PB173 Linux

12 Pamäť

Roman Lacko xlacko1@fi.muni.cz

2022-12-09

Obsah

Správa pamäte

Správa pamäte

Fyzická pamäť

- Elektronické čipy v PC
- Lineárne adresovaná
- Môže mať rôzne zóny

Virtuálna pamäť

- Abstrakcia nad fyzickou pamäťou
- Proces vidí len mapované časti

Správa pamäte: MMU

Memory Management Unit (MMU)

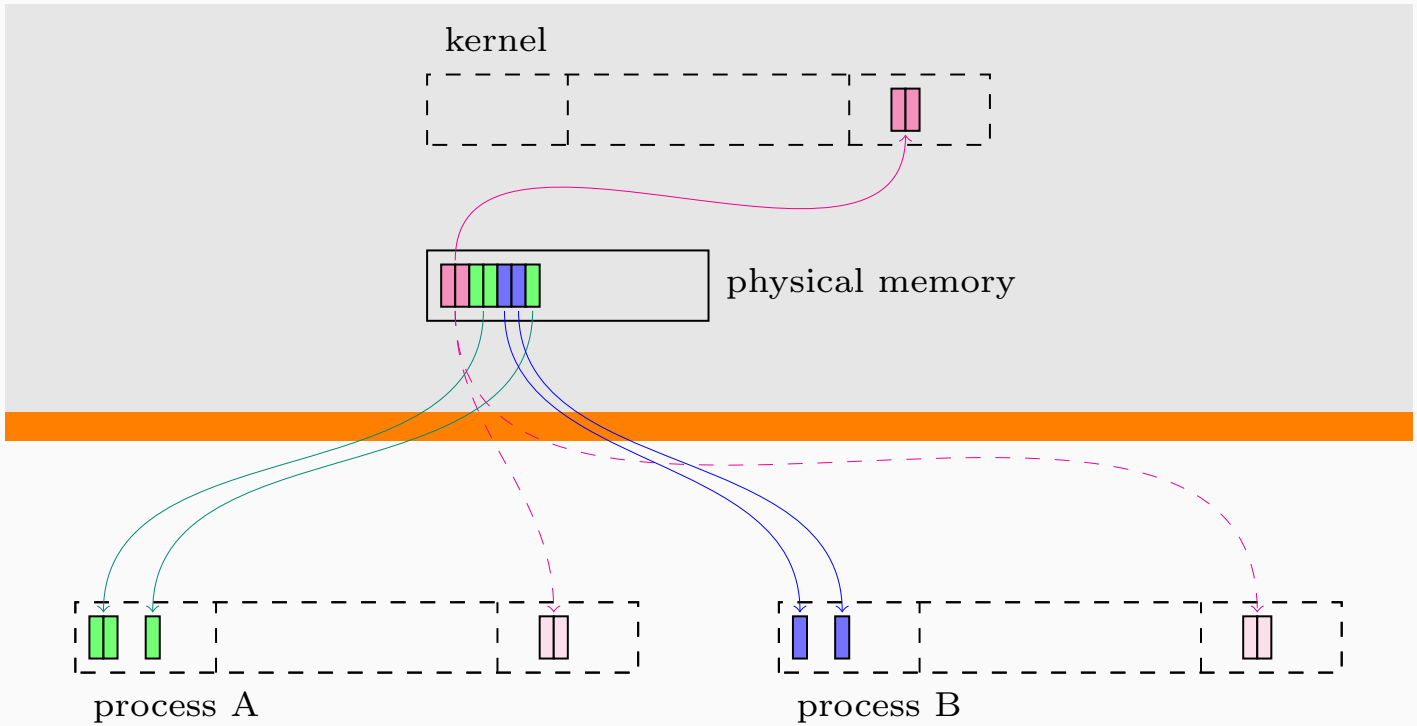
- Preklad medzi adresami fyzickej a virtuálnej pamäte
- *Page Table*
- *Translation Lookaside Buffer*

Stratégie prideľovania pamäte

- *Preallocation*
- *On-Demand Paging*
  - Page Fault
  - Out of Memory Killer
- Hybridné schémy a rozšírenia

Správa pamäte: OOM

Správa pamäte



Prideľovanie pamäte

Alokácia pamäte (C99, C11, POSIX)

```c
#include <stdlib.h>

void *malloc(size_t size);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);

/* C11 */
void *aligned_alloc(size_t alignment, size_t size);

/* Obsolete:
 * • posix_memalign(), memalign()
 * • valloc(), pvalloc() */
```

Prideľovanie pamäte

Alokácia pamäte (Linux)

- Posunitie zarážky `.data` segmentu

```c
#include <unistd.h>

int brk(void *addr);
void *sbrk(intptr_t increment);
```

- Explicitná zmena mapovania pamäte (po reklame)

Mapovanie pamäte

Mapovanie pamäte

Mapovanie pamäte

- Virtuálna pamäť procesu je rozdelená na stránky
- Asociácia medzi súborom a regiónom pamäte

Linux

- *Anonymous Mappings*
- *File-Backed Mappings*
- `/proc/${PID}/maps`

Mapovanie pamäte

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
#define MAP_FAILED ((void*) -1)

/* prot: */
#define PROT_{EXEC,READ,WRITE,NONE}  /* … */

/* flags: */
#define MAP_SHARED    /* … */
#define MAP_PRIVATE   /* … */

#define MAP_ANONYMOUS /* … */
#define MAP_POPULATE  /* … */
#define MAP_LOCKED    /* … */
```

Mapovanie pamäte: `MAP_FILE`

Mapovanie súboru

- Stránka pamäte $\leftrightarrow$ blok súboru
- `prot` musí byť kompatibilné s `mode` pre `open()`
- *On-Demand* sprístupnenie (ale `MAP_POPULATE`)
- Viditeľnosť zmien v súbore závisí na režime:

  `MAP_SHARED` Zmeny sa po čase (alebo po `msync()`) prejavia

  `MAP_PRIVATE` Kópia stránok len pre proces

Mapovanie pamäte: MAP_FILE

Zmena veľkosti mapovania

```
#include <sys/mman.h>

void *mremap(void *old_addr, size_t old_size, size_t new_size,
             int flags, /* void *new_address */);

#define MREMAP_MAYMOVE   /* … */
#define MREMAP_DONTUNMAP /* … */
```

Zmena veľkosti súborového mapovania

Viď systémové volanie `ftruncate()` (10 Súbory).

Mapovanie pamäte: `MAP_FILE`

Vynútenie synchronizácie

```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);

/* flags: */
#define MS_ASYNC      /* … */
#define MS_SYNC       /* … */

#define MS_INVALIDATE /* … */
```

Porovnajte s `fsync()`

Mapovanie pamäte

Prístupové práva

```
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);
```

Prenositeľné sú len niektoré kombinácie:

- `PROT_NONE`
- `PROT_READ`
- `PROT_READ | PROT_WRITE`

Mapovanie pamäte

Operácie súvisiace so swap

```c
#include <sys/mman.h>

int mincore(void *addr, size_t length, unsigned char vec[]);
// sizeof(vec) >= (length + PAGE_SIZE - 1) / PAGE_SIZE

int mlock(const void *addr, size_t len);
int mlock2(const void *addr, size_t len, int flags);
int munlock(const void *addr, size_t len);

int mlockall(int flags);
int munlockall(void);
```

Toto nie je mechanizmus medzivláknovej synchronizácie!

Mapovanie pamäte

Optimalizácia prístupu

```c
#include <sys/mman.h>

int madvise(void *addr, size_t len, int advice);

#define MADV_NORMAL     /* … */
#define MADV_RANDOM     /* … */
#define MADV_SEQUENTIAL /* … */
#define MADV_WILLNEED   /* … */
#define MADV_DONTNEED   /* … */ /* → MAP_ANONYMOUS */

int posix_madvise(void *addr, size_t len, int advice);
#define POSIX_MADV_*    /* … */
```

Mapovanie pamäte: MAP_ANONYMOUS

MAP_ANONYMOUS

- Mapovanie bez súboru
- Vynulované stránky
- De facto alokácia pamäte

Prenositeľné správanie vyžaduje `fd = -1` a `offset = 0`.

SunOS 4.0 (1988) bez MAP_ANONYMOUS

```
int zero_fd = open("/dev/zero", O_RDWR);
void *mem = mmap(NULL, length, PROT_READ | PROT_WRITE,
                 MAP_PRIVATE, zero_fd, 0);
```

Mapovaná pamäť

Anonymná pamäť ako deskriptor

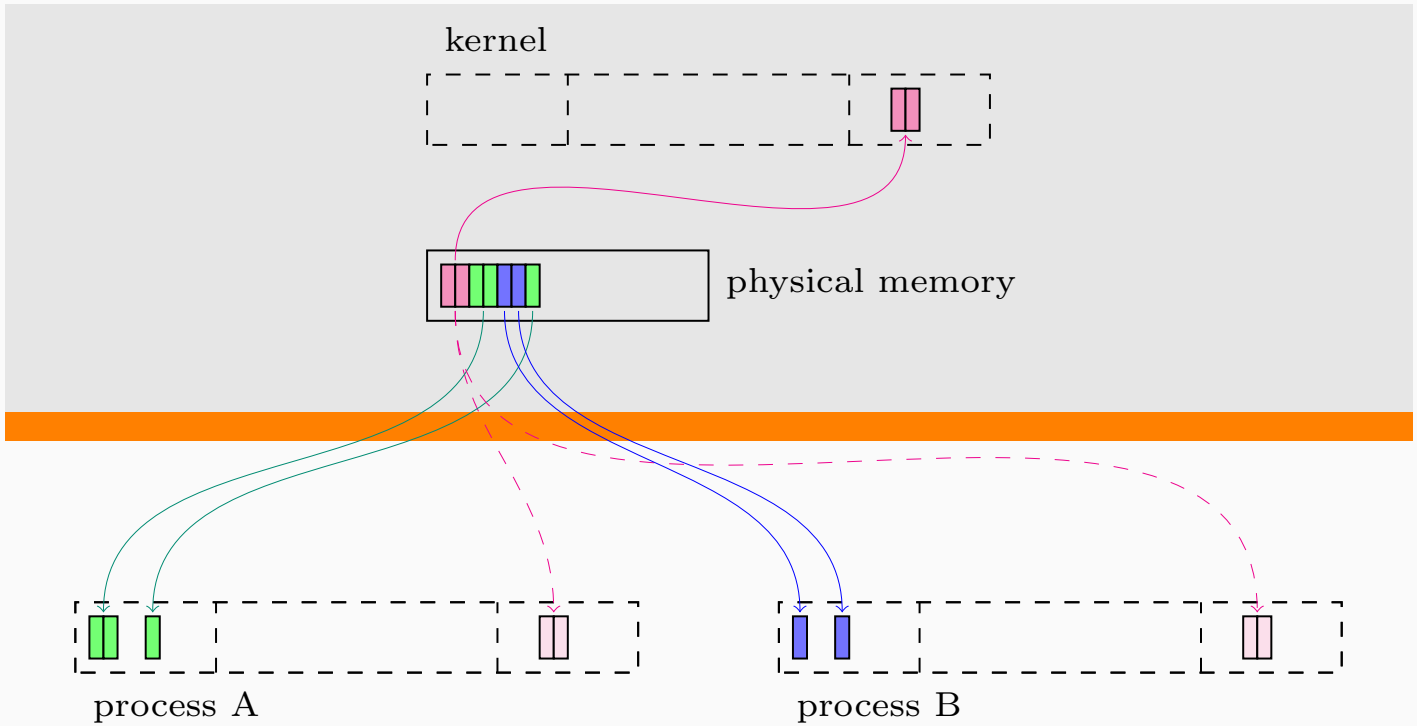
```
int memfd_create(const char *name, unsigned int flags);

#define MFD_CLOEXEC /* … */
```

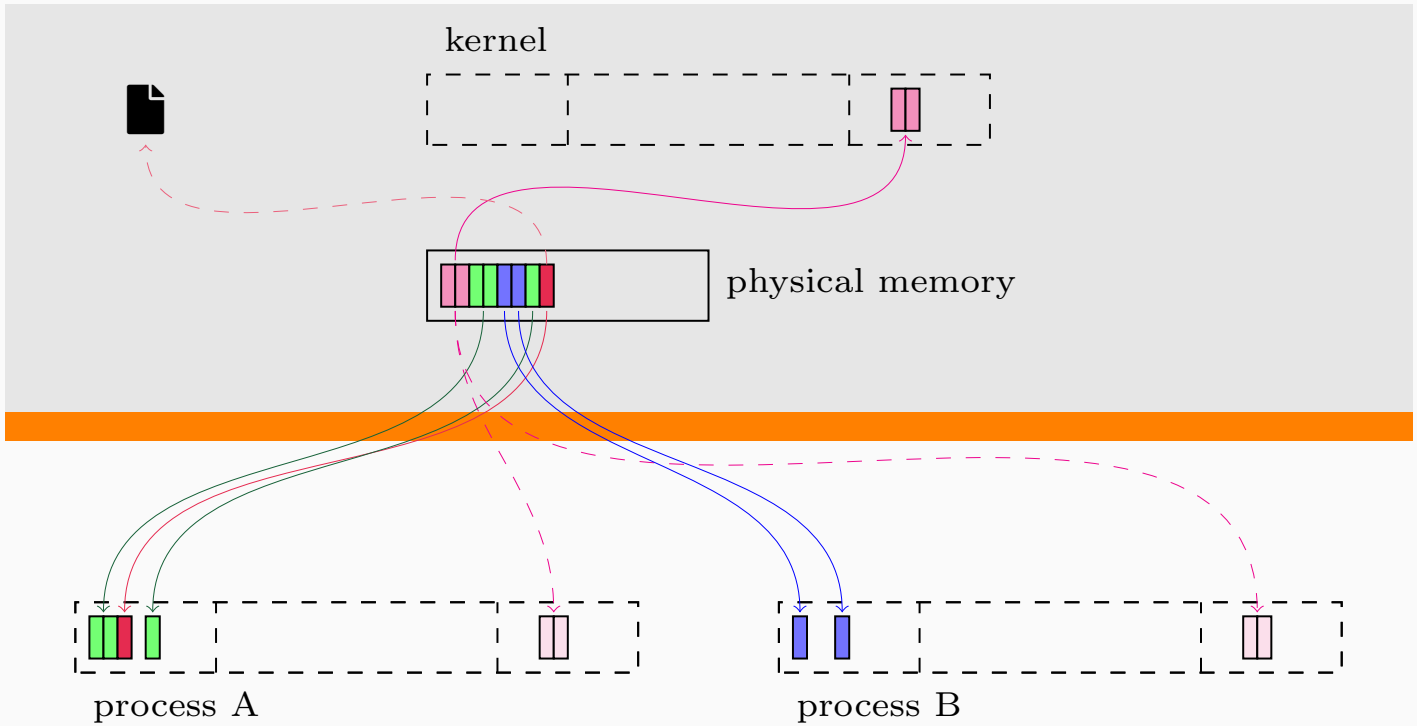
- Ekvivalent anonymného mapovania
- Zväčšuje sa podľa potreby
- Iniciálne `O_RDWR`, zmena cez `fcntl()`
- `name` nemá prakticky žiadny význam
- `ftruncate()`, `read()`, `read()`

Zdieľaná pamäť

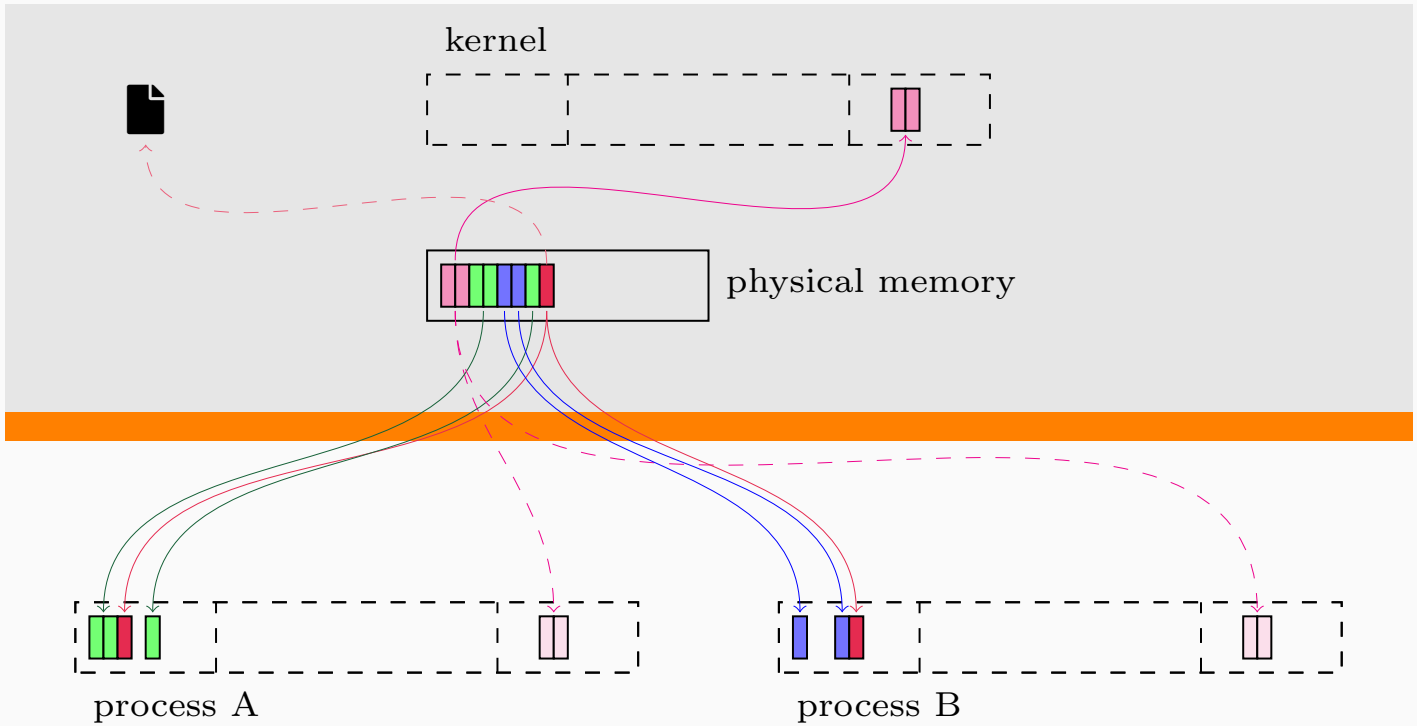
Zdieľaná pamäť: Princíp



Zdieľaná pamäť: Princíp



Zdieľaná pamäť: Princíp



Zdieľaná pamäť

Zdieľanie mapovaného súboru

- `MAP_SHARED`
- Veľmi rýchle IPC
- Nutná synchronizácia prístupu
  - `mutex`, `rwlock`, `eventfd` (príbuzné procesy)
  - Súborové zámky, SysV semafory (všeobecne)

Synchronizačné prvky pre nepríbuzné procesy:

- `pthread_mutexattr_setpshared()`
- `ptherad_rwlockattr_setpshared()`
- `sendmsg()` s `SCM_RIGHTS` (`man 7 unix`) a `eventfd()`

Zdieľaná pamäť

Zdieľanie anonymného regiónu

- `MAP_SHARED | MAP_ANONYMOUS`
- Zdieľaný len s príbuznými procesmi

Zdieľaná pamäť

Zdieľanie pomenovaného regiónu bez súboru

```
#include <sys/mman.h>

int shm_open(const char *name, int oflag, mode_t mode);
int shm_unlink(const char *name);
```

- `name` je logický identifikátor zdieľaného segmentu.
  Mal by byť tvaru `/name`.
- `oflag` a `mode` ako pre `open()`
  `O_WRONLY` použiť nejde
- `ftruncate()` zmení veľkosť segmentu
- `mmap()` pripojí segment do pamäte procesu

Ďalšie možnosti

userfaultfd

```c
#include <linux/userfaultfd.h>

int syscall(SYS_userfaultfd, int flags);
```

- Deskriptor na udalosti *Page Fault*
- Konfigurácia cez `ioctl()`
- `read()` vracia informácie o výpadkoch stránok

SysV IPC

- `man 7 ipc`
- Zdieľaná pamäť (POSIX: `man 7 shm_overview`)
- Semafory (`man 7 sem_overview`)
- Správy (`man 7 mq_overview`)

CLI nástroje: `ipcs`, `ipcmk`, `ipcrm`

SysV IPC: Zdieľaná pamäť

```c
#include <sys/ipc.h>

key_t ftok(const char *pathname, int proj_id);

int shmget(key_t key, size_t size, int shmflg);
#define IPC_PRIVATE /* … */
#define IPC_CREAT   /* … */

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
void *shmat(int shmid, const void *shmaddr, int shmflg);
int shmdt(const void *shmaddr);
```

SysV IPC

Zdieľaná pamäť

- Staršie a menej pohodlné rozhranie než POSIX
- Stále používané

Semafory, **Správy**

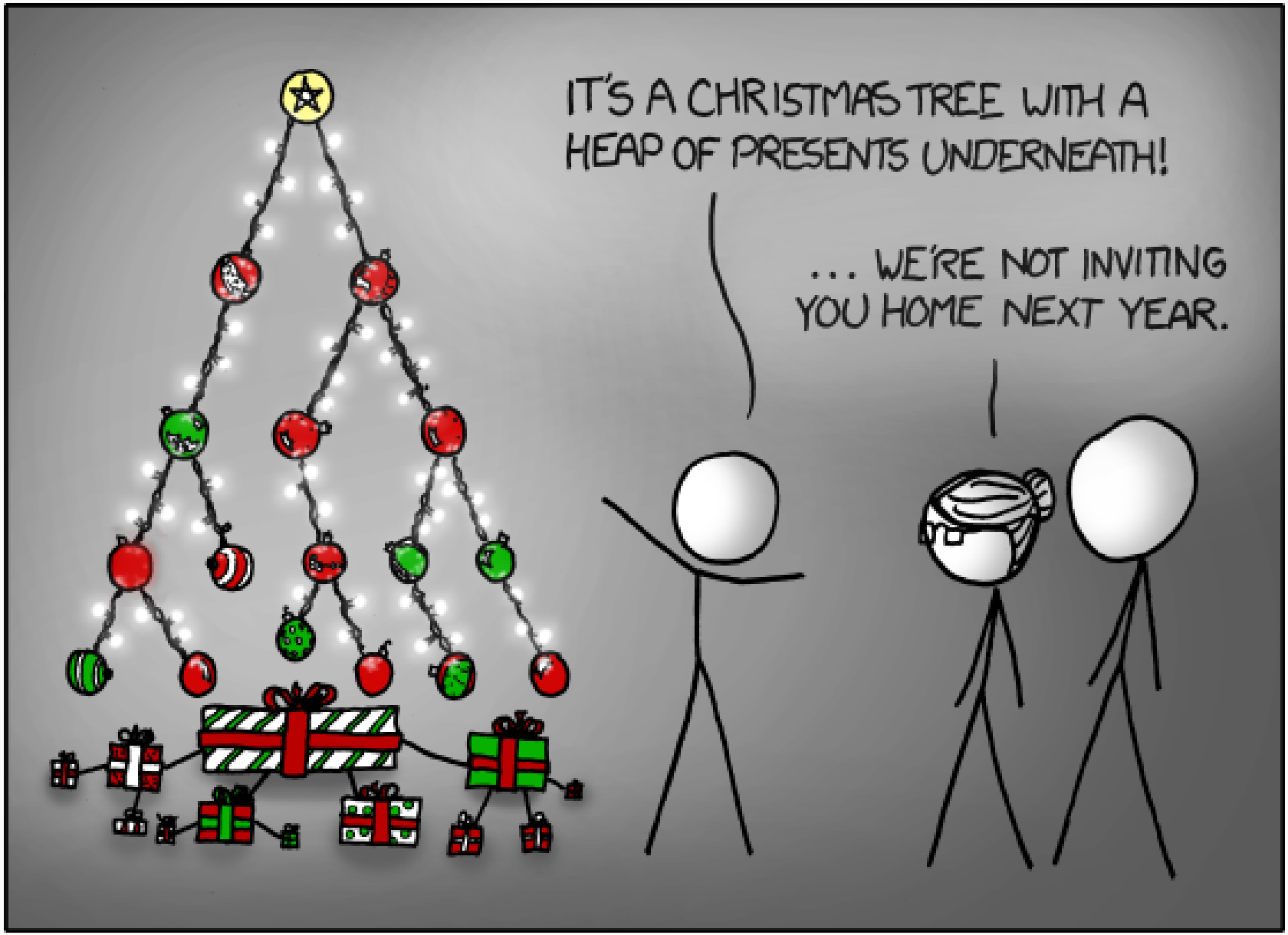
- Takmer nepoužívané

Záver

Zdroje

- Linux Memory Management

Záver


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.