Embedded Systems Laboratory

Catedra de Calculatoare

Facultatea de Automatica si Calculatoare

Sisteme Încorporate

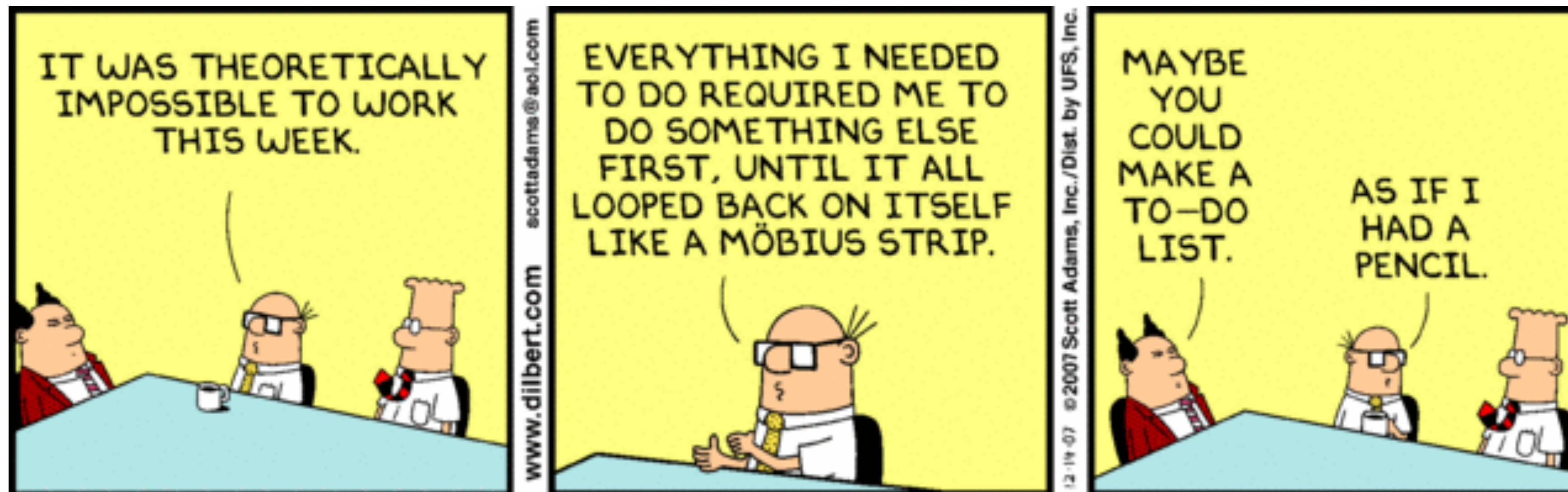
Cursul 9

Software de timp real

Sisteme de operare de timp real

Planificare

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București



http://dilbert.com/strips/comic/2007-12-14/

Definitie: Sisteme de timp real

Embedded Systems Laboratory

- Un sistem embedded care monitorizeaza, raspunde la stimuli sau controleaza mediul extern in timp real (sisteme reactive)
- Exemple:
  - Vehicule (automobil, avion, …)
  - Controlul traficului(autostrada, aerian, cai ferate, …)
  - Controlul proceselor (uzina electrica, chimica, …)
  - Sisteme medicale (terapia prin iradiere, …)
  - Telefonie, radio, comunicatii prin satelit
  - Jocuri de calculator

Caracteristici

- Constrangeri de timp/termen limita
  - Corectitudine temporala si functionala
- Hard deadline
  - Trebuie sa respecte termenul intotdeauna
  - Controller pentru traficul aerian
- Soft deadline
  - Trebuie sa respecte termenul frecvent
  - Decoder MPEG
- Concurenta (procese multiple)
  - Face fata la semnale multiple de intrare si iesire
- Fiabilitate
  - Cat de des se defecteaza sistemul
- Toleranta la defecte
  - Recunoasterea si tratarea erorilor si defectelor
- Sisteme Critice
  - Cost mare al unui defect
  - Sistem hard real time $\Rightarrow$ sistem critic

Program de timp real pe un microprocesor

Embedded Systems Laboratory

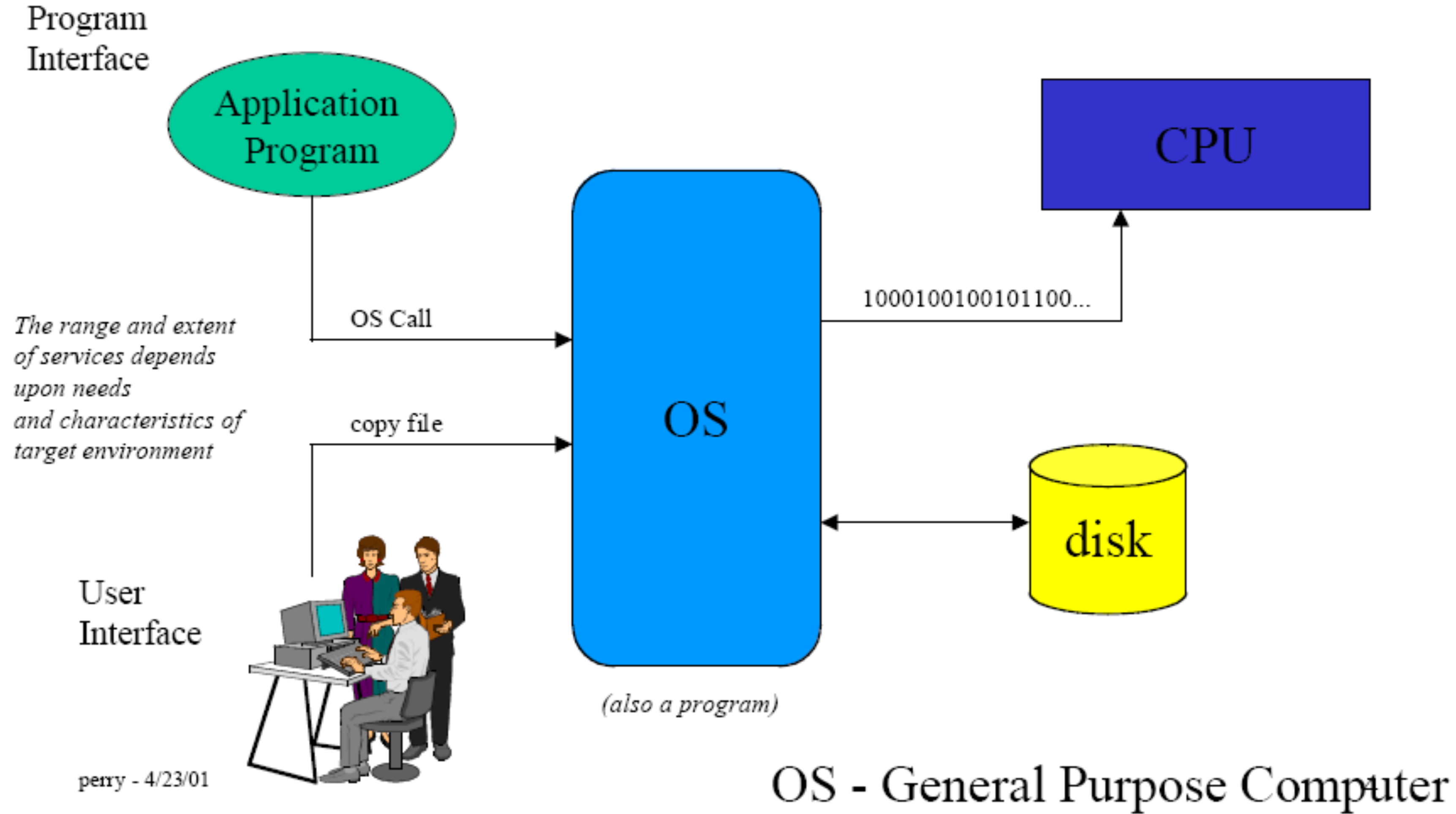
- Abordari tipice:
  - Sincron
    - O singura bucla de program
  - Asincron
    - Sistem foreground/background
    - Multitasking

Sisteme de operare

- Ce este un sistem de operare?

- O colectie organizata de extensii software a hardware-ului care indeplinesc urmatoarele functii:
  - Rutine de control pentru operarea sistemului (permit accesul la resursele calculatorului: file-system, I/O, memorie etc)
  - Un mediu pentru executia de programe

Serviciile unui SO

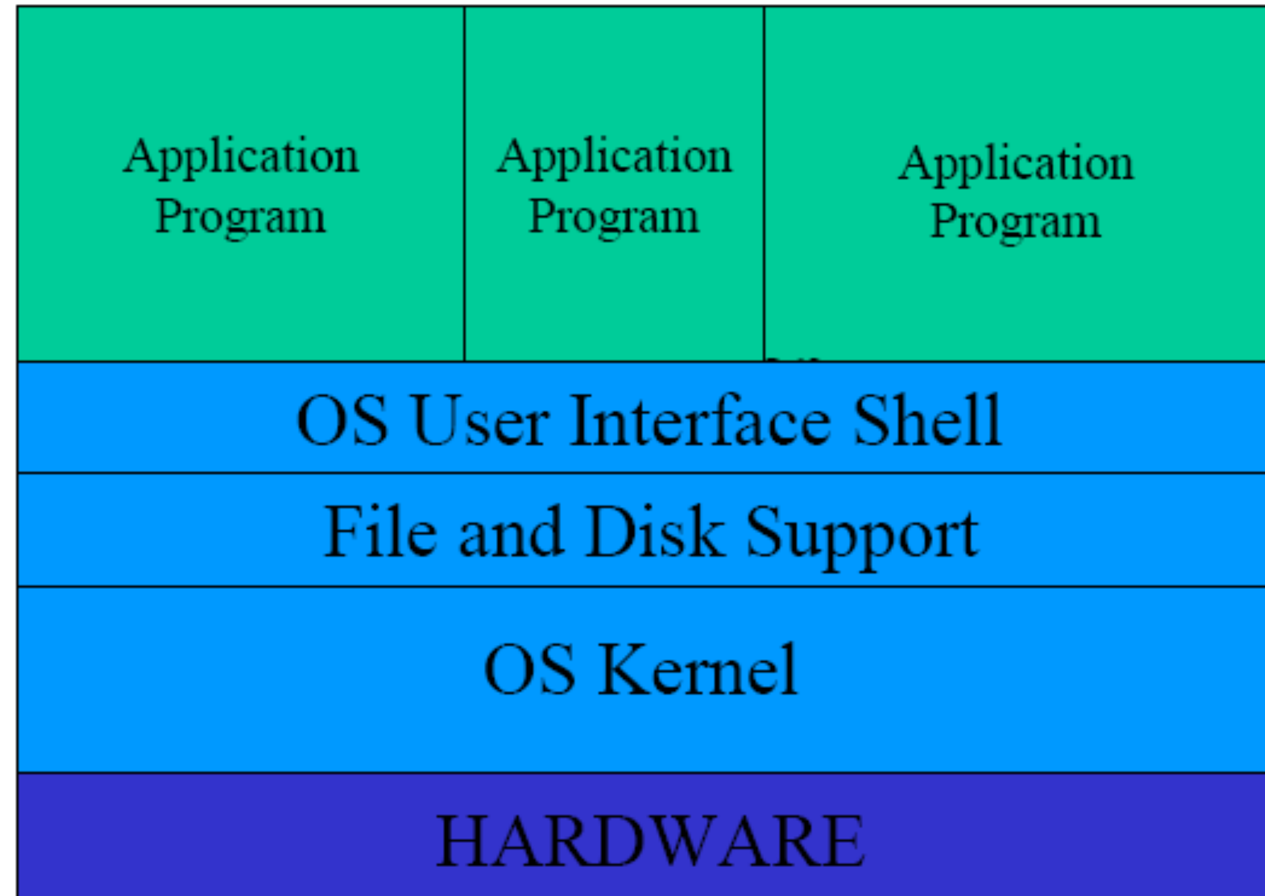


Sisteme de operare

- Ce face de fapt un sistem de operare?
- Administreaza resursele de sistem (procesor, memorie, I/O, etc.)
  - Tine evidenta asupra statusului si “proprietarului” fiecarei resurse
  - Decide cine primeste resursa
  - Decide cat de mult timp resursa poate fi alocata
- In sisteme cu executie concurenta
  - Arbitreaza si rezolva conflictele de resurse
  - Optimizeaza performantele in contextul utilizatorilor multipli
- Ganditi-va la un SO ca la proprietarul unei carti pe care toti studentii de la acest curs trebuie s-o citeasca
  - Care sunt problemele care apar?

SO – Structura Ierarhica

Embedded Systems Laboratory

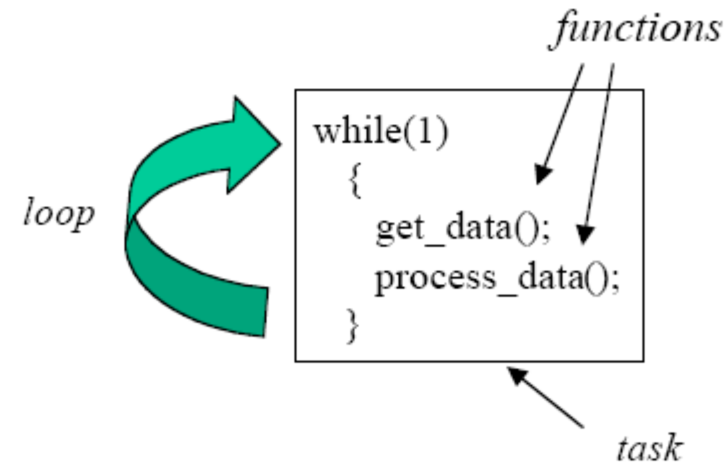


Sisteme de operare

- Tipuri de sisteme de operare
  - Cele mai simple = kernel de dimensiuni mici pe un procesor embedded
  - Complexe = SO comercial Full-Featured
    - Securitate
    - Utilizatori multipli
    - Suport grafic
    - Suport pentru retea
    - Drivere comunicatie cu o gama larga de periferice
    - Programe cu executie concurenta

Taskuri si Functii

- Un task este un proces repetitiv
  - Bucla infinita
  - Conceptul de baza in sistemele de operare de timp real (RTOS).



- O functie este o procedura care poate fi apelata. Aceasta ruleaza apoi intoarce o valoare la terminarea executiei.
  - process_data();
  - int add_two_numbers(int x, int y);

RTOS

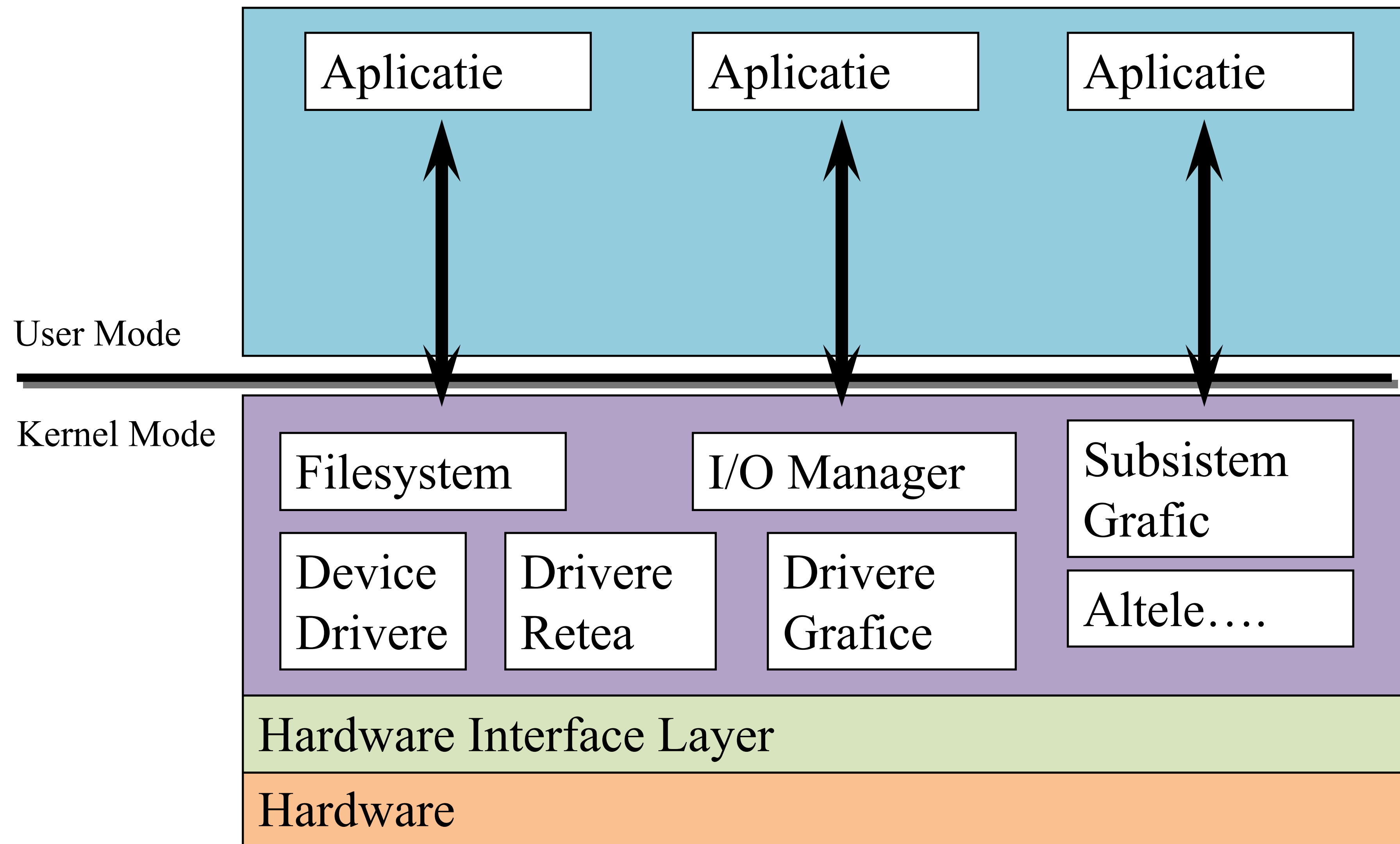
- In majoritatea cazurilor, RTOS = OS Kernel
  - Un sistem embedded este proiectat pentru un singur scop asa ca majoritatea functionalitatilor unui SO comercial sunt redundante (consola, interfata grafica, suport tastatura, mouse etc.).
  - RTOS permite controlul ferm asupra resurselor sistemului
    - Nu exista procese de background inutile
    - Numar maxim de task-uri care pot rula pe sistem
  - RTOS permite controlul temporizarii proceselor
    - Manipularea prioritatii task-urilor
    - Optiuni de setare a mecanismului de planificare

Implementarea in Real-Time OS

- Nuclee mici si rapide
- Extensii de timp real la sisteme de operare comerciale
- Sisteme de operare experimentale
- Parte din run-time-ul unor limbaje de programare
    - Java (embedded real-time Java)
- Kernel monolitic vs. Microkernel

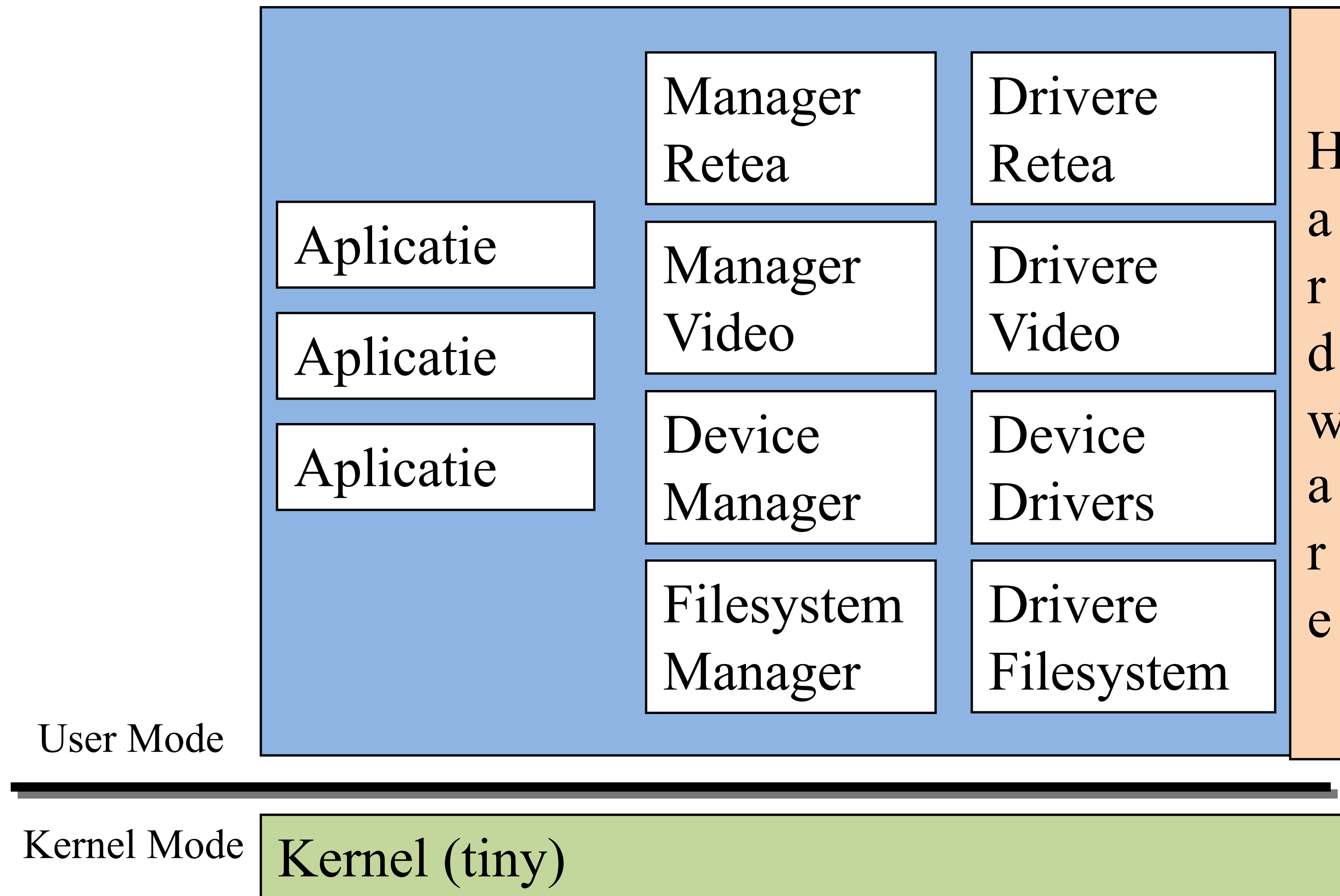
Organizarea unui kernel monolitic

Embedded Systems Laboratory



Organizarea unui microkernel

Embedded Systems Laboratory



RTOS - Kernel

Embedded Systems Laboratory

- Kernel-ul OS indeplineste 3 functii:
  - **Task Scheduler** : Determina care task va rula in cuanta de timp urmatoare pentru un sistem multitasking.
  - **Task Dispatcher**: Produce informatia necesara in contextul pornirii unui task
  - **Intertask Communication**: Implementeaza un mecanism de comunicatie intre doua procese

RTOS - Kernel

- Daca ne intoarcem la analogia cu cartea
  - **Task Scheduler**: Cine primeste cartea si cand?
  - **Task Dispatcher**: Managementul transportului cartii de la o persoana la alta
  - **Intertask Communication**: Ce se intampla daca un student vrea sa vorbeasca cu altul? Doar un singur student poate avea cartea la un moment dat.

RTOS Kernel

- Strategii de design pentru nucleul RTOS
  - Polled Loop Systems
  - Sisteme Interrupt Driven
  - Sisteme Foreground / Background
  - Multi-tasking
  - Full Featured RTOS

Polled Loop System

- Cel mai simplu nucleu real-time
- O singura instructiune repetitiva testeaza un flag care indica daca un eveniment s-a produs sau nu.
- Nu este nevoie de comunicatie intre task-uri sau mecanisme de planificare – avem un singur task.
- Se comporta excelent in cazul canalelor de viteza mare de transmisie a datelor.
  - Evenimentele se petrec la intervale de timp uniforme (si relativ mari)
  - Procesorul se ocupa numai de canalul de date

Exemplu

Embedded Systems Laboratory

- Un automat programabil care trebuie sa indeplineasca urmatoarele functii
  - La fiecare 20ms trebuie sa actualizeze ceasul sistemului
  - La fiecare 40ms ruleaza un modul de control
  - Inca trei module fara constrangeri puternice de timp
    - » Actualizarea ecranului operatorului
    - » Primirea de comenzi de la operator
    - » Inregistrarea unui istoric de evenimente si comenzi

Program cu o singura bucla de control

Embedded Systems Laboratory

```
while (1) {
    wait_clock_tick();
    if(time_for_clock) update_clock();
    if (time_for_control) do_control();
    else if (time_for display update) refresh_display();
    else if (time_for_input) get_input();
    else if (time_for_log) save_log();
}
```

- Trebuie ca: $t1 + max(t2, t3, t4, t5) \leq 20\ ms$
  - Se preteaza la programe simple, cu numar limitat de functii si constrangeri

Alt Exemplu


```
int main(void) {
    Init_All();
    for (;;) {
            IO_Scan();
            IO_ProcessOutputs();
            KBD_Scan();
            PRN_Print();
            LCD_Update();
            RS232_Receive();
            RS232_Send();
            TMR_Process();
    }
    // n-ar trebui sa ajunga aici
    printf("Eroare…");
return (0);
}
```

Observatii

- Fiecare functie apelata in bucla infinita se executa independent
- Fiecare functie trebuie sa-si inceteze executia dupa un timp rezonabil, indiferent de codul executat.
- Nu se stie frecventa la care se executa bucla principala
  - Frecventa poate varia in functie de evolutia sistemului si de starea curenta
- Bucla contine si functii periodice si functii executabile la anumite evenimente
  - Majoritatea task-urilor sunt event-driven
    - ex. IO_ProcessOutputs este event-driven
    - De obicei au asociate la intrare cozi de mesaje
      - Ex. IO_ProcessOutputs primeste evenimente de la IO_Scan, RS232_Receive si KBD_Scan cand o iesire trebuie activata
  - Celelalte sunt periodice
    - Nu raspund la evenimente dar pot avea perioade diferite si isi pot schimba in timp perioada de executie

Observatii (cont.)

Embedded Systems Laboratory

- Trebuie implementate niste metode simple de comunicatie inter-task
  - Ex. Vrem sa nu mai citim intrarile dupa ce s-a apasat o anumita tasta sau sa repornim citirea la alta apsare
    - Cerere de la KBD_scan() la IO_scan()
  - Ex. Vrem sa restrictionam executia anumitor rutine in functie de circumstante
    - Apare o avalansa de schimbari de stare la intrarile sistemului si legatura RS232 nu poate sa le trimita pe toate
    - Reducem perioada IO_scan() a.i. transmisia sa fie completa
- Uneori este nevoie sa se execute cateva operatii simple dar prioritare
  - Ex. Micsoreaza luminozitatea LCD-ului dupa un timp de la ultima apasare, clipeste cursorul la pozitia curenta pe ecran cu o anumita frecventa fixa.
  - De cele mai multe ori aceste procese nu au functii dedicate (sunt prea simple) ci sunt executate sincron de o rutina de timer

Polled Loops

- Pro:
  - Foarte simplu de implementat in cod si de depanat
  - Timpul de raspuns este foarte usor de determinat
- Contra:
  - Poate sa cedeze la evenimente in rafala
  - In general, nu are suficiente functionalitati pentru controlul sistemelor complexe
  - Cicli de ceas pierduti, mai ales daca evenimentul din bucla se petrece foarte rar

Sisteme bazate pe intreruperi

- Ce este o intrerupere?
- Un semnal hardware care initiaza un eveniment
- La receptia unei intreruperi, procesorul:
  - Finalizeaza instructiunea curenta
  - Salveaza Program Conter (ca sa se intoarca de unde a pornit)
  - Incarca in PC adresa de inceput a rutinei de tratare a intreuperii
  - Executa RTI
- De obicei, sistemele de timp real pot sa trateze mai multe intreruperi simultan prin implementarea unui mecanism de prioritati
  - Intreruperile pot fi pornite/oprite
  - Intreruperile cu cea mai mare prioritate sunt tratate primele

Implementarea unei rutine event-driven

Fiecare rutina are atasata o coada de evenimente

Ex. Lista circulara cu doua metode: GetEvent si PutEvent

- Trimiterea si primirea de evenimente

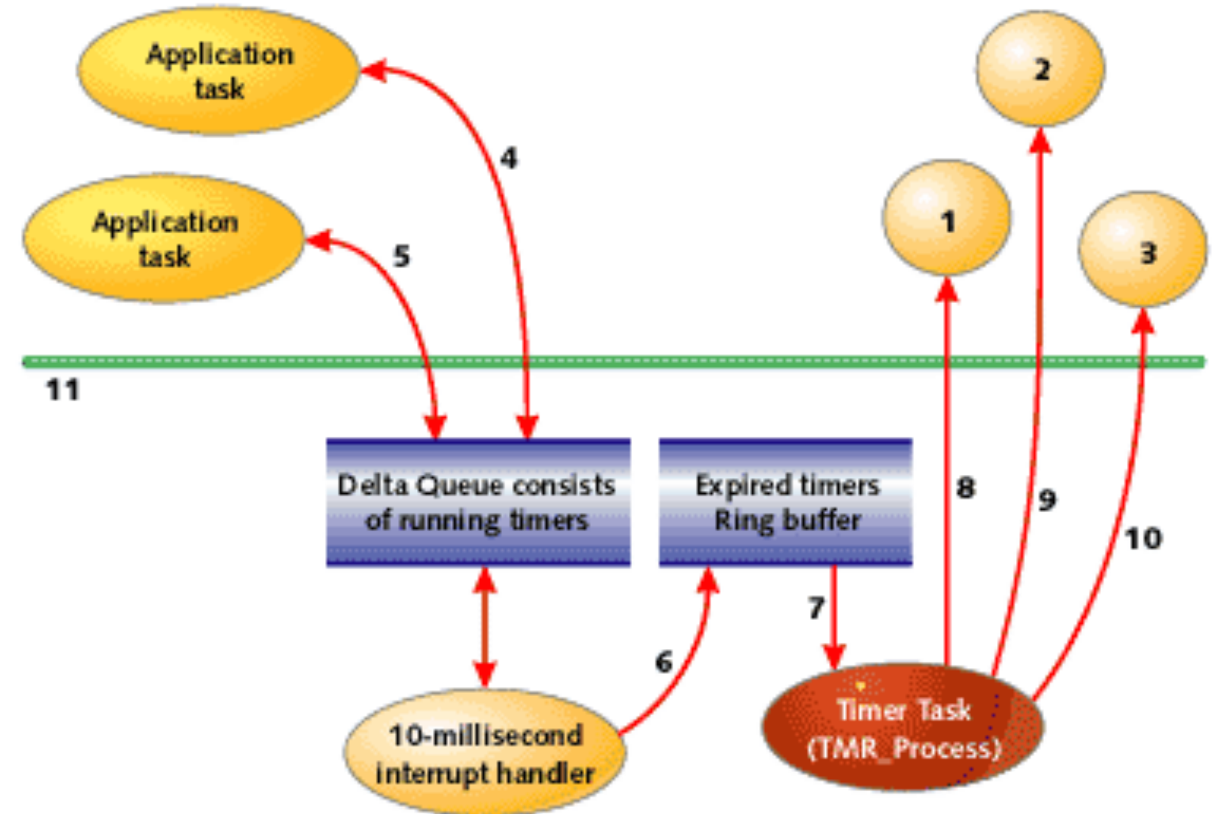
```
void IO_ProcessOutputs(void) {
   int ret;
   EVENT_TYPE OutputEvent;
   OutputEvent.NewState = 1;
   OutputEvent.Number = 1;
   PutEvent(&OutputEvent); // insereaza un eveniment in coada
   // .........
   if ((ret = GetEvent(&OutputEvent) != EMPTY) {
    // am primit un eveniment; proceseaza
    IO_OutputChange(OutputEvent.Number,OutputEvent.NewState)
   }
}
```

Implementarea operatiilor periodice simple

- Mai multe actiuni trebuie executate cu exactitate sau periodic
  - Afisarea cursorului
  - O iesire care trebuie inchisa/deschisa periodic
  - Afiseaza un mesaj la o anumita ora sau periodic
  - Ilumineaza/stinge LCD-ul dupa un timp
- E mai dificil de implementat cate o functie pentru fiecare
  - Se aloca mai multe variabile de incrementare in rutina de intrerupere a timerului
- O solutie: timer software

Timer Software

- Pentru fiecare timer folosit se defineste o functie care se executa la expirarea intervalului
- Rutine de TMR_Start si TMR_Stop
- Toate timerele din sistem sunt tinute in “coada delta” in ordinea expirarii lor.
  - Timerele care expira peste 10, 60, & 200 tick-uri de ceas vor fi stocate:
    - 10 50 (= 60 - 10) 140 (= 200 - 60)



Sisteme Foreground/Background

- Cea mai comun intalnita solutie hibrida pentru aplicatiile embedded simple
- Foloseste un fir de executie interrupt-driven (foreground) SI un fir de executie in bucla principala (background)
- Toate solutiile real-time sunt niste cazuri speciale de sisteme foreground/background
- Polled loops = Sistem Background-only
- Sisteme Interrupt-only = Sisteme Foreground-only
- Tot ce nu este time-critical trebuie sa fie in fundal
- Background este procesul cu prioritatea cea mai mica

Sisteme Foreground/Background

Foreground (interrupt) *(t1)*

```
on interrupt {
  do_clock_module();
  if(time_for_control) do_control();
}
```

Background *(t2)*

```
while (1) {
  if (time_for_display_update) do_display();
  else if (time_for_operator_input) do_operator();
  else if (time_for_request) do_request();
}
```

- Departajarea relaxeaza constrangerile: $t1 + t2 \leq 20$ ms

Abordarea Multi-Tasking

Embedded Systems Laboratory

- O bucla de control: un singur “task”
- Foreground/background: doua task-uri
- Generalizare: task-uri multiple
  - Numite si procese, thread-uri
  - Fiecare proces se executa in paralel
    - Procesele interactioneaza simultan cu elementele externe
      - Monitorizeaza senzorii, controleaza efectoarele, trateaza , IO etc.
    - Creeaza iluzia paralelismului
  - Cerinte
    - Planificarea proceselor
    - Partajarea datelor intre procese concurente

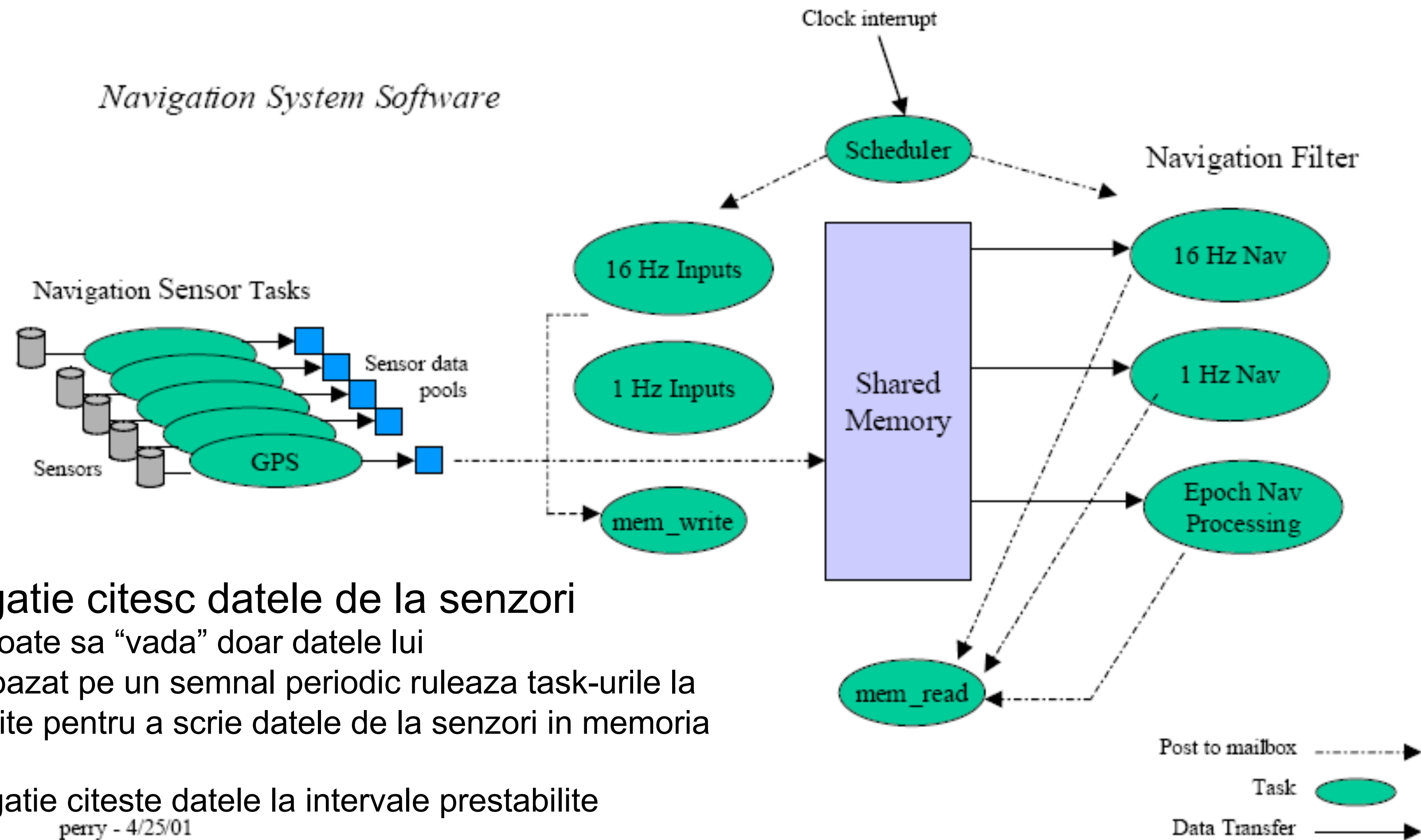
Multitasking

- Ce inseamna Multitasking?
  - Procese separate impart acelasi procesor (sau procesoare)
  - Fiecare task se executa in contextul propriu
  - Detine procesorul pentru cuanta curenta de timp
  - Are variabile proprii
  - Poate fi intrerupt
- Mai multe task-uri pot interactiona si functiona ca un program unitar.

Caracteristicile unui task

- Un proces poate avea
  - Cerinte de resurse
  - Prioritate atasata
  - Relatii de precedenta
  - Cerinte de comunicare
  - Si, cel mai important, constrangeri de timp
    - » Specificarea momentului de timp la care sa se execute sau sa se termine o actiune
    - » Ex. Perioada unui proces periodic
    - » Sau deadline pentru un proces neperiodic

Exemplu Multitasking



Task-urile de navigatie citesc datele de la senzori

- Fiecare task poate sa “vada” doar datele lui
- Planificatorul bazat pe un semnal periodic ruleaza task-urile la frecvente diferite pentru a scrie datele de la senzori in memoria partajata.
- Filtrul de navigatie citeste datele la intervale prestabilite

Multitasking

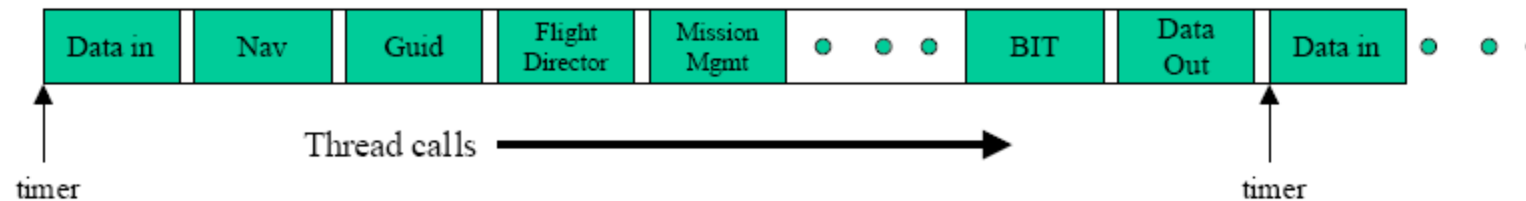
- Context switching
  - Atunci cand CPU-ul schimba un fir de executie cu un altul se face o schimbare de context
  - Se salveaza MINIMUL de informatii menite sa refaca procesul intrerupt la o apelare ulterioara
    - Exemplul cu cartea: De ce e nevoie? *(nume, pagina, paragraf , cuvantul_nr.)*
  - Intr-un sistem de calucul MINIMUL este, de cele mai multe ori
    - Continutul registrelor
    - Continutul PC
    - Continutul registrelor de coprocesor (daca e cazul)
    - Adresa paginii de memorie
    - Adresele I/O-urilor mapate in memorie
    - Variabile speciale
  - In timpul schimbarii contextului, intreruperile sunt dezactivate
  - Sistemele de timp real trebuie sa aiba timpi minimi pentru context-switching.

Multitasking

- Cate task-uri impart acelasi procesor?
  - Sisteme cu executie ciclica
  - Sisteme round-robin
  - Sisteme preemptive

Sisteme cu executie ciclica

- Foloseste o planificare statica pentru a ordona toate firele de executie



- Pro:
  - Usor de implementat (folosite in sisteme critice si de mentinere a vietii)
- Contra:
  - Nu sunt foarte eficiente d.p.d.v. al folosirii CPU
  - Nu permit un timp de raspuns optim (intotdeauna, unele sarcini au prioritate mai mare)

Sisteme Round-Robin

- Procesele se executa secvential pana la finalizarea tuturor
- De multe ori in conjunctie cu o schema de executie ciclica
- Fiecarui task ii este alocata o cuanta de timp.
- Exista un timer de sistem care genereaza o intrerupere la expirarea fiecarei cuante de timp
- Task-ul se executa pana la finalizare sau pana la terminarea cuantei de timp alocate lui.
- Contextul este salvat sau refacut la fiecare iesire/intrare a procesului intr-o cuanta de executie.

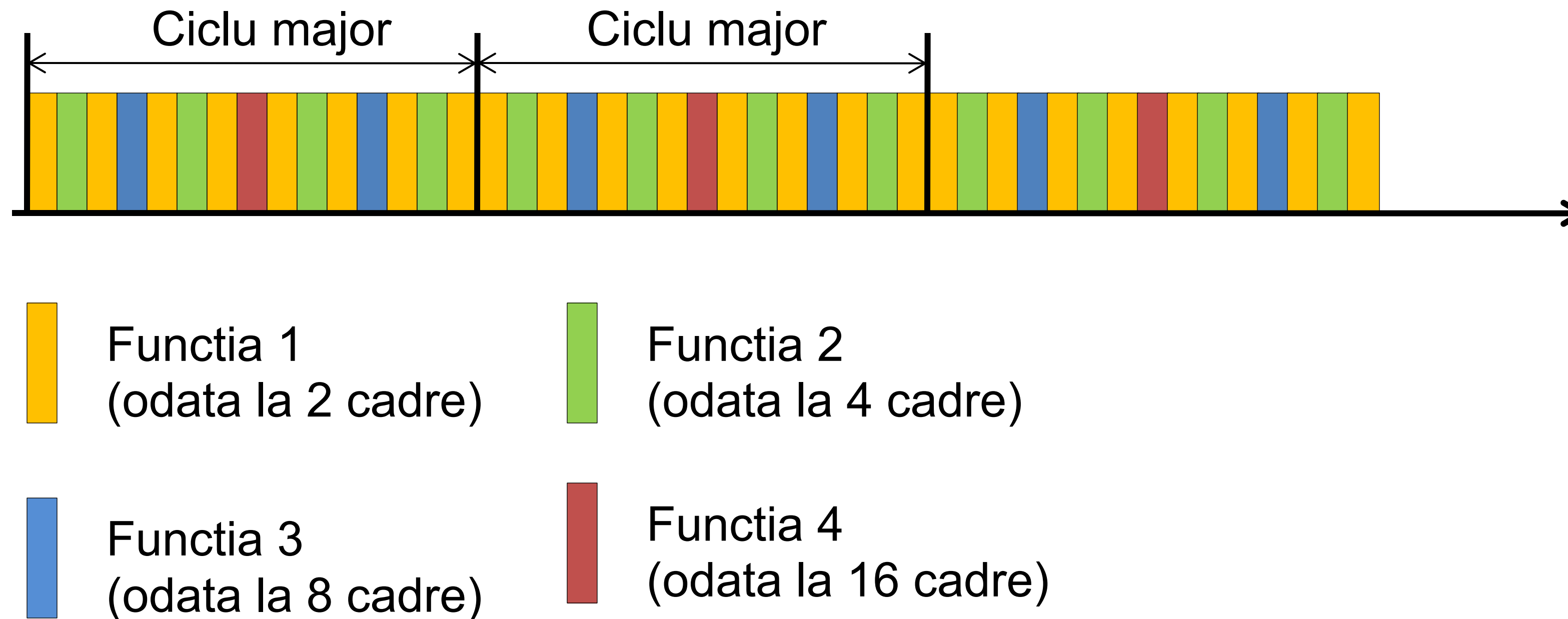
Planificarea Statica

Embedded Systems Laboratory

- Aplicabila la task-urile periodice
- Se construieste o tabela si fiecare proces are alocata o cuanta de timp
- Previzibil dar inflexibil
  - Tabela este total refacuta cand un singur proces isi modifica timpul de executie
- Timpul este impartit in cicli minori (o cuanta) si un timer declanseaza executia procesului planificat pentru cuanta respectiva de timp.
- Un set de cicli minori constituie un ciclu major care se repeta countinuu.
- Operatiile sunt implementate ca niste proceduri si sunt incluse in liste de executie pentru fiecare ciclu minor
- La inceputul unui ciclu minor timerul apeleaza in ordine fiecare procedura din lista respectiva.
- Fara preemptare: operatiile lungi trebuie “sparte” pentru a putea incape intr-un ciclu minor.

Exemplu

- Patru functii care se executa la 50, 25, 12.5 si 6.25Hz (20,40,80 si 160ms) pot fi planificate intr-o executie ciclica cu un ciclu minor de 10ms:



Sisteme preemptive

- Un task cu prioritate mai mare poate sa preempteze pe un al doilea, daca acesta din urma este in executie in cuanta curenta de timp
- Prioritatile alocate fiecarui task sunt bazate pe urgenta executiei fiecarui task in parte
- Prioritatile pot sa fie fixe sau dinamice

Preemptare

- Non-preemptive: procesul, odata pornit nu se opreste decat dupa ce si-a terminat executia
  - » Ex.: N task-uri, fiecare task j apelat la un interval T_j are nevoie de un timp C_j de executie atunci: $T_j \geq C_1 + C_2 + \ldots + C_N$ in cel mai rau caz (toate celelalte N-1 task-uri sunt si ele gata)

- Preemptive: un proces poate fi oprit pentru rularea altui proces
  - Complica implementarea
  - Dar putem face mai bine planificare

Planificarea (scheduling)

Embedded Systems Laboratory

- Date de intrare
  - Unul sau mai multe procese
  - Timpii de activare, executie si deadline pentru fiecare proces

- Algoritm de planificare: politica de alocare a proceselor pe unul sau mai multe procesoare
- Planificare fezabila: daca algoritmul de planificare poate satisface toate constrangerile
- Algoritm optim: Un algoritm de planificare care produce un rezultat fezabil (daca acesta exista)

Evaluarea performantelor algoritmilor de planificare

Embedded Systems Laboratory

- Cazul static: planificare off-line care asigura ca toate deadline-urile sunt satisfacute
  - Metrica secundara:
    - » Maximizarea numarului mediu de sosiri devreme
    - » Minimizarea numarului mediu de intarzieri
- Cazul dinamic: nici o garantie a priori ca termenul limita va fi satisfacut
  - Metrica:
    - » Maximizarea numarului de procese care satisfac termenul limita
- Pentru amandoua cazurile:
  - Overhead de planificare minim

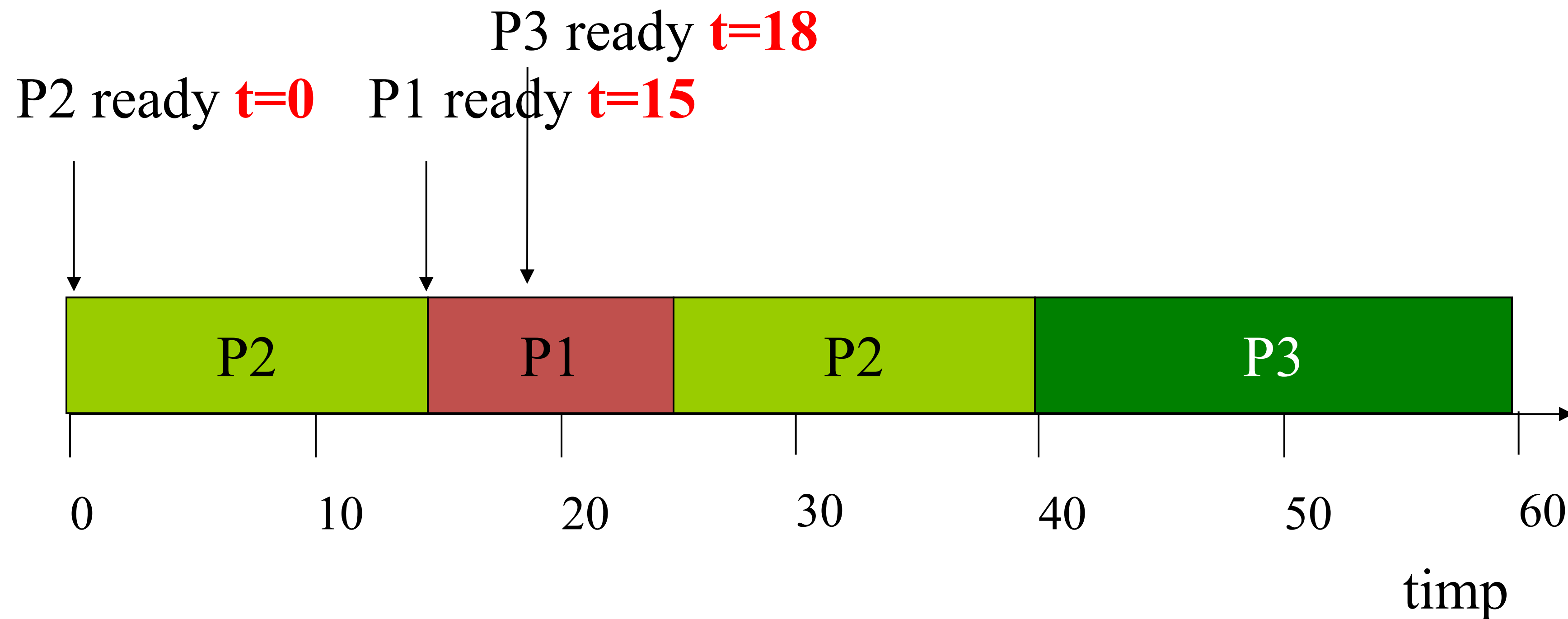
Planificarea bazata pe prioritati

Embedded Systems Laboratory

- Fiecarui proces i se atribuie o prioritate (static sau dinamic)
  - Atribuirea prioritatilor se face tinandu-se cont de constrangerile de timp
  - Prioritatea statica: atragatoare pentru un sistem simplu (ieftin si nu necesita recalculari)
- La un moment de timp, ruleaza sarcina cu cea mai mare prioritate
  - Daca ruleaza un proces cu prioritate mica, si soseste un altul cu prioritate mai mare, primul proces este preemptat
- Atribuirea unor prioritati corespunzatoare permite tratarea anumitor situatii in timp real.

Exemplu

- P1: prioritate 1, timp executie 10
- P2: prioritate 2, timp executie 30
- P3: prioritate 3, timp executie 20



Algoritmi de planificare pe prioritati

Embedded Systems Laboratory

- Rate Montonic Scheduling (RMS)
  - Prioritati statice bazate pe perioade de timp
    - » Prioritate mai mare pentru perioade mai scurte
  - Optim, intre toti algoritmii de prioritizare statica
- Earliest-Deadline First (EDF)
  - Atribuire dinamica
  - Cu cat e mai aproape termenul unui proces, cu atat e mai mare prioritatea
  - Aplicabil atat pentru sarcinile periodice, cat si pentru cele neperiodice
  - Se recalculeaza prioritatile la sosirea unei sarcini noi
    - » Mai costisitor in privinta overhead-ului obtinut la rulare

Planificarea constransa de toleranta la defecte

Embedded Systems Laboratory

- Exemplu: mecanism de impunere a unui termen limita pentru a garanta terminarea executiei unei sarcini principale daca nu exista nici un defect si alocarea unei perioade suplimentare de timp pentru intrarea in executie a unei sarcini alternative (de precizie mai mica) in cazul unei avarii.
    - Daca nu se produce avaria, timpul suplimentar alocat este reutilizat
- Alta abordare: scenariu de rezerva prevazut in algoritmul de planificare normal si activat la producerea unei defectiuni

Planificarea cu realocarea resurselor

Embedded Systems Laboratory

- Sunt variatii in timpii de executie ai unui proces
  - Unele sarcini pot sa termine mai devreme decat era planificat
- Dispecerul de sarcini poate sa realoce acest timp pentru alte sarcini
  - Ex. Sarcinile care nu sunt de timp real pot fi rulate in intervalele realocate (idle slot)
  - Si mai bine: foloseste timpul suplimentar pentru a garanta terminarea sarcinilor cu constrangeri de timp

Procesarea cu o anumita precizie

- Ideea principala: aranjeaza calculele a.i. sa se produca rezultate mai precise daca este mai mult timp la dispozitie
    - Daca nu este suficient timp exista totusi un rezultat de calitate mai proasta in loc de nimic
- Poate fi cuplata cu optimizarea consumului de energie (vezi cursurile anterioare)

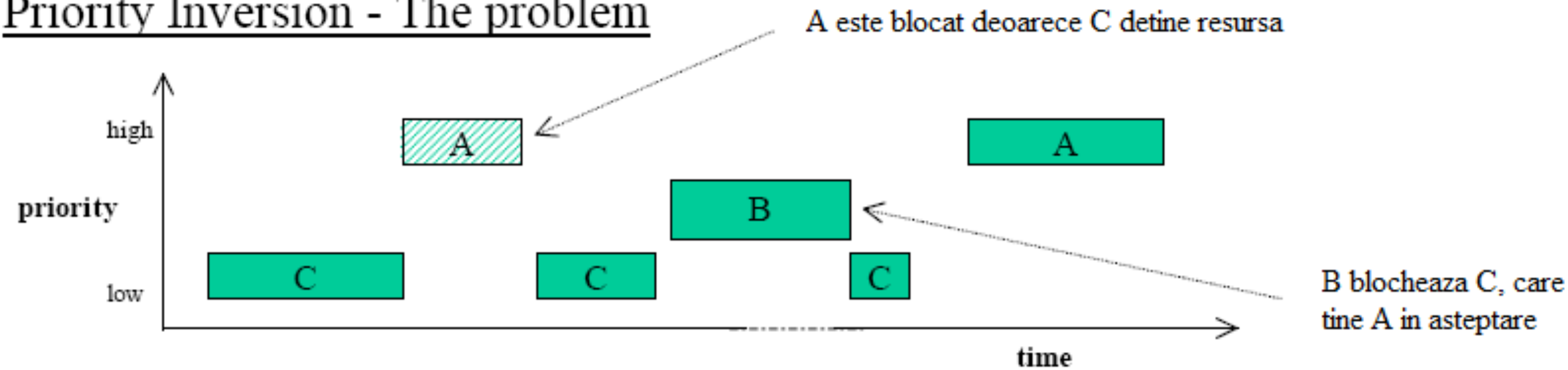
Multitasking

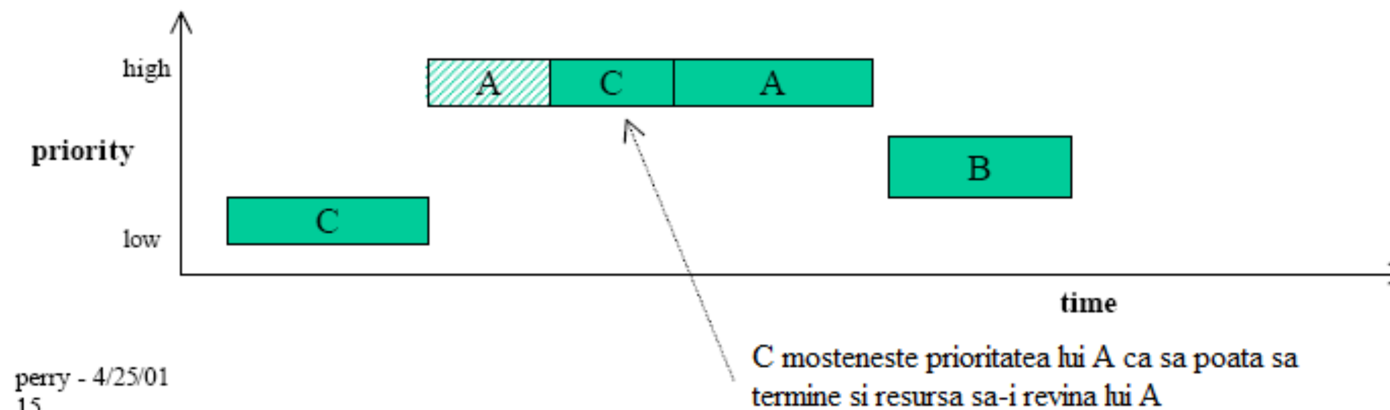
- Multitasking-ul nu este perfect
  - Task-urile de prioritate mare acapareaza resursele si “infometeaza” task-urile cu prioritate mica
  - Task-urile cu prioritate mica impart aceeasi resursa cu cele de prioritate mare si le blocheaza pe acestea din urma
- Cum trateaza un RTOS aceste probleme?
  - Rate Monotonic Systems (frecventa de executie mare = prioritate mare)
  - Mostenirea prioritatii

Multitasking

- Priority Inversion / Priority Inheritance
  - Task A si Task C impart aceeasi resursa
  - Task A este High Priority
  - Task C este Low Priority
  - Task A este blocat cand Task C se executa (adica A ajunge sa aiba prioritatea lui C - Priority Inversion)
  - Task A va fi blocat pentru si mai mult timp daca Task B de prioritate medie vine si blocheaza Task C inainte ca acesta sa termine
  - Un RTOS bun detecteaza aceasta conditie si promoveaza temporar Task C la High Priority a Task A (Priority Inheritance)

Priority Inversion/Inheritance



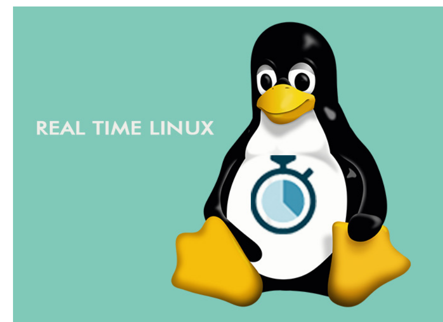


RTOS Full-Feature

- Extind solutiile foreground/background
  - Adauga:
  - Interfete de retea
  - Drivere dispozitive
  - Solutii complexe pentru debugging
- Sunt alegerea cel mai des intalnita pentru sistemele complexe
- Foarte multe sisteme de operare disponibile deja pe piata.

Cateva exemple de RTOS

- Pentru sisteme mici
  - NuttX
  - TinyOS (Berkeley)
  - ContikiOS
- Sisteme medii
  - FreeRTOS
- Sisteme complexe
  - VxWorks
  - Real-time Linux



TinyOS

Embedded Systems Laboratory

- Kernel monolitic
- Scris in o varianta de C, nesC
- Algoritmi, protocoale, drivere, sisteme de fisiere si shell
- Dezvoltat pentru retele de senzori
- Nu mai este dezvoltat activ

tinyos.net

ContikiOS

- Arhitectura pe mai multe niveluri
- Scris in C, dar anumite parti pot fi scrise si in C++
- Servicii pentru device drivers, comunicatie si manipularea datelor
- Stiva uIP, device driver loader si system Protothreading
- Protothreads sunt o implementare simpla de multithreading, fara a folosi stiva
- Code replacement la runtime

contiki-os.org/

Exemplu: proces periodic in ContikiOS

Embedded Systems Laboratory

```
PROCESS_THREAD(etimer_process, ev, data) {
        static struct etimer et;
        PROCESS_BEGIN(); /* set timer to expire after 5 seconds */
        etimer_set(&et, CLOCK_SECOND * 5);
        while(1) {
                PROCESS_WAIT_EVENT(); /* Same thing as PROCESS_YIELD */
                if(etimer_expired(&et)) { /* Do the work here and restart timer to get it periodic !!! */
                        printf("etimer expired.\n");
                        etimer_restart(&et);
                }
        }
        PROCESS_END();
}
```

NuttX

- Creat de Gregory E. Nutt
- Primul release public: feb 2007
- Ruleaza pe uC si uP de la 8 la 32 de biti
- BSD license
- Amprenta memorie foarte mica (zeci de kB)
- Foarte personalizabil
- Inspirat de Linux/Unix
  - VFS
  - MTD
  - PROCFS
  - NuttShell (nsh)

nuttx.apache.org

NuttX

- POSIX complaint
- Fully preemptible
- Virtual File System (VFS)
- Loadable kernel modules
- Symmetric Multi-Processing (SMP)
- Realtime scheduling (FIFO, RR, SPORADIC)
- Tickless operation support (lower power consumption)
- Pseudo-terminals (PTY) and I/O redirection

Real-time Linux & VxWorks

- Microcontroller (MMU-less) :
  - uClinux - (kernel < 512KB) cu implementare full TCP/IP

- VxWorks
  - Multitasking
  - IPC – mutex, cozi de mesaje
  - Sistem de fisiere
  - IPV6
  - VxSim – simulator
  - Sisteme care folosesc VxWorks:
    - Boeing 787
    - ASIMO
    - Linksys WRT54G
    - F18 - Super Hornet
    - Mars Reconaissance Orbiter
    - Spirit & Opportunity Mars Exploration Rovers
    - Deep impact, Stardust

Alegerea unui RTOS

- De unde stiu care este cea mai buna solutie pentru aplicatia mea?
- Un indiciu important il primesc de la modul cum ajung datele in sistemul meu
  - Neregulat (o secventa stiuta dar variabila de intervale in care primesc date sau evenimente)
  - Rafala (secventa arbitrara limitata la un numar maxim de evenimente simultane)
  - Limitat (interval minim intre doua intrari succesive de date)
  - Limitat cu rata medie (intervale imprevizibile dar apropiate de o medie globala)
  - Nelimitat (pot sa fac doar o predictie statistica)
- Care este I/O-ul critic?
- Exista termene limita absolute?

Alegerea unui RTOS

Embedded Systems Laboratory

- Exemple din viata reala:
  - Vehicul reutilizabil pentru lansarea satelitilor pe orbita
    - Vectorul de control pentru propulsie necesita date care estimeaza altitudinea odata la 40msec sau racheta devine instabila
  - *Executie ciclica.*
  - Navigatia si controlul unui submarin.
    - Interfata cu senzori multipli care sunt esantionati cu rate diferite
    - Informatia de la unitatea inertiala de referinta este cruciala dar temporizarea exacta a datelor de intrare nu este esentiala
  - *Schema de executie preemptiva de pe un RTOS comercial. Task-urile importante au prioritatea cea mai mare.*

Alegerea unui RTOS

- Sistem de control avionica – are nevoie de date de la suprafetele de control al zborului si de la echipamentul de navigatie la fiecare 50ms
  - *Executie ciclica. Fiecare task ruleaza pana la finalizare. Toate task-urile ruleaza in serie.*
  - *Ultimele task-uri s-ar putea sa nu termine executia pana la aparitia intreruperii de 50ms.*
- Microcontroller care opereaza antene radar si comuta intre ele pentru a determina pozitia unui obiect. Daca primeste semnal, alimenteaza circuitul de procesare a semnalului.
  - *Polled loop.*

Concluzii

- Multi-tasking implica partajarea resurselor de catre mai multe procese
- Fiecare task are contextul propriu de executie
- Mai multe task-uri se pot combina pentru a forma un program
- Sunt mai multe cai de a implementa multi-taskingul
  - Cyclic Executive
  - Round Robin
  - Sisteme bazate pe prioritati
- Unele sisteme sunt construite prin combinarea mai multor concepte RTOS
- Nu exista o singura cale care sa fie garantat buna pentru implementarea unui sistem embedded dar exista cu siguranta cai gresite. Alegeti-va RTOS-ul potrivit aplicatiei!