI
  + CMP_REQUEUE_PI

```
static inline int
usema_down(ulock_t *ulock)
{
    if (!__ulock_down(ulock))
            return 0;
    return sys_ulock_wait(ulock);
}

static inline int
usema_up(ulock_t *ulock)
{
    if (!__ulock_up(ulock))
            return 0;
    return sys_ulock_wakeup(ulock);
}
```

Pospora v GLIBC

- Robust FUTEX od 2.6.17
- Priority inheritance pro uživatelské procesy od 2.6.18
  pthread_mutexattr_setprotocol (..., PTHREAD_PRIO_INHERIT)
- glibc/nptl/sysdeps/unix/sysv/linux/i386/lowlevellock.h
- Ulrich Drepper: Futexes Are Tricky
  http://people.redhat.com/drepper/futex.pdf

První pokus o futex

```
class mutex
{
  public:
   mutex () : val (0) { }
   void lock () {
     int c;
     while ((c = atomic_inc (val)) != 0)
       futex_wait (&val, c + 1); }
   void unlock () {
     val = 0; futex_wake (&val, 1); }
  private:
   int val;
};
```

Špatně:

- livelocks caused by the unconditional change of the futex variable must be avoided
- the futex value must not overflow
- in case it is known no threads wait on the mutex the futex wake call should be avoided.

Druhý pokus o futex

```
class mutex2
{
  public:
   mutex () : val (0) { }
   void lock () {
     int c;
     if ((c = cmpxchg (val, 0, 1)) != 0)
       do {
          if (c == 2
              || cmpxchg (val, 1, 2) != 0)
            futex_wait (&val, 2);
       } while ((c = cmpxchg (val, 0, 2))
                != 0);
   }
   void unlock () {
     if (atomic_dec (val) != 1) {
       val = 0;
       futex_wake (&val, 1);
     }
   }
  private:
   int val;
};
```

		mutex	mutex2
lock	atomic op	1	1
	futex syscall	0	0
unlock	atomic op	0	1
	futex syscall	1	0

Cena bez kolize

Cena při kolizi

		mutex	mutex2
lock	atomic op	$1+1$	$\frac{2}{3}+\frac{1}{2}$
	futex syscall	$1+1$	$1+1$
unlock	atomic op	0	1
	futex syscall	1	1

Finální řešení obyčejného futexu

```
class mutex3
{
  public:
   mutex () : val (0) { }
   void lock () {
     int c;
     if ((c = cmpxchg (val, 0, 1)) != 0) {
       if (c != 2)
         c = xchg (val, 2);
       while (c != 0) {
         futex_wait (&val, 2);
         c = xchg (val, 2);
       }
     }
   }
   void unlock () {
     if (atomic_dec (val) != 1) {
       val = 0;
       futex_wake (&val, 1);
     }
   }
  private:
   int val;
};
```

Cena při kolizi

		mutex2	mutex3
lock	atomic op	$\frac{2}{3} + \frac{1}{2}$	$\frac{1}{2} + 1$
	futex syscall	$1 + 1$	$1 + 1$
unlock	atomic op	1	1
	futex syscall	1	1

Pokračování o driverech a jádře

- Embedded Linux driver development

  http://free-electrons.com/docs/kernel