



Laborator 3

Kernel API

Sisteme de Operare 2 (SO2)

Departamentul de Calculatoare

Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

- ▶ Alocarea memoriei
- ▶ Locking
- ▶ Contexte și sincronizări
- ▶ Liste

Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

- ▶ rezidentă
- ▶ swapabilă

kmalloc

- ▶ `void * kmalloc (size_t size, int flags);`
- ▶ alocă numai memorie rezidentă
- ▶ flags : GFP_KERNEL, GFP_ATOMIC

kfree

- ▶ `void * kfree (const void * mem);`

Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

spinlock

- ▶ `spinlock_t`
- ▶ `spin_lock_init()` – inițializare
- ▶ `spin_lock()` – obținere lock
- ▶ `spin_unlock()` – eliberare lock

semafor

- ▶ `struct semaphore`
- ▶ `sema_init()` – inițializare
- ▶ `down()` – decrementare
- ▶ `up()` – incrementare

- ▶ `atomic_t`
- ▶ `atomic_set()`
- ▶ `atomic_add()`
- ▶ `atomic_sub()`
- ▶ `atomic_inc()`
- ▶ `atomic_dnc()`

- ▶ Nu voi apela funcții care pot face sleep în interiorul unei regiuni critice protejate de un spinlock...
- ▶ Nu voi apela funcții care pot face sleep în interiorul unei regiuni critice protejate de un spinlock...
- ▶ Nu voi apela funcții care pot face sleep în interiorul unei regiuni critice protejate de un spinlock...
- ▶ Nu voi apela funcții care pot face sleep în interiorul unei regiuni critice protejate de un spinlock...
- ▶ Nu voi apela funcții care pot face sleep în interiorul unei regiuni critice protejate de un spinlock...



Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

Context proces

- ▶ cod kernel care rulează în numele unui proces ce a făcut un apel de sistem
- ▶ poate fi preemptat

Context întrerupere

- ▶ cod kernel care rulează un ISR
- ▶ nu poate fi preemptat, drept urmare:
 - ▶ nu poate face sleep
 - ▶ nu poate folosi mutex
 - ▶ accesa memorie virtuală din user-space
- ▶ codul trebuie să fie rapid

Presupunem o secțiune critică în următoarele condiții

- ▶ accesată doar în context proces, un singur procesor, fără preemptivitate

Presupunem o secțiune critică în următoarele condiții

- ▶ accesată doar în context proces, un singur procesor, fără preemptivitate
- ▶ *nu necesită sincronizare*

Presupunem o secțiune critică în următoarele condiții

- ▶ accesată doar în context proces, un singur procesor, fără preemptivitate
- ▶ *nu necesită sincronizare*
- ▶ accesată în context proces și context întrerupere, un singur procesor, fără preemptivitate

Presupunem o secțiune critică în următoarele condiții

- ▶ accesată doar în context proces, un singur procesor, fără preemptivitate
- ▶ *nu necesită sincronizare*
- ▶ accesată în context proces și context întrerupere, un singur procesor, fără preemptivitate
- ▶ *dezactivarea întreruperilor la intrarea în regiunea critică*

- ▶ accesată în context proces și context întrerupere, un singur procesor, cu preemptivitate

- ▶ accesată în context proces și context întrerupere, un singur procesor, cu preemptivitate
- ▶ *dezactivarea preemptivității și a întreruperilor la intrarea în regiunea critică*

- ▶ accesată în context proces și context întrerupere, un singur procesor, cu preemptivitate
- ▶ *dezactivarea preemptivității și a întreruperilor la intrarea în regiunea critică*
- ▶ accesată în context proces și context întrerupere, mai multe procesoare, cu preemptivitate

- ▶ accesată în context proces și context întrerupere, un singur procesor, cu preemptivitate
- ▶ *dezactivarea preemptivității și a întreruperilor la intrarea în regiunea critică*
- ▶ accesată în context proces și context întrerupere, mai multe procesoare, cu preemptivitate
- ▶ *dezactivare întreruperi și preemptivitate pe procesorul local, funcționalitate spinlock completă*

Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

- ▶ struct list_head
- ▶ list_add()
- ▶ list_del()
- ▶ list_for_each()
- ▶ list_for_each_safe()

Introducere

Alocarea memoriei

Locking

Contexte și sincronizări

Liste

Keywords

- ▶ contexte de execuție
- ▶ printk
- ▶ kmalloc, kfree
- ▶ spinlock_t
- ▶ struct semaphore
- ▶ atomic_t