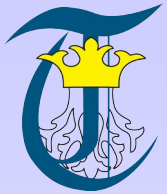


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare distribuite și în timp real

– curs –
master SECI anul I

ș.l. dr. ing. Kertész Csaba-Zoltán

Linux și real-time

- sistem timp-real:
  - răspunsul sistemului la apariția unui eveniment este realizat într-un timp corespunzător
- clasificare:
  - hard real-time
    - sistemul garantează un timp de răspuns pentru cel mai rău caz, care este realizat în toate condițiile
  - soft real-time
    - sistemul nu garantează timpul de răspuns pentru orice condiție, dar în marea majoritate a cazurilor, răspunsul vine în timp corespunzător

Standardul POSIX pentru RTOS

- un sistem RTOS trebuie să satisfacă următoarele caracteristici
  - multitasking / multithreading
    - RTOS trebuie să ofere suport pentru rularea aplicațiilor pseudoparalel
  - priorități
    - taskurile trebuie să aibă asociate priorități diferite, taskurile critice având priorități mai ridicate
  - moștenire de priorități
    - RTOS trebuie să suporte un mecanism de moștenire a priorităților

Standardul POSIX pentru RTOS

- preemption
  - când un task de prioritate mai mare devine ready, taskul de prioritate mai mică în rulare trebuie întrerupt
- latența întreruperilor
  - RTOS trebuie să aibă o latență a întreruperilor (timpul între apariția întreruperii și deservirea acesteia) predictibilă (și cât mai mică)
- latența planificatorului
  - timpul între momentul când un task devine rulabil și până când începe să ruleze trebuie să fie determinist

Standardul POSIX pentru RTOS

- IPC și sincronizare
  - cea mai populară metodă de IPC în sisteme embedded este message passing, RTOS trebuie să ofere suport pentru transmisia mesajelor în timp constant
  - RTOS trebuie să ofere suport pentru semafoare și mutex
- alocare dinamică de memorie
  - RTOS trebuie să ofere un mecanism de alocare dinamică a memoriei într-un timp constant

Natura non-RTOS a Linuxului

- Linux a evoluat ca un SO de uz general, și în consecință are următoarele caracteristici:
  - latență mare la întreruperi
  - latență mare a planificatorului datorită naturii non-preemptive a kernelului
  - diferite servicii a SO (IPC, alocare dinamică de memorie, etc.) nu au un timp de execuție determinist
  - alte caracteristici (ca memoria virtuală, apelurile sistem) rezultă în timpi de răspuns a SO în general non-deterministe

Modificare Linux pentru real-time

- două soluții adoptate
  - realizarea unui sistem hard real-time
    - RTLinux
    - se folosește un real-time executive (kernel simplu RTOS) care satisface toate criteriile RTOS și rulează Linux ca un singur task
  - adăugare suport în kernel pentru execuții soft real-time
    - uCLinux, MontaVista, etc...
    - soluții ce modifică timpii de răspuns a sistemului
    - cele mai multe au fost portate în kernelul 2.6

Linux soft real-time

- timpii de răspuns generale a sistemului sunt reduse în așa fel încât în majoritatea cazurilor să ofere răspunsuri la evenimente în timp corespunzător
- pentru tratarea unui eveniment sistemul trebuie să realizeze o multitudine de operații asociate diferitelor subsisteme între apariția evenimentului și lansarea procesului care răspunde la eveniment
  - toate aceste operații trebuie optimizate pentru ca timpul de răspuns general să fie corespunzător

Tratarea evenimentelor

- evenimentul cauzează o întrerupere la procesor
- se rulează ISR din driverul corespunzător
- ISR verifică în wait-queue dacă există un proces care așteaptă evenimentul
- dacă se găsește astfel de procese se rulează o funcție wake-up care scoate procesul din coada wait și depune în coada ready
- kernelul apelează planificatorul la primul moment în care aceasta este posibil
- planificatorul va schimba la procesul respectiv dacă aceasta are suficientă prioritate

Caracteristici OS ce influențează timpul de răspuns

- latența întreruperilor
  - timpul până când ISR este executat
- durata ISR
- latența planificatorului
  - timpul după terminarea ISR până la rularea planificatorului
- durata planificării

Latența întreruperilor

- principale cauze de latență mare
  - dezactivarea întreruperilor pentru un timp îndelungat
    - porții din kernel și din drivere trebuie să se protejeze față de rutine de întreruperi
    - astfel de secțiuni critice sunt înconjurate de apeluri de tip `local_irq_disable` sau `spin_lock_irq`
  - înregistrarea întreruperilor greșită
    - driverele scrise greșit pot să înregistreze ISR-ul în mod greșit
    - există două moduri de înregistrare: fast irq și slow irq, dacă un dispozitiv mai puțin important înregistrează fast irq atunci poate bloca întreruperile mai importante

Durata ISR-urilor

- durata ISR-ului este în totalitate lăsată pe seama programatorului de driver
- durata ISR-ului poate fi non-deterministă dacă se folosește soft-irq
  - ISR poate fi împărțit în două secțiuni: o parte critică care se execută în interiorul ISR și nu mai poate fi întrerupt și o parte care va fi tratat ulterior de kernel și care poate fi întrerupt de alte irq hardware
  - soft-irq este de obicei tratat de daemonul ksoft-irqd care este non-real-time, astfel folosirea soft-irq pentru evenimente real-time trebuie evitată

Latența planificatorului

- latența planificatorului (mai ales la versiuni mai vechi) este principalul contributor la timpul de răspuns mare
- kernelul până la 2.4 avea o natură non-preemptivă:
  - procesele din user-space puteau fi înlăturate, dar funcțiile kernel apelate din user-space nu puteau fi întrerupte, astfel se crea și în interiorul proceselor zone în care procesul nu putea fi întrerupt
  - dezactivarea întreruperilor (ce include și întreruperea timerului asociat cu planificatorul) putea avea impact asupra latența planificatorului

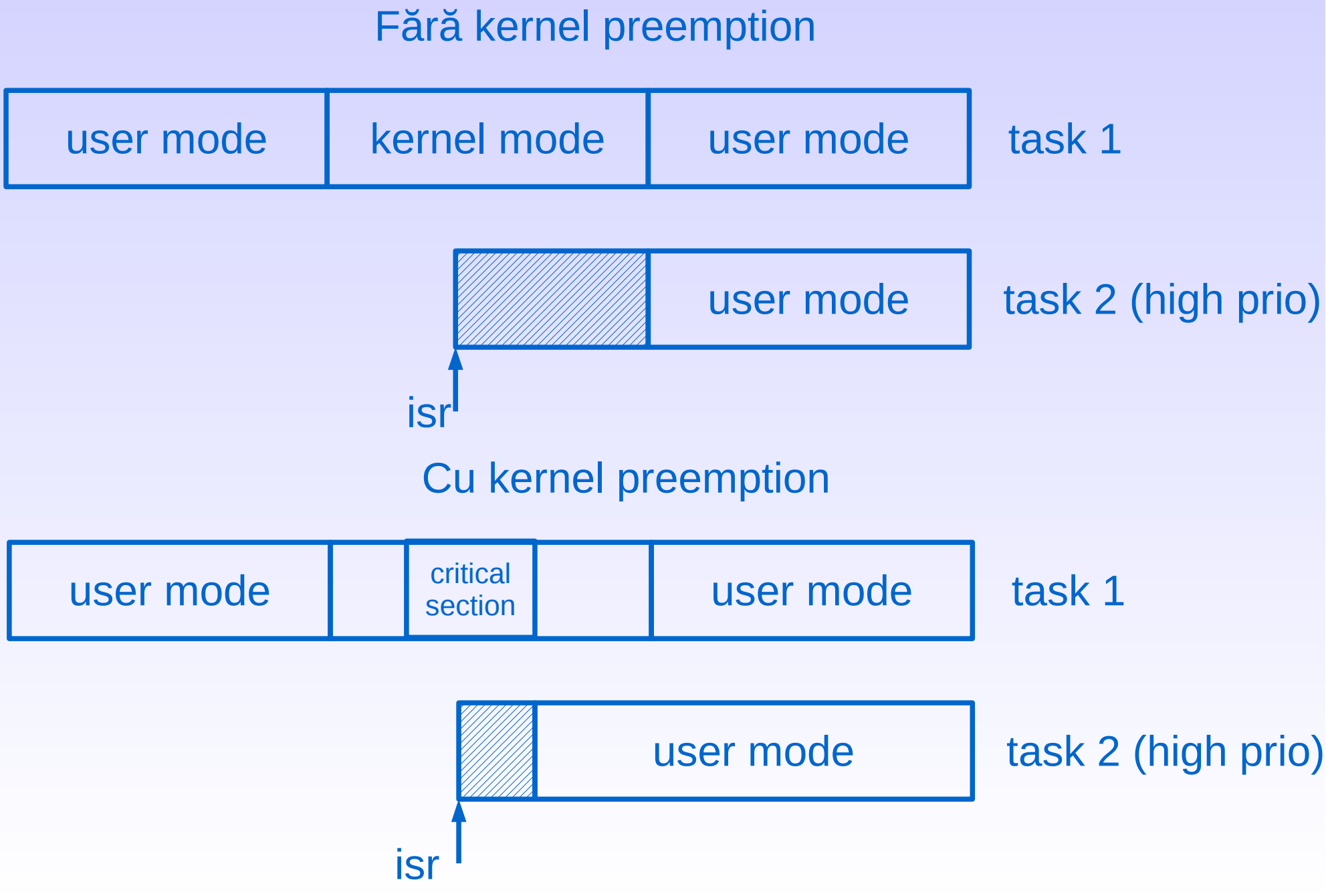
Kernel preemption

- odată cu creșterea suportului pentru SMP în Linux au fost identificate și un număr mare de secțiuni critice care au fost protejate cu spinlock
- s-a observat că funcțiile din kernel puteau fi înlăturate la planificare fără probleme dacă acestea nu se află în secțiuni critice
- pentru Montavista a fost creat un patch care folosește acest lucru și permite înlăturarea kernelului la planificare
  - acest patch a fost mai târziu reintrodus în kernelul 2.6

Kernel preemption

- pentru suportul kernel preemption a fost introdus o variabilă `preempt_count` și două primitive:
  - `preempt_disable`: incrementează `preempt_count`
  - `preempt_enable`: decrementează `preempt_count`
  - dacă `preempt_count` este 0 atunci kernelul poate fi înlăturat fără probleme
- toate rutinele spinlock au fost modificate pentru a apela `preemt_disable` la intrare și `preempt_enable` la ieșire
- la isr-uri a fost adăugată o funcție care verifică `preempt_count` și apelează planificatorul dacă aceasta este 0

Latența planificatorului



Patchuri de scădere a latenței

- pe parcursul dezvoltări kernelului 2.6 au fost adăugate și niște patchuri pentru micșorarea latenței (mai ales de Ingo Molnár și Andrew Morton)
  - s-a adăugat puncte de planificare explicite în interiorul blocurilor kernel care se execută mai îndelungat
  - deseori (mai ales la parcurgerea listelor lungi) sa introdus puncte unde spinlock-ul a fost oprit, apelat planificatorul și restabilit spinlock-ul
- aceste patchuri deseori trebuie descărcate separat de kernel, dar dau rezultate foarte bune

Durata planificării

- durata planificării este timpul necesar pentru planificator ca să aleagă procesul care va fi activat, să modifice listele (run, ready) corespunzătoare și să-l schimbe contextul la noul proces
- inițial planificatorul din Linux (pornind cu conceptul de SO de uz general) s-a concentrat pentru a realiza planificare cât mai corectă, de aceea a fost sacrificat durata deterministă a planificării
  - de obicei planificatorul rula în timp O(n) pentru numărul proceselor

Planificatorul O(1)

- începând de versiunea 2.4.20 a fost introdus în kernel planificatorul O(1) de Ingo Molnár
- acest planificator permite alegerea procesului ce trebuie luat într-un timp constant față de numărul proceselor
  - sunt implementate 2 cozi: procesele active și procesele expirate
  - cozile sunt ordonate în funcție de priorități
  - indexarea cozilor este făcut într-un bitmap,astfel căutarea procesului cel mai prioritar devine O(1)
  - la folosirea cuantei de timp asociate procesul activ va fi depus în lista proceselor expirate
  - la golirea listei active cele două liste se inversează

User-space real-time

- kernelul Linux (mai ales începând din 2.6) datorită planificatorului O(1) și patchurilor de scădere a latenței a devenit un SO soft real-time
- pentru ca întregul sistem să satisfacă criteriile de real-time și aplicațiile trebuie să se ruleze într-o manieră deterministă
- pentru aceasta Linux (începând cu versiune 2.6) oferă suport pentru extensiile POSIX real-time

Extensii POSIX real-time

- planificare cu priorități fixe
- blocarea memoriei
- cozi de mesaje POSIX
- memorie partajată POSIX
- semnale real-time
- semafoare POSIX
- timere POSIX
- operații IO asincrone

Extensii POSIX real-time

- prioritățile fixe, memory lockingul, memoria partajată și semnale real-time au fost suportate de Linux chiar de la începuturi
- cozile de mesaje și timere POSIX au fost introduse în kernelul 2.6
- operații IO asincrone inițial a fost oferite numai în bibliotecile libc, dar începând cu 2.6 treptat și aceste operații au fost incluse în kernel
- la momentul actual Linux oferă tot suportul pentru extensii POSIX real-time, rămânând la latura dezvoltatorilor să folosească acestea

Planificare cu priorități fixe

- parametrii asociați proceselor (referitor la RT):
  - clasa de planificare
  - prioritate
  - cuanta de timp
- clasa de planificare precizează algoritmul de planificare folosit pentru proces:
  - `SCHED_FIFO`: procesele rulează unul după celălalt în ordinea creării până la terminare sau blocare
  - `SCHED_RR`: (Round Robin) procesele rulează până la cuanta precizată după care vor fi puse la capătul listei
  - `SCHED_OTHER`: planificatorul standard (non-RT)

Priorități specificate de utilizator

- utilizatorul poate modifica prioritatea asociată procesului (sched_setparam)
- clasele de planificare SCHED_FIFO și SCHED_RR întotdeauna au o prioritate mai mare decât clasa SCHED_OTHER (prioritate fixată pe 0)
- clasa SCHED_OTHER este tratată cu algoritm standard de planificare pe baza unei valori nice
- valoarea nice nu are efect în cazul planificărilor FIFO și Round Robin, însă contribuie la calcularea cuantei de timp (la Round Robin)

Blocarea memoriei

- aplicațiile real-time au nevoie de un timp de răspuns determinist ce nu poate fi satisfăcut dacă se folosește paginare
- pentru a realiza timpul de răspuns determinist un proces poate să blocheze o parte (sau chiar toată) memoria alocată procesului, ca această zonă să rămână definitiv în memorie
- soluții de blocare:
  - în timpul rulării cu funcțiile mlock și mlockall
  - în timpul legării prin intermediul ldscript-ului
  - cu atributul gcc: __section__

IPC POSIX

- cozi de mesaje
  - mesajele pot fi prioritizate, transmisie blocantă
  - mq_open, mq_close, mq_unlink, mq_send, mq_recv
- memorie partajată
  - cea mai rapidă soluție, acces fără apeluri sistem
  - shm_open, shm_unlink
- semafoare
  - sincronizare între procese
  - sem_open, sem_close, sem_unlink, sem_wait, sem_post

Semnale timp real

- sunt folosite pentru notificarea proceselor despre apariția unui eveniment asincron
- semnalele real-time sunt o extensie a semnalelor native Linux:
  - număr mai mare de semnale utilizator
  - semnale prioritizate
  - informații atașate semnalelor
  - semnalele sunt salvate într-o coadă
- funcții:
  - sigaction (definirea unui handler), sigqueue (transmiterea unui semnal)

Ceasuri și timere POSIX

- oferă acces mai avansat la ceasul sistem
  - CLOCK_REALTIME: ceasul universal
  - CLOCK_MONOTONIC: timpul trecut de la pornirea sistemului
  - CLOCK_PROCESS_CPUTIME_ID: timpul total a unui proces petrecut în rulare
  - CLOCK_THREAD_CPUTIME_ID: similar pentru threaduri
- accesul la ceas:
  - clock_settime, clock_gettime, clock_getres,
- blocarea procesul pentru o perioadă
  - clock_nanosleep

Timere

- folosite pentru temporizare
  - timer_create, timer_delete
- pot fi setate pentru o anumită perioadă după care vor genera un semnal ce poate fi tratat
  - timer_settime
- rezoluția standard în Linux este de 1ms
- MontaVista a introdus un timer de înaltă rezoluție (1µs)
  - CLOCK_REALTIME_HR
  - CLOCK_MONOTONIC_HR

Operaţii IO asincrone

- operațiile IO standard sunt blocante (read, write)
- pentru un timp de răspuns mai bun poate fi nevoie ca procesul să realizeze alte calcule în timp ce așteaptă pentru terminarea operației IO
- aio_read (citire), aio_write (scriere), aio_error (starea operaţiei), aio_return (numărul de octeți transferați), aio_cancel (oprirea operației), aio_suspend (blocarea procesului)
- operațiile nu țin la curent offsetul în fișier
- după terminarea operației IO procesul este notificat printr-un semnal

Operații IO asincrone

- operațiile AIO POSIX sunt oferite în Linux în user-space prin implementarea unor thread-uri
- kernelul 2.6 introduce suport pentru AIO la nivel kernel
- aceste AIO kernel vin cu un nou set de apeluri sistem (definite în libaio), care este diferit de apelurile POSIX
  - io_setup (crearea operației), io_submit (pornirea operației), io_getevents (citirea operațiilor terminate), io_wait (blocare proces până terminarea operației), io_cancel (oprirea operației), io_destroy (ștergerea contextului de operație)