Robotics Real-Time Programming

Ludovic Saint-Bauzel

Robotics Real-Time Programming

Ludovic Saint-Bauzel

2013-14

Contents

1 Introduction

1.1 What is a robot ?

1.1.1 Definition

Robot

- Definition : An Autonomous Robot is a complex system that has the ability to decide its actions on its environment from its sensing, state in order to fill his aims. Survive, Assist ...
  - Sensors
  - Decision
  - Action
- Physical ability : **read** information from sensor, **send** orders to actuators, have a regulation (**Time** based computation) and a decision computation unit (**policy**).

Robot : Computer science point of view
A computer science way to see a robot is to say it is a computer with peripherals that bring sensing, actuation ability and because of real life interaction need to put the stress on the time. So like a computer we will have to deal with :

- bus
- memory
- cpu

1.1.2 Hardware

Structure Choice criteria

- $\boxed{\text{Critical path?}}$
  - the main loop
  - delayed sensors
- Method
  - Tasks
  - Pipes (read only, read write, write)

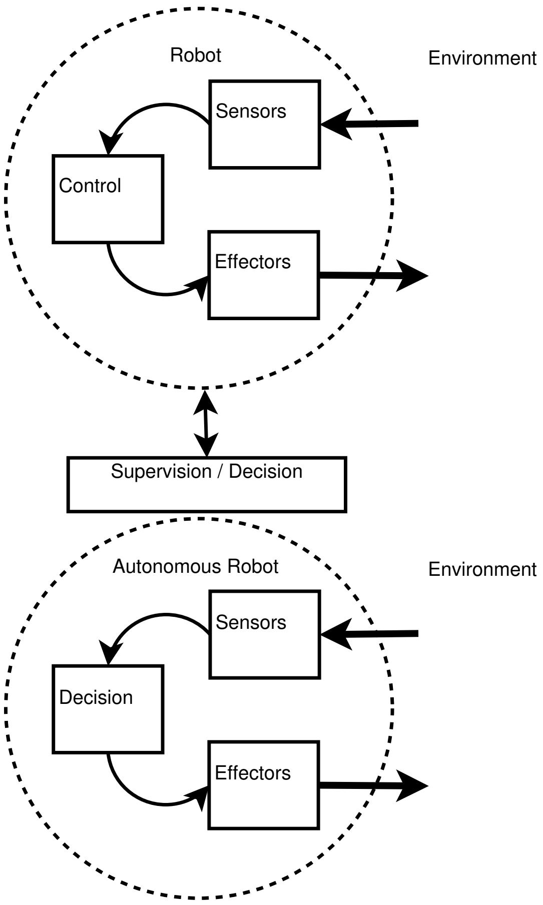


Figure 1: Classic (up) and Autonomous (down) Robot Models

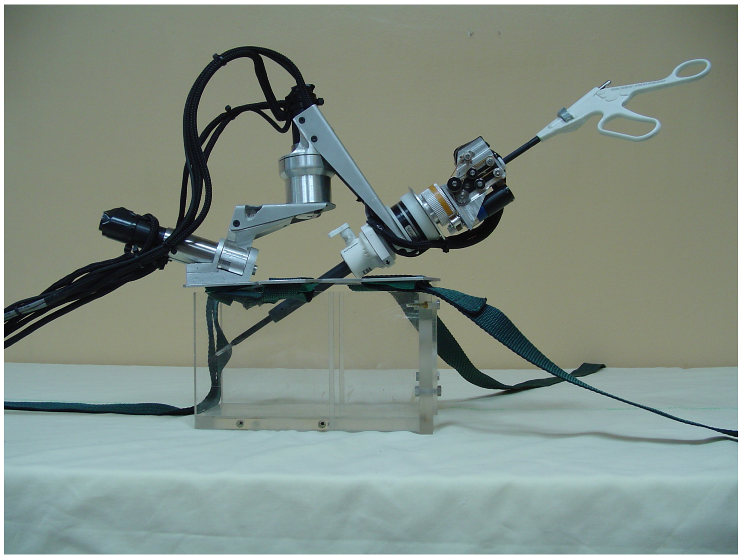


Figure 2: mc2e

- Parameters
  - Computation
  - Time constraints
  - Memory needs
  - Precision of measure

Monocomputer

- Properties
  - Limited Computation Ability
  - Delays are limited too
  - Simple but efficient
- Recent Improvements
  - GPU
  - Multicore CPU
- Example

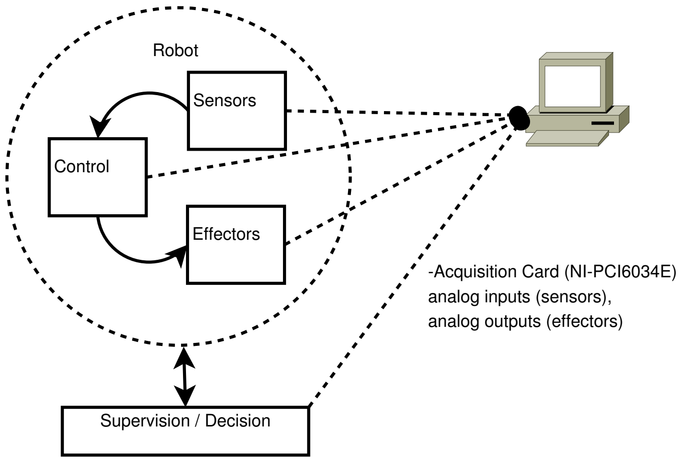


Figure 3: mc2e Architecture

Monocomputer

- Properties
  - Limited Computation Ability
  - Delays are limited too
  - Simple but efficient
- Recent Improvements
  - GPU
  - Multicore CPU
- Example

Multi computer

- Properties
  - High scalability of computation
  - High time lost : Bus communication
- Recent Improvement
  - Distributed algorithms developpment
- Example

Figure 4: Staubli

Multi computer

- Properties
  - High scalability of computation
  - High time lost : Bus communication
- Recent Improvement
  - Distributed algorithms developpment
- Example

Master Computer - Slave Computing Units

- Master is connected to slave unit
- Often a bus
- Image

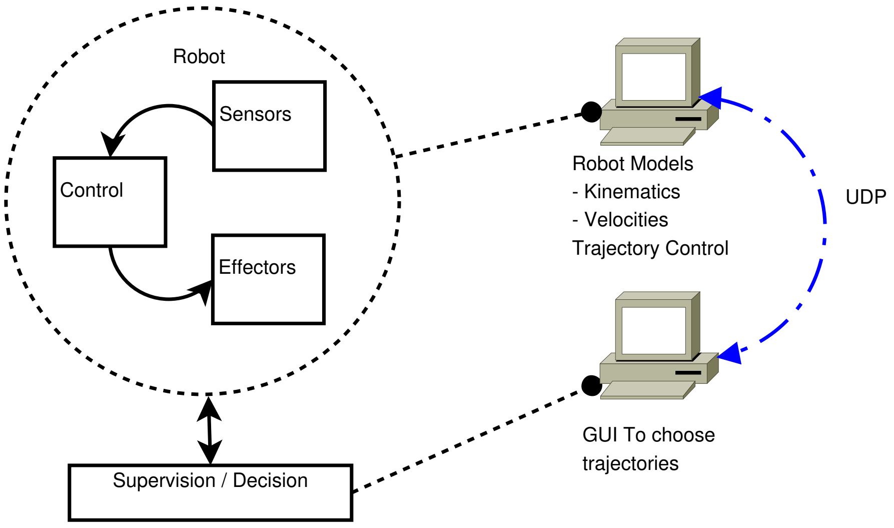


Figure 5: Staubli Architecture

Bus

- Definition
  - Physical connection
  - Multiple devices
- Common elements :
  - Base Address called ports
  - Local Address
- Examples :
  - Serial
    - RS485
    - USB
    - CAN
  - Parallel
    - ISA
    - PCI
    - ...

1.1.3 Software

How the robot can make environment interaction ?

- Categories
  - Perception
  - Decision
  - Actions
- Computer Science Name : **tasks**

Real-Time Programming

- Low Level Access
- High Time constraints
  - Don't mean quick!
  - Mean ability to respect those constraints

Low Level Programming
LINUX

```
#include <asm/io.h>
...
 unsigned char inb (unsigned short port) ;
 void outb (unsigned char byte, unsigned short port) ;
 unsigned char insb (unsigned short port, void *addr,
                                  unsigned long count) ;
 void outsb (unsigned short port, void *addr,
                                  unsigned long count) ;
```

DOS

```
#include <dos.h>
...
 unsigned char inputb (unsigned short port) ;
 void outputb (unsigned short port,unsigned char byte) ;
```

1.2 What is a controller ?

1.2.1 System Design

SADT : Description

SADT : Structure

Petri Net

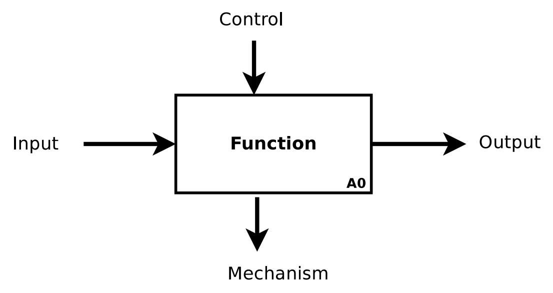


Figure 6: SADT: Structured Analysis and Design Technique

1.2.2 Methodology

Find the tasks!

- No Fuzzy properties
  - Real-Time
  - Good Refresh rate
- Nothing is perfect
  - Properties
    - Time (Loop (Freq?), Oneshot)
    - Load (CPU)
  - Boundaries
    - Delays
    - Memory
- Task repeated every miliseconds with acceptable delay of 0.01 ms

Classical Robotics Tasks

- Control Task
  - Loop, Delay
  - Variable Parameters
  - Static parameters
- User Interface
  - Supervision
  - Send Orders (Asynchronous, Parameters)

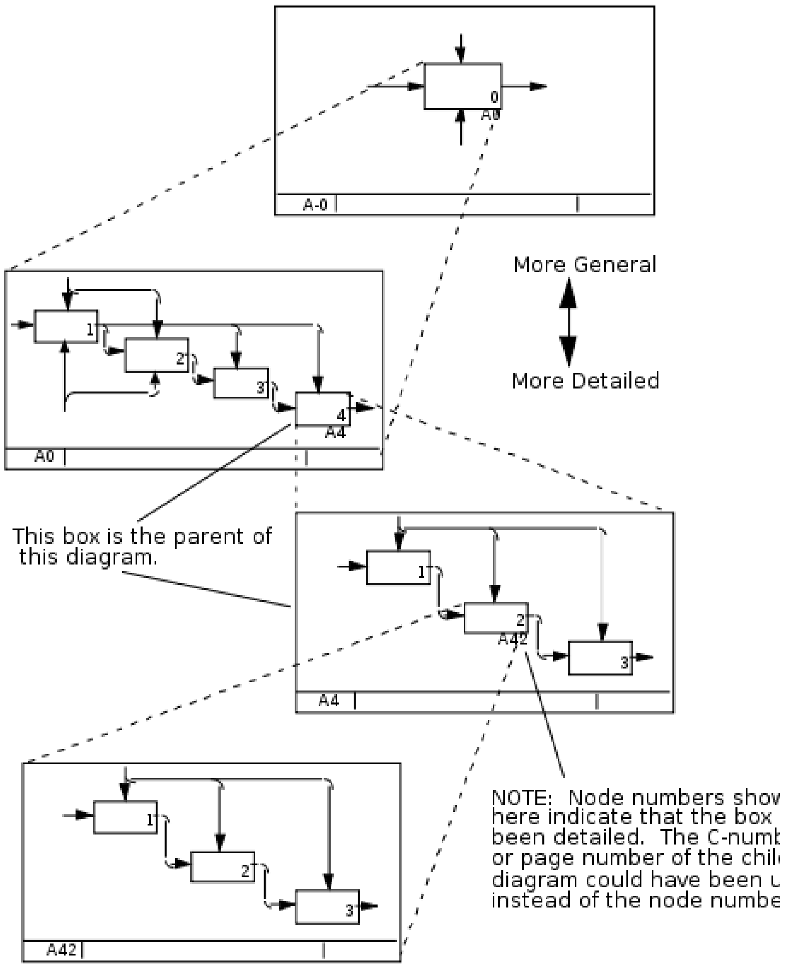

0
A0
A-0
More General
More Detailed
1
2
3
4
A4
A0
This box is the parent of this diagram.
1
2
A42
3
A4
1
2
3
A42
NOTE: Node numbers show here indicate that the box been detailed. The C-numb or page number of the chil diagram could have been u instead of the node numbe

Figure 7: SADT : Hierarchical Structure

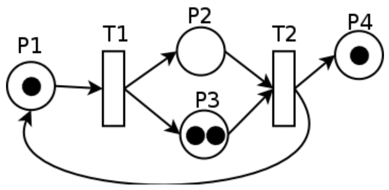


Figure 8: Petri Net

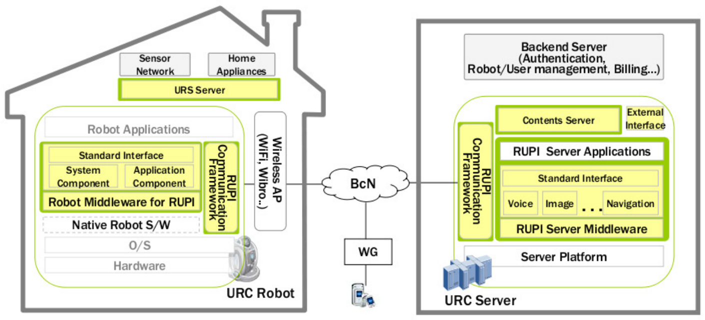


Figure 9: RUPI

MVC+

- View
  - Supervision
  - GUI
- Controller
  - Send Orders
  - Control Task Information
  - GUI
- Model
  - Synchronization
  - Communication System
  - Database
- **RT Task**

2 Robotics Software Frameworks

2.1 RUPI

2.1.1 RUPI Point Of View

Challenging Scenario

2.2 OROCOS

2.2.1 Open RObotics COntrol Software

Introduction

The aim of this part is to present a framework that is a state of art of what can be done by a system that manages the control of a robot. Of course this framework is not perfect and some choices can be questionned but i decided to choose this to give you the main rationales that one can imagine to use to make a good controller. As a summary of the properties I appreciate:

- *component programming* that provides a good reusability of the code and lets deploy differents things according to the needs
- *Dataflow* : The main system is based on a software dataflow approach. That is very close to the control of systems.
- *State-Machine* of components lets work with hooks for every state. Gives a supervision of the components.
- *State-Machine* of deployment, lets implement complex behaviour, change the control kind
- *Activity mechanism* to update each component : gives a wide variety of updating mechanism different to the data-flow
- *logging* Thanks to the component approach and the hook approach, it is possible to subscribe to ports of component and see what happens

OROCOS Project

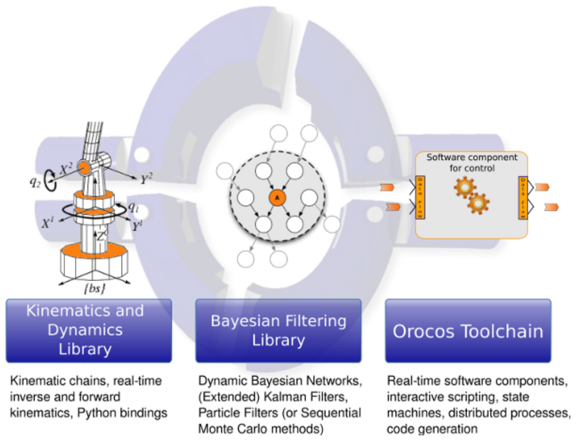

Software component for control
Kinematics and Dynamics Library
Bayesian Filtering Library
Orocos Toolchain
Kinematic chains, real-time inverse and forward kinematics, Python bindings
Dynamic Bayesian Networks, (Extended) Kalman Filters, Particle Filters (or Sequential Monte Carlo methods)
Real-time software components, interactive scripting, state machines, distributed processes, code generation
{bs}

Figure 10: Orocos Project

History

- December 2000, as an idea of Herman Bruyninckx
- European Funding
- European Labs
  - K.U.Leuven in Belgium (Orocos@KUL)
  - LAAS Toulouse in France (Orocos@LAAS)
  - KTH Stockholm in Sweden (Orocos@KTH)

Orocos Toolchain = RTT + OCL

- Real-Time Toolkit
- Orocos Component Library

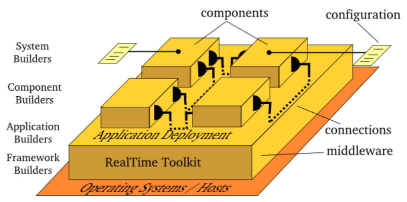


Figure 11: Orocos Component/Plugins Architecture

Task Context

OCL -> Logging Tools

- 2 Solutions
  - Custom Component
    - + Complete control of the generated file
    - - Some additional code that is not the heart of the controller
  - Listen a component through logging tools
    - In C++ code :
    - In deployment :
    -

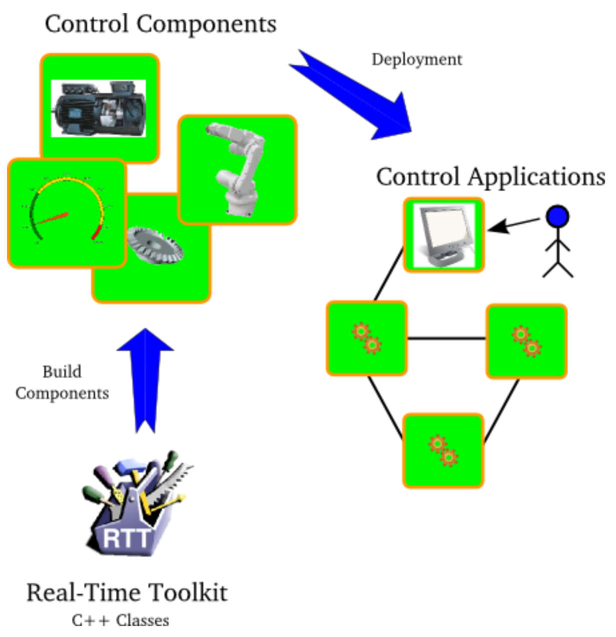


Figure 12: RTT Workflow

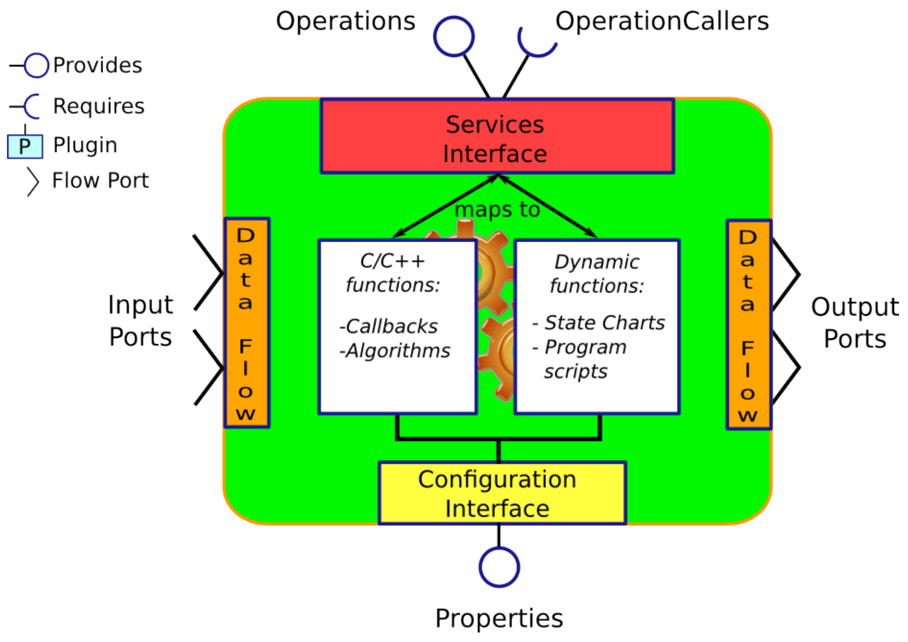


Figure 13: Component Description

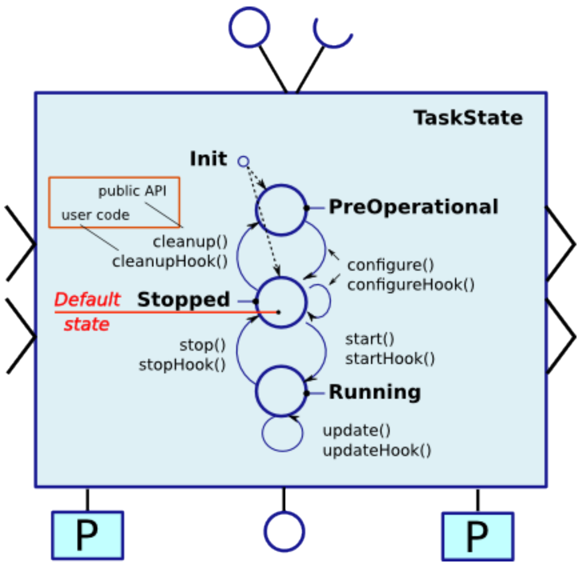
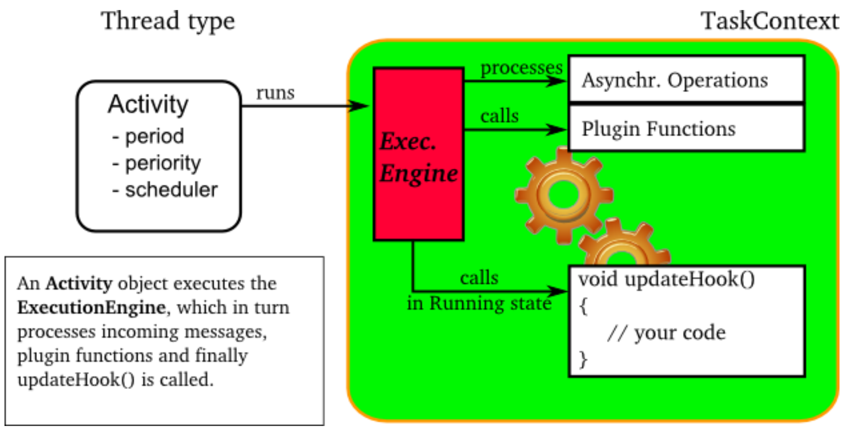


Figure 14: Component State Diagram



An **Activity** object executes the **ExecutionEngine**, which in turn processes incoming messages, plugin functions and finally updateHook() is called.

Figure 15: Component Activity Diagram

Task State Diagram

Activity vs Threads

Data Flow

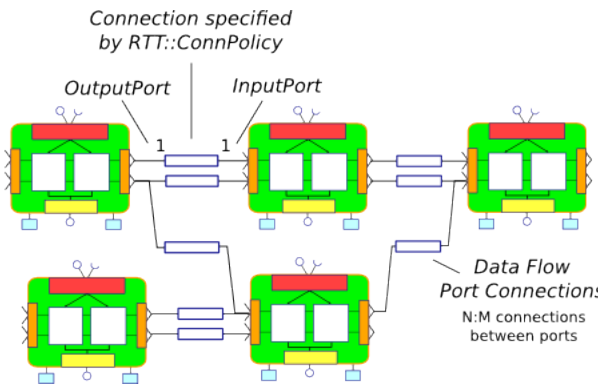


Figure 16: Components Connections

Example : Definition

- Supervisor
  - Initialisation
  - Waiting
  - Standing-Up
  - Walking
  - Alert
- Sensors
  - Force Handles
  - Orientation of the arm
  - Orientations of the wheels(Beta_L,Theta_L, Beta_R,Theta_R)
- Controls
  - Impedance Control
  - Position Control
  - Speed Control

- Models
  - XZ MGD
  - XYPhi MGD
  - Trajectory Generator

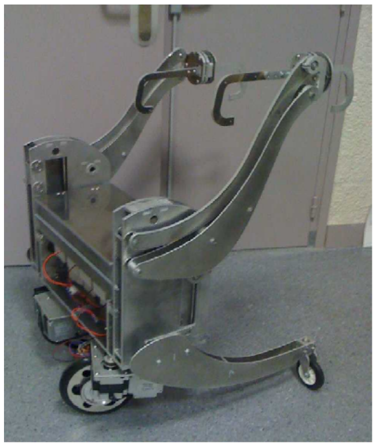


Figure 17: Walky ISIR

Example : Modelling State Machine

Example : Modelling Data Flows

Example : State Machine (Excerpt)

```
StateMachine OurStateMachineType
{
  var bool detected;
  initial state INIT {
                entry {
                        ConsoleOut.display("[INIT]");
                        /*...*/ Plant.configure();
                }
            transition select Waiting;
        }
  state Waiting {
        entry {
                  STSDetector.start();
        }
        transition STSDetected(detected)
                   if (detected) then select SitToStand;
} /*...*/
RootMachine OurStateMachineType statemachine
```

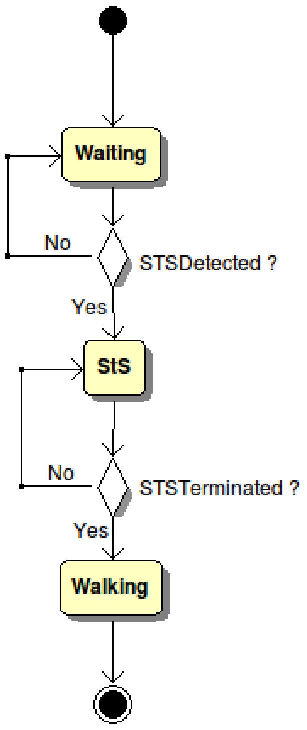


Figure 18: State Machine of Walky Experiment

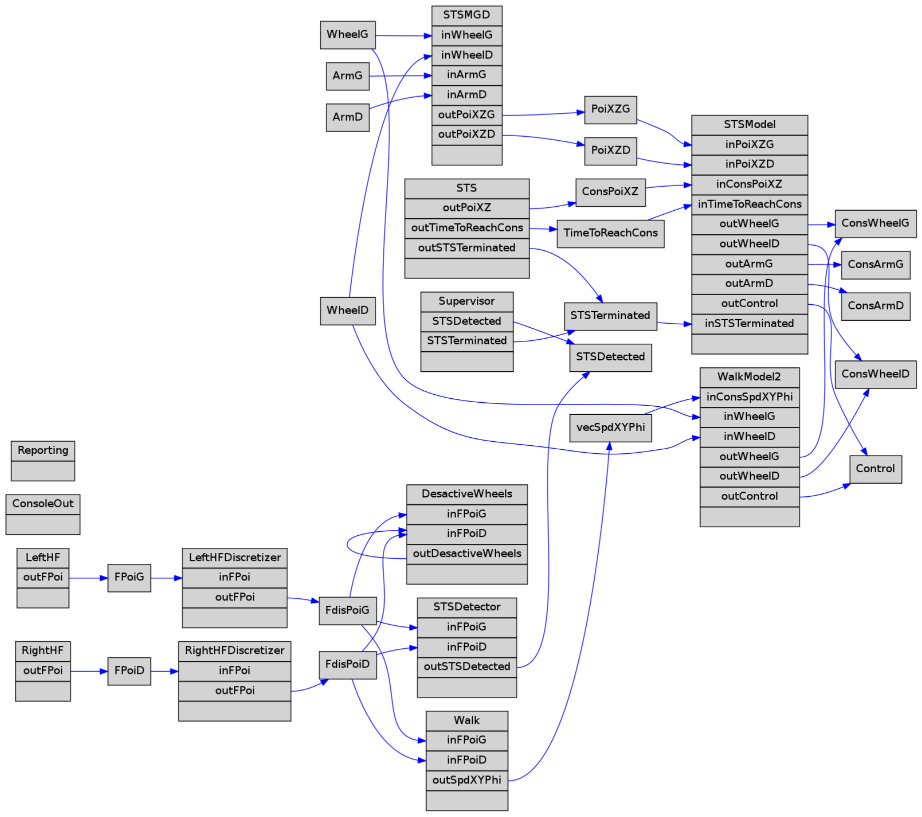


Figure 19: Dataflow of Walky Controller

Example : Component Skeleton(Excerpt)

```
class Supervisor : public TaskContext {
protected:
  InputPort<std::vector<char> > inButtons;
  InputPort<bool> STSDetected; /*...*/
  string stateMachineName;
public:
Supervisor(string const& name)
        : TaskContext(name),
          stateMachineName("statemachine")
{
  this->addEventPort("STSDetected",STSDetected);
  this->addProperty("stateMachineName",stateMachineName);
}

bool configureHook() {
   return true;
}

bool startHook() {
scripting::ScriptingService::shared_ptr sa
= boost::dynamic_pointer_cast<scripting::ScriptingService>(
provides()->getService("scripting"));
scripting::StateMachinePtr sm = sa->getStateMachine(stateMachineName);
if(!sm) {
log(Error) << "State Machine not loaded in Supervisor."<< endlog();
return false;
}
return sm->activate() && sm->start();
}

void updateHook() {
}

void errorHook() {
}

void stopHook() {
scripting::ScriptingService::shared_ptr sa
= boost::dynamic_pointer_cast<scripting::ScriptingService>(
provides()->getService("scripting"));
scripting::StateMachinePtr sm = sa->getStateMachine(stateMachineName);
if(!sm) {
log(Error) << "State Machine not loaded in Supervisor."<< endlog();
return;
}
sm->stop();
sm->deactivate();
}

void cleanupHook() {
}

};
```

Example : Deployment File(Excerpt)

```
<properties>
 <struct name="Supervisor" type="Supervisor">
  <struct name="Activity" type="Activity">
   <simple name="Period" type="double"><value>0.1</value></simple>
   <simple name="Priority" type="short"><value>50</value></simple>
   <simple name="Scheduler" type="string"><value>ORO_SCHED_RT</value></simple>
  </struct>
  <struct name="Peers" type="PropertyBag">
   <simple type="string"><value>STSDetector</value></simple>
<!--... [EVERY PEERS that will be touched by this component] ... -->
  </struct>
  <struct name="Ports" type="PropertyBag">
   <simple name="STSDetected" type="string"><value>STSDetected</value></simple>
  </struct>
  <simple name="RunScript" type="string"><value>deployment/smSTSandWALK.osd</value></simple>
 </struct> <!--.......--> </properties>
```

2.3 ROS

2.3.1 Robotics Operating System

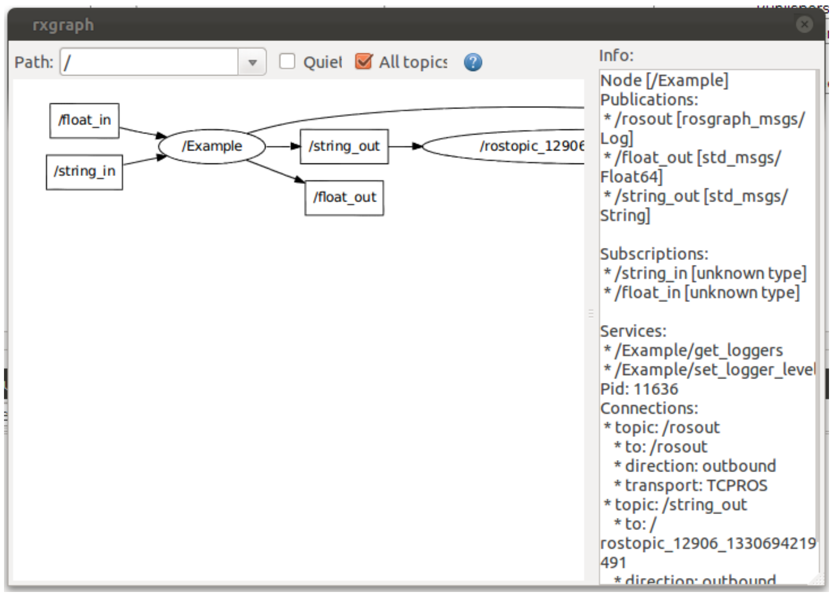
Definition

- ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.
- Willow Garage
  - Created in 2006
  - 2008 first stable release of ROS
  - 2010 ROS 1.0

Connecting Middleware

- Nodes
  - Server (roscore)
  - Output (rosout)
- Topics
  - Publishers
  - Subscribers
- Parameterized Connections
  - TCP

  - SharedMemory
  - Serial (Arduino RosSerial)



```
Info:
Node [/Example]
Publications:
 * /rosout [rosgraph_msgs/
Log]
 * /float_out [std_msgs/
Float64]
 * /string_out [std_msgs/
String]

Subscriptions:
 * /string_in [unknown type]
 * /float_in [unknown type]

Services:
 * /Example/get_loggers
 * /Example/set_logger_level
Pid: 11636
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /string_out
    * to: /
rostopic_12906_1330694219
491
    * direction: outbound
```

Figure 20: rxgraph of /Example node with its topics

Meta-Operating System

- Meta-Operating-System
  - rosdep
  - rosmake
  - roscd
  - rosed
  - rosrun
- Usefull tools
  - rxgraph
  - rxplot

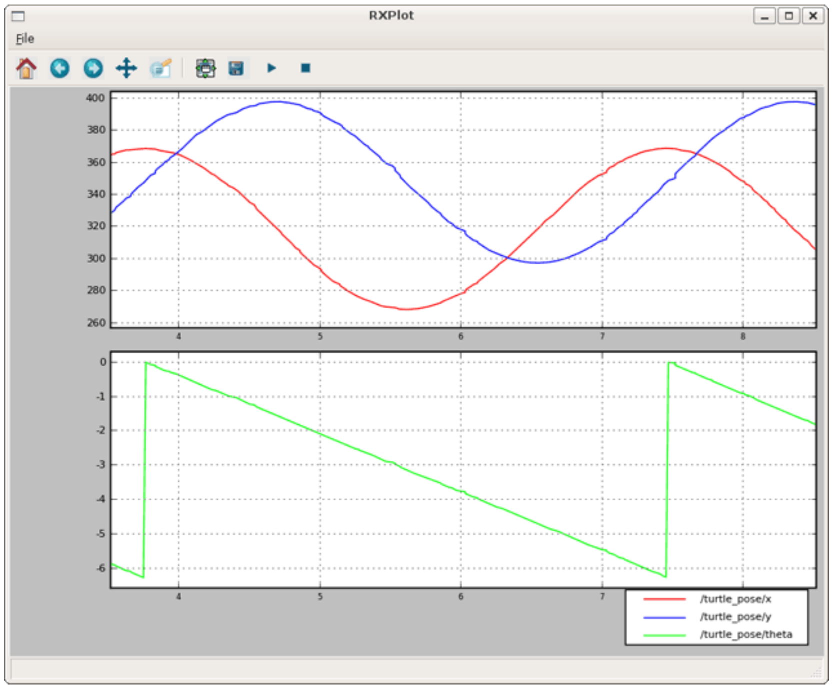


Figure 21: rxplot of /Example node with its topics

excerpt Manifest

```
<package>
  <description brief="depth_reg">

     depth_reg

  </description>
  <author>bonjean</author>
  <license>BSD</license>
  <review status="unreviewed" notes=""/>
  <url>http://ros.org/wiki/depth_reg</url>
  <depend package="cv_bridge"/>
  <depend package="roscpp"/>
  <depend package="sensor_msgs"/>

</package>
```

2.4 Frameworks

2.4.1 Conclusion

Connectivity Question

- Exercice :
- Description
  - Kinect
  - OpenCV
  - SLAM
  - Directions
- Aim
  - When someone do hello then go to its direction

Example

- Exercice
  - Description
    - Kinect
    - OpenCV
    - SLAM
    - Directions
  - Aim
    - When someone do hello then go to its direction
- Result
  - Kinect -> OpenCV -> Position -> SLAM -> Directions

Connectivity Solutions

- In OROCOS :
  - Omni ORB
  - From Scratch Components Connection
    - RPC
    - CORBA
    - Sockets -> Server/Client Approach
- **ROS**

Conclusion

- Overview Implementation
  - Controller on one Computer : OROCOS
  - Communication between computing units : ROS
- Method
  - Identify the states of your system (Tasks, loops)
  - Think of the time, delays and how to overlay some of them
  - Write the data flow for each state
  - Look at the Computing Units At least one node per Unit (ROS)
  - Therefor the components are described (inside one node)

3 OS : Tasks And Threads

3.1 Operating System Approach

3.1.1 Introduction

Operating System few words
A software :
or precisely the first software executed by your computer Consequences : Manager

- Schedule task order
- Manage resources in order to fulfil every needs
- Use of all the resources.

Resources

Computer Science :

- Processor : CPU, IRQ
- Communication : BUS
- Memory
- ...

Robotics :

- Motor
- Sensors

What is RealTime

Two examples :[3mm]
A navigation system computing the best path for a boat.
A navigation system computing a boat position during its sailing.

- First case: Computing time is not a constraint, the result is this only thing that is interesting
- Second case, if computing time is too long, position is false.

An informal definition

Real-Time system : A system where the behaviour relies on precision of computation and also the time when it is produced.

In other words, a delay is considered as an error that can lead to severe consequences

3.1.2 Properties

Time scheduling

- time-driven system : measure of time leads to actions
- event-driven system : arising of an event leads to actions
- reactive system : constrained time to process

Predictability determinism reliability
Real time system : in worst case.[3mm]

Predictability : Ability to identify in advance if a system is able to respect its time constraints Knowledge of parameters related to computation.

Determinism : remove any uncertainty in individual task behaviour and when they are together

- Variation of task execution time
- IO duration, reaction time
- Interruption reaction

Reliability : Behaviour and Fault tolerance

Classical system limits
Classical systems are based on multiple task not well adapted to real time constraints :

- scheduling policy that aims to share time in a balance way for each task
- access mechanisms that shares resources and synchronizes time uncertainty
- interruption management is not optimal
- Virtual memory and cache memory management generate some delays
- timer management is not very accurate

3.1.3 Multitasks

Services of a multitasks system
Conclusion : A single program rarely use all CPU load[2mm]

Used those idle times to compute some concurrent other tasks from other software : a part of what an operating system must do

- running concurrent tasks
- exchanging with outside world
- managing memory resources
- sharing hardware, synchronization, communication
- time management

What are the limits for real-time system?

CPU Management - Task, process

Standard operating systems manage process that can be described as follow :

- separated memory room
- special channels for communication
- high time cost spent for creation of a new one
- protection mechanisms are also time costing
- switch between two processes need a lot of memory switches

CPU Management - Task, process

One process works on many memory areas

- computing area (instructions of the software)
- a heap data area
- a stack area for temporary data (variable)

Context of a process is composed of registers, a name, a state, etc. [3mm]

A **thread** only own individual context data , everything else is shared with father process

Management multiple task and scheduling

A activated process can have many states, here are the main one :

- **created**
- **terminated**
- **running** : task is currently using the CPU. We say it is **elected**.
- **waiting** : task is asking for CPU. It is **electable**.
- **blocked** : task is blocked waiting for an event to arise.

Management multiple task and scheduling

State transitions are :

- **waking** : blocked $\rightarrow$ waiting
- **allocating** : waiting $\rightarrow$ running
- **unallocating** or **pre-empted** : running $\rightarrow$ waiting
- **blocking** : running $\rightarrow$ blocked

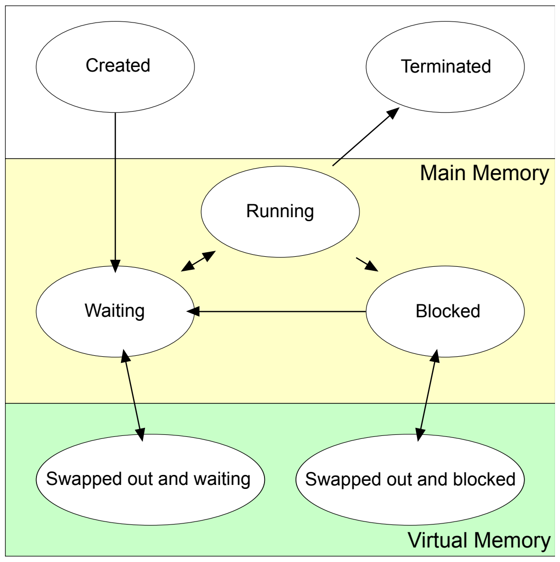


Figure 22: Processus States

Scheduling

Scheduler must manage how the task are allocated to the CPU.

Schedule in the best way is **very** hard.

A scheduler called **with request** or **pre-emptive** if running process can be unallocated by scheduler.

Non pre-emptive systems can do context switch only when a task is terminated or when it is asking for context switch

Scheduler works most of the time on-line. That means that a policy must exists and be applied[3mm]

Scheduling policy

Classical scheduling rules :

- First In First Out (FIFO)
- Each one its turn (tourniquet, *Round Robin* : RR)
- Priority based

Example

Task	Length	Priority	Start time
T1	6	1	0
T2	3	2	0
T3	4	1	0

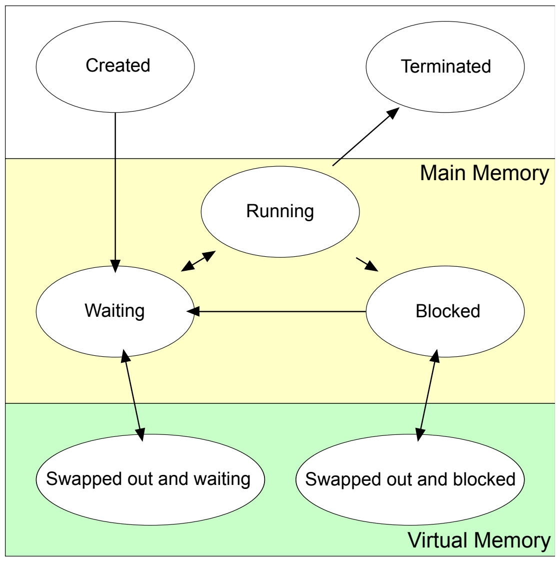


Figure 23: Processus Transitions

Scheduling example 1

- FIFO

T1	T1	T1	T1	T1	T1	T2	T2	T2	T3	T3	T3	T3

- RR

T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T3	T1	T1

- Priority

T2	T2	T2	T1	T1	T1	T1	T1	T1	T3	T3	T3	T3

Scheduling example 1

Scheduler with task interactions

Task 1 (prio = 2)	Task 2 (prio = 2)	Task 3 (prio = 3)
		(not started)
T[1.1]	T[2.1]	T[3.1]
wait (event)	Signal	wait (1 sec.)
T[1.2]	(Task 1, event)	
	T[2.2]	T[3.2]
	Starting Task 3	

Scheduling example 2

Pre-emption after OS signal

Task 1 (prio = 2)	Task 2 (prio = 2)	Task 3 (prio = 3)
T[1.1] Wait (evt)		
	T[2.1] Signal T1	
T[1.2]		
	T[2.2] Start T3	
		T[3.1] Wait 1 sec. T[3.2]

Scheduling example 2

Without pre-emption

Task 1 (prio = 2)	Task 2 (prio = 2)	Task 3 (prio = 3)
T[1.1] Wait (evt)		
	T[2.1] Signal T1 T[2.2] Start T3	
		T[3.1] Wait 1 sec.
T[1.2]		
		T[3.2]

Interruptions

- 2 kind
  - **asynchronous** : emitted by external context (hardware) can arise at any time
  - **synchronous** : emitted by a specific instruction can only arrive in known date for the developer
- processed by the same mechanism

Interruption management

The way interruptions are processed have a large influence on efficiency of the OS

- Control of interruptions : mostly hardware
  - Context switch
  - Source identification
- Processing interruption
  - Short : same context
  - Long : task are called : signal/wait
- Back to original process

Resources management
Those mechanisms are used to give one access to one resource in one time.

- Define some sections as **atomic** : **critical sections**
- Mask interruptions (hardware)
- Scheduler is disabled during this critical section (software)
  - Mutex
    - Block other tasks whatever priority they are
    - Problem to choose the task when Mutex is released
  - Counting Semaphores to manage a set of devices

Synchronization
Force an order of processing instructions of a process

- Mutex
- Rendezvous
- Product/Consume

Uses the following mechanisms :

- Signal an event
- Semaphores use
- Mailboxes
- Rendezvous

Synchronization Limits
Maintain coherency of data : creation of waiting queues

- "Balanced" Systems inefficient
- Priority : Synchronization brings some priority issues where a low priority can block a high priority task

Some solutions :

- Priority based waiting management : unlock view the priority level of the task
- Flag like "availability test"
- Time out on blocking flags

Input/Output
Different policies for CPU scheduling and for Input/Output (IO)

- Access through a specialized layer
- Synchronous processing
- Asynchronous processing
- Difficult to manage priorities
- Bad time management

3.1.4 Real-Time

Services

- **Definition** : a software that schedules computing of tasks in one shot way,manage devices of the system and bring development libraries (Application Programming Interface or API)
- It is organized with :
  - a kernel that schedules tasks and manage devices
  - often it proposes services packaged as modules for example. These service can bring the ability of the system to manage File Systems, network devices or any services that are needed by software

Components
The main components of a RTOS are :

- scheduler, inside every kernel, it is the component that apply different algorithms to manage access to the CPU
- kernel interfaces that give developers the ability to create software
  - tasks
  - semaphores
  - message queue...
- services : actions that a kernel can do on a device or on the system like time measurement, interruption, bus controller,...

3.2 Scheduling

3.2.1 Model

Scheduler

Aim : address any needs of a real-time software like emergency stop, high level action or reactivity need

Taxonomy :

- Off-line/On-line algorithm : static/dynamic algorithm
- Static/Dynamic priorities
- pre-emptive/non pre-emptive algorithms :
  - Ability to have mutual exclusion of a device
  - Scheduler cost low
  - Small efficient

Scheduler

Properties that we look for :

- Feasibility : ability to decide "a priori" that all the constraints will be fulfilled
- Predictability : response time of tasks is predictable.
- Optimality : Optimal if able to find a schedule of every set of feasibility tasks
- Complexity : feasibility test is long and difficult?
- Easiness to implement

Scheduler

Scheduler works only on active tasks :

- Schedule Table : off-line scheduling
- Off-line priorities definition, root of the scheduling
- On-line analysis and scheduling

Analysis recipe

Scheduling problem :

- To Model tasks of the system and their constraints
- To Choose a scheduling algorithm
- To Validate feasibility on a set of tasks
  - Theoretical feasibility : scheduling ability and complexity
  - Empirical feasibility : implement scheduler

Task model
Task families :

- Linked task or not (Precedence constraint).
- Important task or not
- Repetitive or periodic task : activated regularly
- Non periodic task
  - Sporadic : irregular activation but a minimum time between each activation
  - Aperiodic : deadline is less strict and no minimum time between each activation

Task model
Parameters for task i

- S_i : start time when task arrives in the scheduler
- C_i : Computation time needed by the task (Capacity).
- P_i : Period
- D_i : Deadline of the task
- R_i : Earliest activation time
- Aperiodic task is defined by : [2mm]
  (S_i, C_i, D_i, R_i)
- Periodic task is defined by :
  (S_i, C_i, D_i, P_i)

Task model
Let's consider a simplified model :

- Periodic task
- Independent (No precedence constraint)
- Starting time (worst case) : $S_i = 0$
- Deadline equal to Period : $P_i = D_i$

Scheduler Structure

Computation of scheduling information

- On-line or Off-line
- Period, Deadline...

Waiting queue management

- One queue for each priority
- FIFO policy
- Round Robin policy

Election phase

- Priority, deadline, ...

3.2.2 Algorithms

Rate Monotonic Algorithm

Off-line analysis, fixed priority, periodic tasks : Static software

Rationale

- Computation phase : priority = $\frac{1}{period}$
- Election phase : highest priority is chosen

Properties

- Low Complexity
- Optimal Algorithm in fixed priority algorithm set

Rate Monotonic Algorithm

Example

- Task 1 : $C_1 = 6, P_1 = 10$
- Task 2 : $C_2 = 9, P_1 = 30$
- Pre-emptive case
- Non pre-emptive case

Can be scheduled if load rate of the CPU U agrees with the following sufficient condition :

$$U = \sum_{i}^{n} \frac{C_i}{T_i} \leq n \times (2^{\frac{1}{n}} - 1)$$

Critical time theorem

If every tasks both arrive at the same time in a system if they respect their first deadline,

then

every other following deadlines will be respected whatever time they arrive in the system

- it's a necessary and sufficient condition if every tasks arrive at the same time
- else, it is a sufficient condition

If $D_i = T_i$, finishing test is :

$$\forall i, 1 \leq i \leq n \quad \min_{0 \leq t \leq D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1$$

Response time computation

Response time TR: Duration between time when a task begin and time when it is finished. Result can be exact relying on task model. [3mm]

$$TR_i = C_i + \sum_{j \in hp(i)} I_j$$

$$TR_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{TR_i}{P_j} \right\rceil C_j$$

Where $hp(i)$ represents a set of tasks with a higher priority than i.

Response time computation

Computation method : iterative way of evaluation

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

- Let start with $w_i^0 = C_i$
- Fail if $w_i^n > P_i$
- Achieved if $w_i^{n+1} = w_i^n$

Example : with $P_1 = 7$, $C_1 = 3$, $P_2 = 12$, $C_2 = 2$, $P_3 = 20$, $C_3 = 5$. We assume that every task have the same priority and are ran in this order

$$TR_1 = 3 \quad TR_2 = 5 \quad TR_3 = 18$$

Aperiodic tasks with high priorities
Periodic task dedicated to these tasks.

- bring feasibility in the worst case
- dedicated task is not activated if there is no aperiodic task to compute
- easy solution, but sometime we loose some CPU load

Earliest Deadline First EDF algorithm

- Periodic task and aperiodic
- On-line Algorithm, more useful than Rate Monotonic

Rationale :

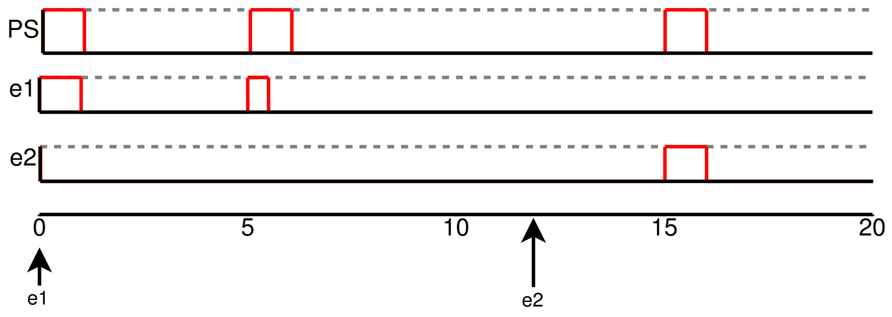
- *Computation phase : Deadline computation*
  With $priority_i(t)$ current priority at t and i the task :
  - Aperiodic task : $priority_i(t) = S_i + D_i - t$
  - Periodic task : $priority_i(t) = S_i(t)$ [last start activation time when time is t] $+P_i - t$
- *Election phase*: Earliest deadline is chosen.(Min)

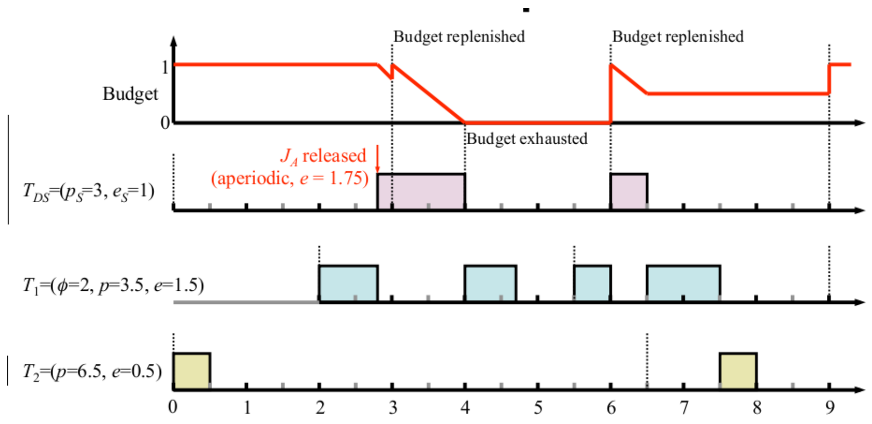
EDF Example
Let consider 2 tasks : [3mm]
$T_1 : C_1 = 6, P_1 = 10$[3mm] $T_2 : C_2 = 9, P_2 = 30$

- Pre-emptive case
- Non pre-emptive case

EDF Properties
Ability to schedule : Pre-emptive case, periodic and independent tasks

- Necessary and sufficient condition if $\forall i, D_i = P_i$; Only necessary if $\exists i, D_i \leq P_i$:
$$U = \sum_{j=1}^{n} \frac{C_j}{P_j} \leq 1$$
- Sufficient condition if $\exists i, D_i \leq P_i$:
$$U = \sum_{j=1}^{n} \frac{C_j}{D_j} \leq 1$$





Server for aperiodic and sporadic jobs

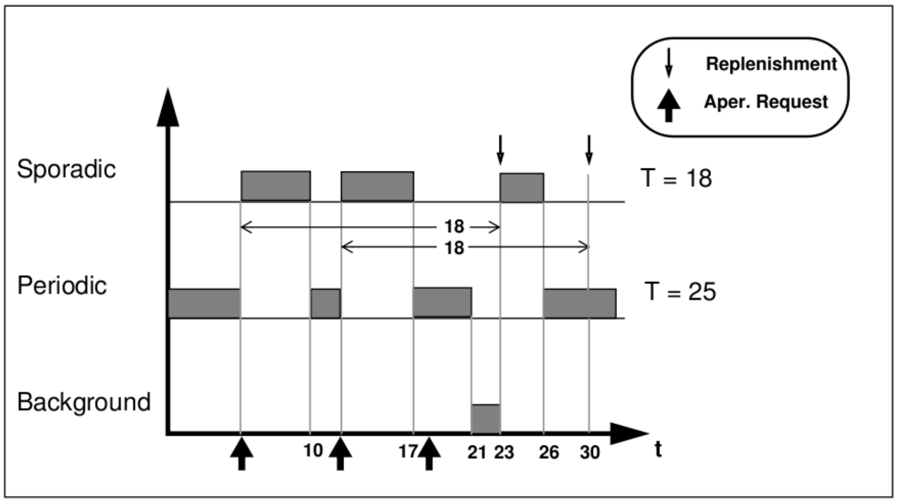
- Periodic server :
  - Idle if waiting queue is empty
  - Running in its time

Server for aperiodic and sporadic jobs

- Deferred server :
  - Time Budget
  - Waiting for task
  - Used to be in highest priority

Server for aperiodic and sporadic jobs

- Sporadic server :
  - Like Deferred server
  - But extra computation is done in background (lowest priority)



3.2.3 Thread Scheduling

Services

Properties :

- Threads and process capability
- Fixed priorities, pre-emptive ⇒ RM easy. A minimum of 32 levels is a mandatory
- One waiting queue for each priority and scheduling policy (SCHED_F IFO, SCHED_RR, SCHED_OTHERS).
- Available Services to specific users (like root)

Standard say that scheduling policies must be able to be applied each time the choice threadprocess exists (ex. : choice of a threadprocess to release a semaphore).

Policies

POSIX.4 policies :

```
#define SCHED_OTHER 0
#define SCHED_FIFO 1
#define SCHED_RR 2
```

- Parameter(s) : extensible to future policies

```
struct sched_param{
    int sched_priority;
    ...
};
```

- Parameters modification:
  - Thread : creation of a thread from an attribute or modification of a running thread.
  - Inherit from a fork() or modify of a running process.

API

sched_get_priority_max	Get max priority value.
sched_get_priority_min	Get min priority value.
sched_rr_get_interval	Get duration of one time unit.
sched_yield	Free the CPU of this thread.
sched_setscheduler	Choose the scheduler policy.
sched_getscheduler	Get the scheduler policy value.
sched_setparam	Set parameters of the scheduler.
sched_getparam	Get parameters of the scheduler.
pthread_setschedparam	Set parameters of the scheduler for thread.
pthread_getschedparam	Get parameters of the scheduler for thread.

The last 2 function are working only on threads, other functions can be applied on process/thread.

Example

```
struct sched_param parm;
int res=-1;
...
/* Task T1 ; P1=10 */
parm.sched_priority=15;
res=sched_setscheduler(pid_T1,SCHED_FIFO,&parm)
if(res<0)
perror("sched_setscheduler task T1");
/* Task T2 ; P2=30 */
parm.sched_priority=10;
res=sched_setscheduler(pid_T2,SCHED_FIFO,&parm)
if(res<0)
perror("sched_setscheduler task T2");
```

3.3 Management

3.3.1 Process

Fork

```
#include <unistd.h>
pid_t fork(void);
```

Kill - Signal

```
#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);
```

3.3.2 Posix Thread

POSIX Threads

Defined in chapter POSIX.4a (Portable Operating System Interface). This chapter describe both threads and synchronization tools that are close to threads (ex : mutex).

Properties :

- A POSIX Thread is defined with a identifier that is local to the process.It owns its stack, its context and a set of attributes that describe its behaviour.
- A POSIX Thread can be implemented in user space or in kernel space ⇒ Standard describe only the interface not the way it is coded.

POSIX Threads

- Code must be re-entrant (or "thread safe") ⇒ code is able to be computed in multiple instances in a safe way
- re-entrant code means :
  - don't manipulate shared variables.
  - or it manipulate shared variables in a critical section (that mean no other is able to access this variable when one is manipulating it) .
- Everything must be re-entrant user code but also system libraries (ex. libc.so)

POSIX threads

pthread_create	Creation of a thread. Paramters : code, attributes, arg.
pthread_exit	End of a thread. Parameter : return error value.
pthread_self	returns id of the running thread
pthread_cancel	Destroy a thread. Parameter : id of the thread.
pthread_join	Suspend a thread until another is not finished.
pthread_detach	Suppress parent link between threads.

POSIX Threads

pthread_kill	Emit a signal to a thread.
pthread_sigmask	Modify signal mask of a thread.

In a POSIX.4a system, some services have a different semantic.

- fork() : create a new process containing a thread where the main() function is computed
- exit() : finish a process and all threads it contains.

POSIX Threads

Creation and join example of a thread

```
#include <pthread.h>
void* th(void* arg)
{
    printf("I am thread %d ;
    process %d\n",
    pthread_self(),getpid());
    pthread_exit(NULL);
}
```

POSIX Threads

Creation and join example of a thread

```
#include <pthread.h>
void* th(void* arg)
{
    printf("I am thread %d ;
    process %d\n",
    pthread_self(),getpid());
    pthread_exit(NULL);
}
```

POSIX Threads

```
int main(int argc, char* argv[])
{
    pthread_t id1 ,id2;
    pthread_create(&id1,NULL,th,NULL);
    pthread_create(&id2,NULL,th,NULL);
    pthread_join(id1,NULL);
    pthread_join(id2,NULL);
    printf("End of main thread %d ;
    process %d\n",
    pthread_self(),getpid());
    pthread_exit(NULL);
}
```

POSIX Threads

- With Solaris :

```
gcc -D_REENTRANT create.c -lpthread -lrt
>a.out
  I am thread 4 ; process 5539
  I am thread 5 ; process 5539
  End of main thread 1 ; process 5539
```

- With Linux :

```
gcc -D_REENTRANT create.c -lpthread
>a.out
  I am thread 1026 ; process 1253
  I am thread 2051 ; process 1254
  End of main thread 1024 ; process 1251
```

Thread Attributes

Thread Attributes : properties of a thread that are defined during its creation. No heritage between father and son threads.

Example of attributes :

Attribute Name	Meaning
detachstate	`pthread_join` possible or not
policy	scheduler policy
priority	level of priority of the thread (≥ 0)
stacksize	Size of the allocated stack

Thread Attributes

During creation of a thread, an structure of type `pthread_attr_t` can be filled and given if we want to specify none default attributes to the new thread :

pthread_attr_init	Creation of a default attribute structure.
pthread_attr_delete	Destruction of attribute structure.
pthread_attr_setATT	set the value of "ATT" attribute.
pthread_attr_getATT	set the value of "ATT" attribute.

where ATT is replace by the name of the attribute.

Thread Attributes

```
#include <pthread.h>
void* th(void* arg)
{
  printf("I am thread %d \n",
  pthread_self());
}
```

Thread Attributes

```
int main(int argc, char* argv[])
{
    int i;
    pthread_t id;
    pthread_attr_t attr;
    struct sched_param param;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,
    PTHREAD_CREATE_DETACHED);
    pthread_attr_setschedpolicy(&attr,
    SCHED_FIFO);
    param.sched_priority=1;
    pthread_attr_setschedparam(&attr,&param);
    for(i=1;i<10;i++)
        pthread_create(&id,&attr,th,NULL);
}
```

Thread Attributes

The TSD (Thread Specific Data area) : is an area of memory where are stored specific information of each thread.

Allow extension of regular attributes.

pthread_key_create	Creation of a key.
pthread_key_delete	Destruction of the key.
pthread_getspecific	get pointer value linked to the key of the running thread.
pthread_setspecific	set pointer value linked to the key of the running thread.

Thread Attributes

Creation of a new attribute :

```
pthread_key_t cd_key;

int pthread_cd_init(void)
{
return pthread_key_create(&cd_key,NULL);
}

char* pthread_get_cd(void)
{
return (char*)pthread_getspecific(cd_key);
}
```

Thread Attributes

```
int pthread_set_cd(char* cd)
{
    char* mycd = (char*)malloc(sizeof(char)*100);
    strcpy(mycd,cd);
    return pthread_setspecific(cd_key,mycd);
}
int main(int argc, char* argv[])
{
    pthread_cd_init();
    pthread_set_cd("/here/dir");
    printf("My local directory is %s\n",
    pthread_get_cd());
}
```

3.4 Time Services

3.4.1 Posix Time

Time manipulation

Time linked services :

- What time is it ?
- Block a thread/process during a defined time.
- Wake up a thread/process regularly (timer) ⇒ periodic tasks.

Precision of these services :

- Linked to what hardware we have (clock circuit), its features (interruption period) and also to the software that use it (interruption handler).
- Ex : Linux Intel : circuit activated periodically (10 ms) ⇒ waking between 10 to 12 ms.

Time Manipulation : POSIX Extensions

- Many timers are available ⇒ many physical clocks, profiling.
- Need at least one real-time clock : CLOCK_REALTIME (less than 20 ms precise).
- timespec structure ⇒ "theoretical" precision close to micro-seconds ....
- Available services :
- Get and set clocks.
- Put a task in sleep mode.
- Link periodic timer to UNIX signals ; and maybe with real-time signals. With or without automatic restart.

Time Manipulation

clock_gettime	Get clock value.
clock_settime	Set clock value.
clock_getres	Get clock resolution.
timer_create	Create a timer.
timer_delete	Remove a timer.
timer_getoverrrun	Give the number of unprocessed signals
timer_settime	Activate a timer.
timer_gettime	Get time to the end of the timer
nanosleep	Block a process/thread during a defined duration.

Time Manipulation : Example

```
int main(int argc, char * argv[]){
    timer_t monTimer; struct sigaction sig;
    struct itimerspec ti;
    timer_create(CLOCK_REALTIME,NULL,&monTimer);
    sig.sa_flags=SA_RESTART;
    sig.sa_handler=trop_tard;
    sigemptyset(&sig.sa_mask);
    sigaction(SIGALRM,&sig,NULL);
    ti.it_value.tv_sec=capacite;
    ti.it_value.tv_nsec=0;
    ti.it_interval.tv_sec=0; // here timer is not
    ti.it_interval.tv_nsec=0; // automatically restarted
    timer_settime(monTimer,0,&ti,NULL);
```

Time Manipulation : Example

```
    printf("Begin\n");
    while(go>0)
        printf("I am working .....\n");
    printf("Unblocked by timer : deadline missed ..\n");
}
```

```
int go=1;
void too_late(int sig){
    printf("Signal %d received\n",sig);
    go=0;
}
```

Execution :

```
Begin
I am working .....
I am working .....
I am working .....
I am working .....
```

```
I am working .....
Signal 14 received
Unblocked by timer : deadline
missed ..
```

3.4.2 Xenomai Time services

API Timer Management System

API :

- void **rt_timer_spin** (RTIME ns) : busy clock
- int **rt_timer_set_mode** (RTIME nstick) : TM_ONESHOT or period time

Hardware level :

- oneshot
- periodic

3.5 Synchronization

3.5.1 Mutex

Mutex

- Optimized Semaphores to implement critical section.
- Waiting queue manager : depends of scheduling policy (SCHED_FIFO, SCHED_OTHER, ...). Default behaviour : threads are waked in decreasing order of priority
- Behaviour is defined by a set of attributes, main parameters are :

Attribute name	Meaning
protocol	Inheriting Protocol used
pshared	Inter-process or inter-thread mutex
ceiling	Priority ceiling

Rationale

Programming mechanism that can block a thread while a condition is not true.[3mm]

- Solution is based on a mutex and a Var Cond
- Waking signal has no memory : if nothing is waiting for the condition, waking signal is lost ($\neq$ counting semaphores).
- Waiting queue management : depend on scheduling policy (SCHED_FIFO, SCHED_OTHER, etc ...). Default behaviour : threads are waked in decreasing order of priority.

Mutex

pthread_mutex_init	Initialize a mutex.
pthread_mutex_lock	Lock the mutex in blocking manner eventually.
pthread_mutex_trylock	Non blocking try to lock the mutex
pthread_mutex_unlock	Release the mutex lock.
pthread_mutex_destroy	Destruct the mutex.
pthread_mutexattr_init	Initialize an attribute structure
pthread_mutexattr_setATT	Set attribute ATT.
pthread_mutexattr_getATT	Get attribute ATT value.

o ATT is one of the attributes of the mutex.

API

pthread_cond_init	Initialize a variable.
pthread_cond_destroy	Destruct a variable.
pthread_cond_wait	Wait a waking signal coming from a condition.
pthread_cond_signal	Signal that a condition is true to one thread
pthread_cond_broadcast	Signal that a condition is true to all threads

3.5.2 Variable Conditionnal

Mechanism

A classical program where a thread modify a variable : a mutex is used (var_mutex) and a condition is used (var_cond).

```
pthread_mutex_lock(&var_mutex);
/* Modify the variable in critical
section */
var = .... ;

/* If the condition is fulfilled, we
alert the blocked thread */
if(condition(var))
    pthread_cond_signal(&var_cond);

pthread_mutex_unlock(&var_mutex);
```

Mechanism

Classical program that wait for a variable to be modified :

```
pthread_mutex_lock(&var_mutex);
while(!condition(var))
{
    /* If the condition is not true,
wait */
    pthread_cond_wait(&var_cond, &var_mutex);
}
/* use the variable in critical section */
.....

pthread_mutex_unlock(&var_mutex);
```

Attention pthread_cond_wait is making some implicit lock and unlock of the mutex.

Example

```
int y=2, x=0;
pthread_mutex_t mut;
pthread_cond_t cond;
void* th(void* arg)
{
    int cont=1;
    while(cont){
        pthread_mutex_lock(&mut);
        x++;
        printf("x++\n");
        if (x > y){
            pthread_cond_signal(&cond);
            cont=0; }
        pthread_mutex_unlock(&mut);}
}
```

Example

```
int main(int argc, char* argv){
    pthread_t id;
    pthread_mutex_init(&mut,NULL);
    pthread_cond_init(&cond,NULL);
    pthread_mutex_lock(&mut);
    pthread_create(&id,NULL,th,NULL);
    while (x <= y)
        pthread_cond_wait(&cond, &mut);
    printf("x>y is true\n");
    pthread_mutex_unlock(&mut);}
```

Execution :

```
x++                                x++
x++                                x>y is true
```

3.5.3 Counting Semaphore

Counting semaphore

- Semaphore: waiting queue + counter.
- No management of priority inversion : case must be treated in developer level.
- Used for synchronization and inter-process or inter-thread mutual exclusion.

- Waiting queue management : depend on scheduling policy (SCHED_FIFO, SCHED_OTHER, etc ...). Default behaviour : threads are waked in decreasing order of priority.
- Two kind of semaphore : named and unnamed semaphore.

Counting Semaphore

sem_open	Connection to a named semaphore.
sem_close	Disconnection to a named semaphore.
sem_unlink	Destruction of a named semaphore.
sem_init	Initialisation of a unnamed semaphore..
sem_destroy	Destruction of a unnamed semaphore.
sem_post	Freeing a semaphore.
sem_wait	Acquiring a semaphore.
sem_trywait	Non blocking acquiring of a semaphore.

Counting Semaphore Example

```
#include <pthread.h>
#include <semaphore.h>
sem_t sem;
int main(int argc, char* argv[]){
    pthread_t id;
    struct timespec delay;
    sem_init(&sem,0,0);
    pthread_create(&id,NULL,th,NULL);
    deli.tv_sec=4; deli.tv_nsec=0;
    nanosleep(&delay,NULL);
    printf("main thread %d : free from
              the other thread \n",pthread_self());
    sem_post(&sem);
    pthread_exit(NULL);
}
```

Counting Semaphore Example

```
void* th(void* arg)
{
    printf("thread %d waiting\n",
    pthread_self());
    sem_wait(&sem);
    printf("thread %d unblocked \n",
    pthread_self());
}
```

Execution

```
thread 4 waiting
main thread 1 : free from
the other thread
thread 4 unblocked
```

3.5.4 Xenomai Native API Synchronisation tools

API Mutexes

int **rt_mutex_create** (RT_MUTEX *mutex, const char *name)
int **rt_mutex_acquire** (RT_MUTEX *mutex, RTIME timeout)
int **rt_mutex_release** (RT_MUTEX *mutex)

API Condition variables

int **rt_cond_create** (RT_COND *cond, const char *name)
int **rt_cond_signal** (RT_COND *cond)
int **rt_cond_broadcast** (RT_COND *cond)
int **rt_cond_wait** (RT_COND *cond, RT_MUTEX *mutex, RTIME timeout)

API Counting Semaphores

int **rt_sem_create** (RT_SEM *sem, const char *name,
 unsigned long icount, int mode)
int **rt_sem_p** (RT_SEM *sem, RTIME timeout)
int **rt_sem_v** (RT_SEM *sem)
int **rt_sem_broadcast** (RT_SEM *sem)

3.6 Messages and Communication

3.6.1 Signals

Real-Time Signals

Signal : event delivered in asynchronous way to a process/thread (software interruption).

- Blocked signals (or masked) pending or delivered.
- Default behaviour can be modified by user.
- One signal register for each thread/process.

Real-Time Signals

Existing mechanism in POSIX.1 but with the following issues :

- Implementation pending signal register = unsafe delivery (possible loss).
- Emitting order not respected when they are delivered.
- Does not contain many information and few of them are available for user : not adapted to IPC implementation.
- Not efficient (slow : high latency).

Real-Time Signals : POSIX Interface

kill	Emission of a signal.
sigaction	Connection of a handler to a signal.
sigemptyset	Initialize with all signals disabled.(mask all signals)
sigfillset	Initialize with all signals enabled.(unmask all signals)
sigaddset	Add a signal in the set(unmask this signal).
sigdelset	Remove a signal from the set(mask this signal).
sigismember	Check if the signal is enabled in the set.
sigsuspend	Block a process until a signal is received.
sigprocmask	Install a mask.

Signals : Non RT Example

```
void handler(int sig){
    printf("Signal %d received\n",sig);}

int main(int argc, char * argv[]){
    struct sigaction sig;
    sig.sa_flags=SA_RESTART;
    sig.sa_handler=handler;
    sigemptyset(&sig.sa_mask);
    sigaction(SIGUSR1,&sig,NULL);
    while(1); // program computes !!
}
```

Execution :

```
$sig &
$kill -USR1 14090
Signal 10 received
```

Real-Time signals : POSIX Extended

Range of new signals numbered from SIGRTMIN to SIGRTMAX (that represent a minimum of RTSIG_MAX signals)

- Possibility to use a value linked to this signal.
- No loss of signal : using of a queue for pending signals.
- Ordered delivery : respecting scheduling policy chosen + priority linked to a signal ⇒ SIGRTMIN has the highest priority.
- Emitted with a kill, sigqueue, with a timer or by an asynchronous IO.

Real-Time Signals : POSIX Extended

Complementary interfaces of POSIX.4 :

sigqueue	Emit a real-time signal.
sigwaitinfo	Wait for a signal without handler execution
sigtimedwait	Idem above + timeout on blocking time.

Real-Time Signals : RT Example

```
int main(int argc, char * argv[]){
    struct sigaction sig;
    union sigval val;
    int cpt;
    sig.sa_flags=SA_SIGINFO;
    sig.sa_sigaction=handler;
    sigemptyset(&sig.sa_mask);
    if(sigaction(SIGRTMIN,&sig,NULL)<0)
    perror("sigaction");
    for(cpt=0;cpt<5;cpt++){
        struct timespec delai;
        delai.tv_sec=1; delai.tv_nsec=0;
        val.sival_int=cpt;
        sigqueue(0,SIGRTMIN,val);
        nanosleep(&delai,NULL);
    }
}
```

RealTime Signals : RT Example

```
void handler (int sig, siginfo_t *sip, void *uap)
{
    printf("Received signal %d, val = %d \n",
    sig,sip->si_value.sival_int);
}
```

Execution :

```
\$rt-sig
Received signal 38, val = 0
Received signal 38, val = 1
Received signal 38, val = 2
Received signal 38, val = 3
Received signal 38, val = 4
```

3.6.2 Mail Queue

Mechanism

With UNIX, Communication mechanism is the pipeline (named or not) $\Rightarrow$ too abstract for real-time constraints.[2mm]

Mail queues in POSIX.4 have the same features :

- Priority in emission/reception.
- Can work in Non-blocking
- Can preallocate resources.
- Can communicate in Inter-Process and Inter-Thread mode.
- Unfortunately, priority inversion issues are not addressed.

API

mq_open	Creation or connection to a queue.
mq_unlink	Destruction of a queue.
mq_receive	Reception of the oldest and the most important mail.
mq_send	Emission of a mail with a given priority
mq_close	Disconnection to a queue.
mq_notify	Notification that there is a new mail.
mq_setattr	Set attributes of the queue.
mq_getattr	Get attributes of the queue.

Example

```
/* Emission with a priority 1 (from 0 to
MQ_PRIO_MAX) */
    mq_send(id,buff,100,1);
    pthread_create(&tid,NULL,consumer,NULL);
    pthread_exit(NULL);
}

void* consumer(void* arg){
    mqd_t id;
    char buff [100];
    id=mq_open("/myqueue",O_RDONLY);
    mq_receive(id,buff,100,NULL);
    printf("msg = %s\n",buff);
    mq_unlink("/myqueue");
    pthread_exit(NULL);
}
```

Example

```
#include <pthread.h>
#include <mqueue.h>
int main(int argc, char* argv[])
{
    pthread_t tid;
    mqd_t id;
    char buff [100];
    struct mq_attr attr;
    attr.mq_maxmsg=100;
    attr.mq_flags=0;
    attr.mq_msgsize=100;
    id=mq_open("/mafile",O_CREAT|O_WRONLY,
444,&attr);
    strcpy(buff,"Hi!!");
    ...
}
```

3.6.3 Xenomai Native API

API Event flags groups

- Synchronizing object based on a long word structure.
- Any bit in the word can be used as a user flag
- Task oriented signal mechanism
- Conjunctive or Disjunctive

API Event flags groups

int	**rt_event_create** (RT_EVENT *event, const char *name, unsigned long ivalue, int mode)
int	**rt_event_signal** (RT_EVENT *event, unsigned long mask)
int	**rt_event_wait** (RT_EVENT *event, unsigned long mask, unsigned long *mask_r, int mode, RTIME timeout)
int	**rt_event_clear** (RT_EVENT *event, unsigned long mask, unsigned long *mask_r)

API Messages queue

int	**rt_queue_create** (RT_QUEUE *q, const char *name, size_t poolsize, size_t qlimit, int mode)
void *	**rt_queue_alloc** (RT_QUEUE *q, size_t size)
int	**rt_queue_free** (RT_QUEUE *q, void *buf)
int	**rt_queue_send** (RT_QUEUE *q, void *mbuf, size_t size, int mode)
int	**rt_queue_write** (RT_QUEUE *q, const void *buf, size_t size, int mode)
ssize_t	**rt_queue_receive** (RT_QUEUE *q, void **bufp, RTIME timeout)
ssize_t	**rt_queue_read** (RT_QUEUE *q, void *buf, size_t size, RTIME timeout)

API Messages pipe

int	**rt_pipe_create** (RT_PIPE *pipe, const char *name, int minor, size_t poolsize)
ssize_t	**rt_pipe_receive** (RT_PIPE *pipe, RT_PIPE_MSG **msgp, RTIME timeout)
ssize_t	**rt_pipe_read** (RT_PIPE *pipe, void *buf, size_t size, RTIME timeout)
ssize_t	**rt_pipe_send** (RT_PIPE *pipe, RT_PIPE_MSG *msg, size_t size, int mode)
ssize_t	**rt_pipe_write** (RT_PIPE *pipe, const void *buf, size_t size, int mode)
ssize_t	**rt_pipe_stream** (RT_PIPE *pipe, const void *buf, size_t size)
RT_PIPE_MSG *	**rt_pipe_alloc** (RT_PIPE *pipe, size_t size)
int	**rt_pipe_free** (RT_PIPE *pipe, RT_PIPE_MSG *msg)
int	**rt_pipe_flush** (RT_PIPE *pipe, int mode)

Example : Pipe Kernel side

```
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
#include <native/pipe.h>
#define TASK_PRIO 0 /* Highest RT priority */
#define TASK_MODE T_FPU|T_CPU(0) /* Uses FPU, bound to CPU #0 */
#define TASK_STKSZ 4096 /* Stack size (in bytes) */

RT_TASK task_desc;
```

```
RT_PIPE pipe_desc;

void task_body(void){
RT_PIPE_MSG *msgout, *msgin;
int err, len, n;
for (;;) {
/* ... */
len = sizeof("Hello");
/* Get a message block of the right size in order to
initiate the message-oriented dialog with the
user-space process. */
msgout = rt_pipe_alloc(len);
if (!msgout)
fail();
```

Example : Pipe Kernel side

```
#include <native/pipe.h>
RT_PIPE pipe_desc;
int init_module (void)
{
int err;
/* Connect the kernel-side of the message pipe to the
special device file /dev/rtp7. */
err = rt_pipe_create(&pipe_desc,"MyPipe",7,NULL);
...
}
```

Example : Pipe User side

```
From a regular Linux process:
#include <native/pipe.h>
int pipe_fd;
int main (int argc, char *argv[])
{
/* Open the Linux side of the pipe. */
pipe_fd = open("/dev/rtp7",O_RDWR);
/* ou grace au service de registres*/
pipe_fd = open("/proc/xenomai/registry/native/pipes/MyPipe",O_RDWR);
...
/* Write a message to the pipe. */
write(pipe_fd,"hello world",11);
...
}
```

3.7 IO Control

3.7.1 Memory

Memory Management

A shared time system leads to possible time indeterminism because :

- Dynamic memory allocation.
- Page swap : swapin=swapout.

Solution : limit dynamic memory allocation and lock pages in central memory (POSIX.4 Interface) :

- mlockall()=munlockall() : lock/unlock swap with all memory pages of this process..
- mlock()=munlock() : lock/unlock a range of addresses.

! **Warning** : mlock() is not portable (because there is no memory model in POSIX standard).

API Memory heap

int	**rt_heap_create** (RT_HEAP *heap, const char *name, size_t heapsize, int mode)
int	**rt_heap_alloc** (RT_HEAP *heap, size_t size, RTIME timeout, void **blockp)
int	**rt_heap_free** (RT_HEAP *heap, void *block)

3.7.2 IO

Posix AIO

- aio_read
- aio_write
- aio_return, aio_error
- aio_cancel
- aio_fsync
- aio_suspend

Example :

```
sa.sa_sigaction = aioSigHandler;
if (sigaction(IO_SIGNAL, &sa, NULL) == -1) errExit("sigaction");
aiocb.aio_nbytes = BUF_SIZE;
aiocb.aio_reqprio = 0;
aiocb.aio_offset = 0;
aiocb.aio_sigevent.sigev_notify = SIGEV_SIGNAL;
aiocb.aio_sigevent.sigev_signo = IO_SIGNAL;
aiocb.aio_sigevent.sigev_value.sival_ptr = &ioValue;
aio_read(&aiocb);
err = aio_error (&aiocb));
switch (err) {
   case 0:                     printf("I/O succeeded\n");break;
   case EINPROGRESS:      printf("In progress\n");break;
   case ECANCELED:             printf("Canceled\n");break;
   default:                            errMsg("aio_error");break;
   }
aio_return(&aiocb);
```

API Interrupt management

int **rt_intr_create** (RT_INTR *intr, const char *name, unsigned irq, rt_isr_t isr, rt_iack_t iack, int mode)[Kernel]
int **rt_intr_create** (RT_INTR *intr, const char *name, unsigned irq, int mode)[User]
int **rt_intr_enable** (RT_INTR *intr)
int **rt_intr_disable** (RT_INTR *intr)
int **rt_intr_wait** (RT_INTR *intr, RTIME timeout)

Example : User Interruption

```
#include <sys/mman.h>
#include <native/task.h>
#include <native/intr.h>

#define IRQ_NUMBER 7 /* Intercept interrupt #7 */
#define TASK_PRIO 99 /* Highest RT priority */
#define TASK_MODE 0 /* No flags */
#define TASK_STKSZ 0 /* Stack size (use default one) */

RT_INTR intr_desc;
RT_TASK server_desc;

void irq_server (void *cookie){
for (;;) {
/* Wait for the next interrupt on channel #7. */
err = rt_intr_wait(&intr_desc,TM_INFINITE);
if (!err) {
/* Process interrupt. */
}
}
}
```

Example : User Interruption

```
int main (int argc, char *argv[]){
int err;
mlockall(MCL_CURRENT|MCL_FUTURE);
/* ... */
err = rt_intr_create(&intr_desc,"MyIrq",IRQ_NUMBER,0);
/* ... */
err = rt_task_create(&server_desc,
"MyIrqServer",
TASK_STKSZ,
TASK_PRIO,
TASK_MODE);
if (!err)
rt_task_start(&server_desc,&irq_server,NULL);
/* ... */
}
void cleanup (void){
rt_intr_delete(&intr_desc);
rt_task_delete(&server_desc);}
```

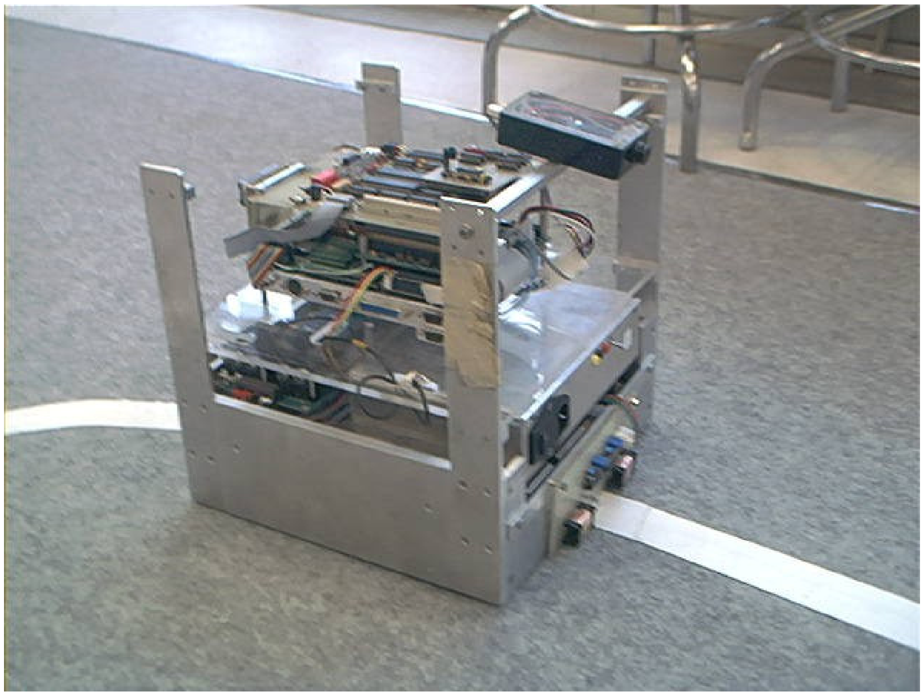


Figure 24: Autonomous Mobile Robot

4 OS Driver Programming

4.1 First Idea : From Scratch

4.1.1 Example

RobModex

- ISA Card
  - 0x100 : Base Address
  - 0x01 - 0x04 : Timer (IRQ 5)
  - 0x05 - 0x08 : DAC1
  - 0x09 - 0x12 : DAC2
  - 0x13 - 0x14 : 2 Wheel Encoder Registers

Closed Loop PID

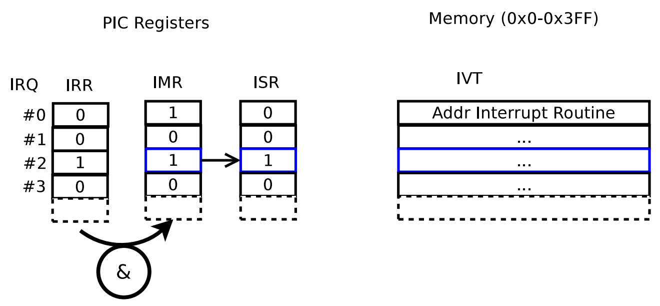
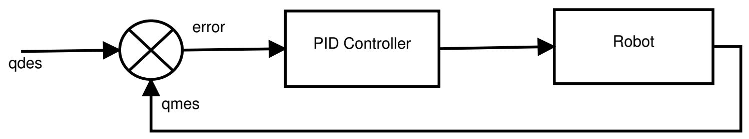


Figure 25: IRQ Process



```
variables qmes, qdes
error(k) =
error_cumul(k) =
error_diff(k)=



qcons=
olderror = error
```

4.1.2 x86 example

Simple IRQ x86

- 0x20
  - Command Register
  - Status Register
- 0x21
  - Int Mask Register
- Internal Registers
  - Int. Req. Reg.
  - In Service Register
- System Memory
  - Interrupt Vector Table (4 bytes)

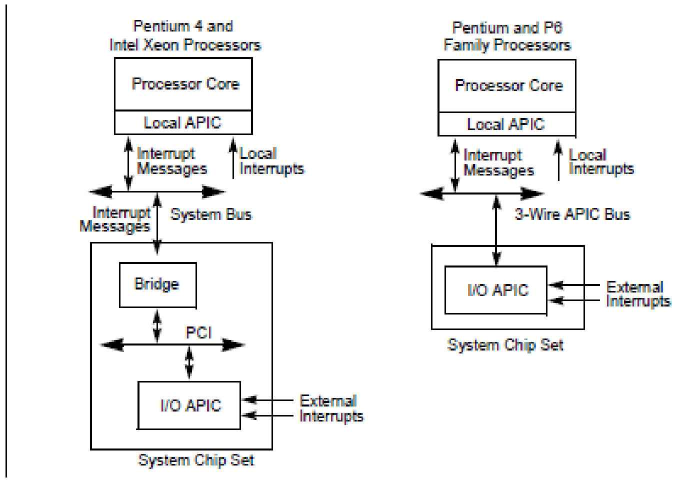


Figure 26: Interruption Recent Mechanism more complex

- Byte 0 : Offset Low Address of the **Interrupt Routine** (Handler)
- Byte 1 : Offset High Address of the IR
- Byte 2 : Segment Low Address of the IR
- Byte 3 : Segment High Address of the IR Offset

Piece of code :
turboC DOS example

```
#define IMR 0x21
void _install_int_function(int IRQn,
               void interrupt (*_new_int_function)())
{
 int inter = IRQn + 8;
 _disable(); //disable interrupts
 //save the old interrupt vector
 _old_int_function=_dos_getvect(inter);
 //install the new interrupt vector
 _dos_setvect(inter,_new_int_function);
 //save the state of the 8259A IMR register
 _old_mask=inportb(IMR);
 //Set new value for IMR register
 outportb(IMR,_old_mask&_interrupt_mask(IRQn));
 _enable(); //enable interrupts
}
```

I/O APIC : Advanced Programmable Interrupt Controller

IRQ

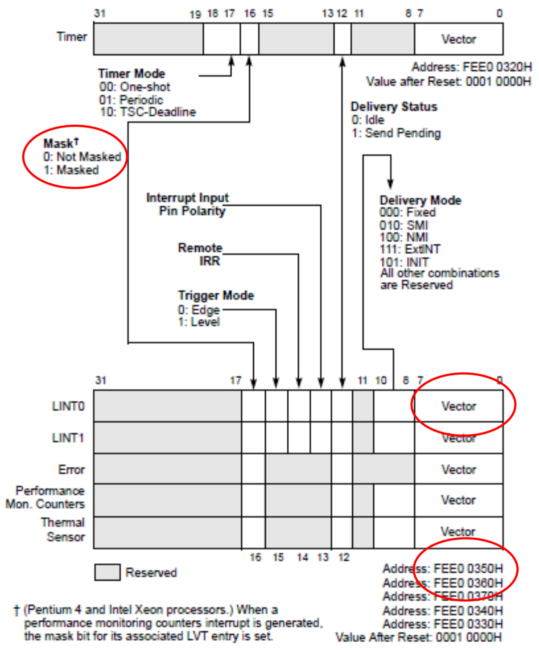


Figure 27: Intel SMP Interrupt Systems

Bus : ISA

- Ax : Addresses
- Dx : Data
- IRQs : 3,4,5,6,7
- AEN : DMA

Bus : ISA

```
#define ISAADDR
short data;
void interrupt _new_int_function()
{

_disable(); //disable interrupts
data=inb(ISAADDR); //getDataFromISA
//that the ISA Component has requested
//to tell the system the interrupt service function has finished
_end_interrupt();
_enable(); //enable interrupts again
}
```

Digital/Analog Converter

RobModex Application

```
#define ISAADDR  .....
short data;

void interrupt _new_int_function()
{
_disable(); //disable interrupts
data=inb(             );








outb(         ,          );
_end_interrupt();
_enable(); //enable interrupts again
}
```

8 Bit XT Bus – top view

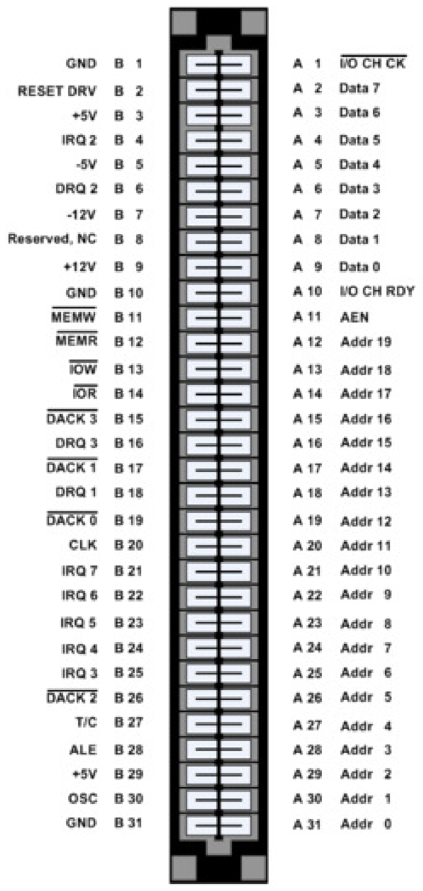


Figure 28: ISA Port

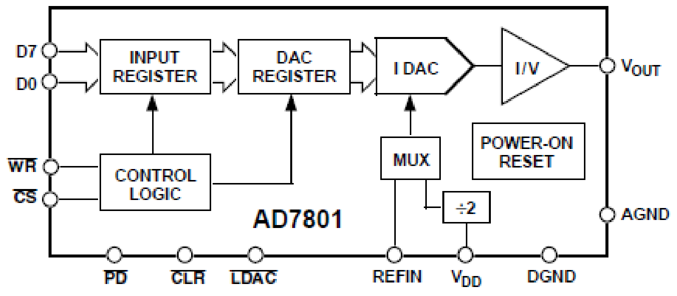


Figure 29: Digital Analog Converter Schematic

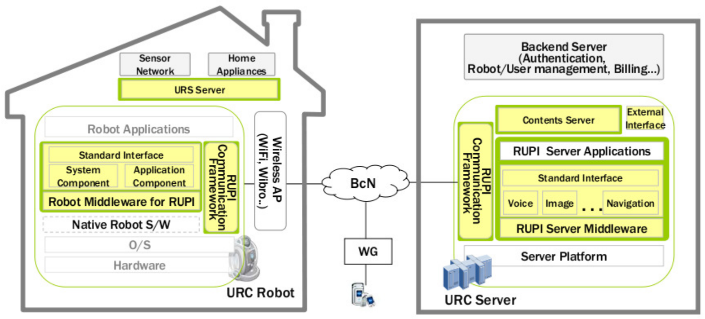


Figure 30: RUPI

4.1.3 From scratch approach

Conclusion

- More Complex = More code
- No- reusability of pieces of code

4.2 Second Idea : Operating System Approach

4.2.1 Introduction

Context

- OS Hardware
- Native Robot SW

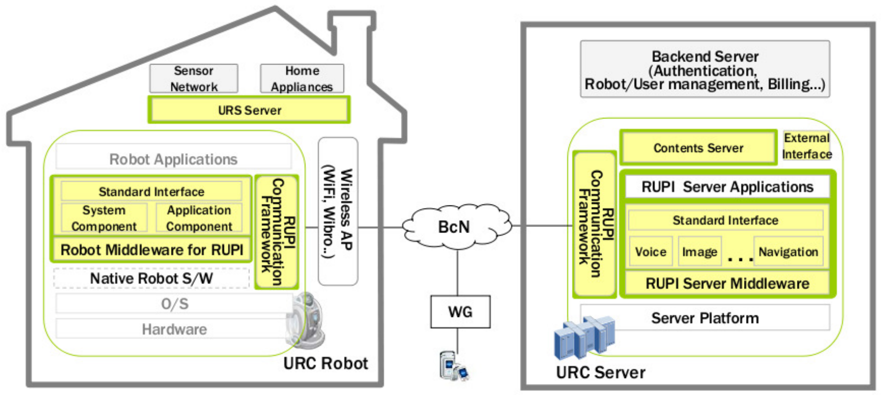


Figure 31: Robot Unified Platform Initiative

- MiddleWare
- Communication Framework

State of Art

- OS Hardware
  - Linux WindowsCE VxWorks
- Native Robot SW
  - Xenomai RTAI RTLinux OROCOSRT
- MiddleWare
  - RoSta Robotics Standards : OROCOS, Microsoft Robotics Studio, URBI, Robotic Operating System (ROS), ORCA. MARIE...
- Communication Framework
  - OROCOS, Microsoft Robotics Studio, URBI, Robotic Operating System, OpenJaus

4.2.2 Linux Kernel Introduction

History

Creator: Linus Torvalds for personnal reasons

- understand Operating System

- funding no cheap OS present on the market place
- Minix had too many limits

Main initial choices taken by Linus

- using GNU licenses tools compiler and tools
- share sources on Internet
- GNU GPL License used
- collaborative work ⇒ quick emergence of a developer community

Key dates

- First stable version (1.0) in 1994 ;
- POSIX compatibility added in version 1.2 (1995) ;
- Support Multi Processors (SMP) and portability improvement in versions 2.0 (1996) ;
- Improvement of general performances, increase of number of platform and number of supported devices in versions 2.2 (1999) ;
- Increasing performances with SMP architectures and network layer in versions 2.4 (2001).
- wide set of improvements internally and in the API (pre-emptive, 64 bit/embedded architectures, unified devices management, ...) (2003).

Features

- Monolithic Kernel
- UNIX type architecture all files concept
- MultiTasks
- MultiUsers
- MultiProcessors
- Multiplatform

Properties

- Dynamic load of kernel modules
- Inter Processes memory protection
- Load of executable on demand
- Page sharing between each executable
- Dynamic cache size management
- Shared libraries supported
- ...

4.3 Linux Kernel

4.3.1 Main Concepts

User/Kernel World
Material protection using CPU mechanisms.
Two distinct worlds

- Kernel space where all is permitted even the worst
- User space where possibilities are restricted with safety mechanisms (ex : memory access)

Schema

User/Kernel Transitions
Three kind of transitions user/kernel :

- system calls ;
- interruptions ⇒ Interruption controller in the kernel ;
- exceptions (ex : illegal memory access , illegal instructions ...).

Limited number of entry points in the kernel ($\tilde{1}90$) for users ⇒ system calls ;
Each user process can be divided in two main parts :

- a kernel part ⇒ System calls (ex : **open()**, **read()**...) ;
- a User part ⇒ all the other things (ex : management, algorithms...).

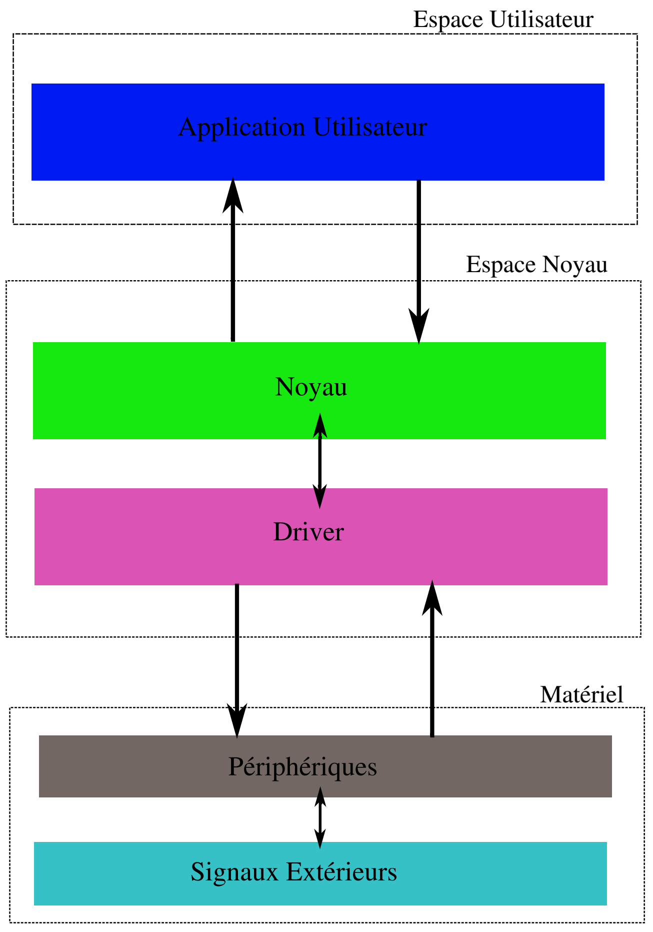


Figure 32: User Kernel Hardware links

Rationale

Main rule
”Always try to bring complexity outside of the kernel ”
Consequences

- kernel must stay as small as possible
- system calls must be limited in their numbers
- all the complexity are in the library called **libc** or
- in user software

Motivations

$\boxed{\text{Kernel device drivers or user mode software?}}$

Kernel device drivers:

- Direct Access : Memory and devices ;
- quickness (less software layers) ;
- manage concurrent accesses to devices ;
- manage access rights.

User software solution :

- keep complexity outside of the kernel ;
- software are running in safe mode ;
- easy to debug.

Ex : Ghostscript (printers), Sane (scanners)...

Kernel Structures

Many structures are commonly used in the kernel :

- Complex structures (ex : task_struct defined in <linux/sched.h>) ;
- complex linked lists (ex : structures list vm_area_struct organised as AVL tree - <linux/mm.h>) ;
- pseudo-objects composed of pointers on functions structures as a method of it (ex : module structure defined in <linux/module.h>).

4.3.2 Components and mechanisms

Memory management

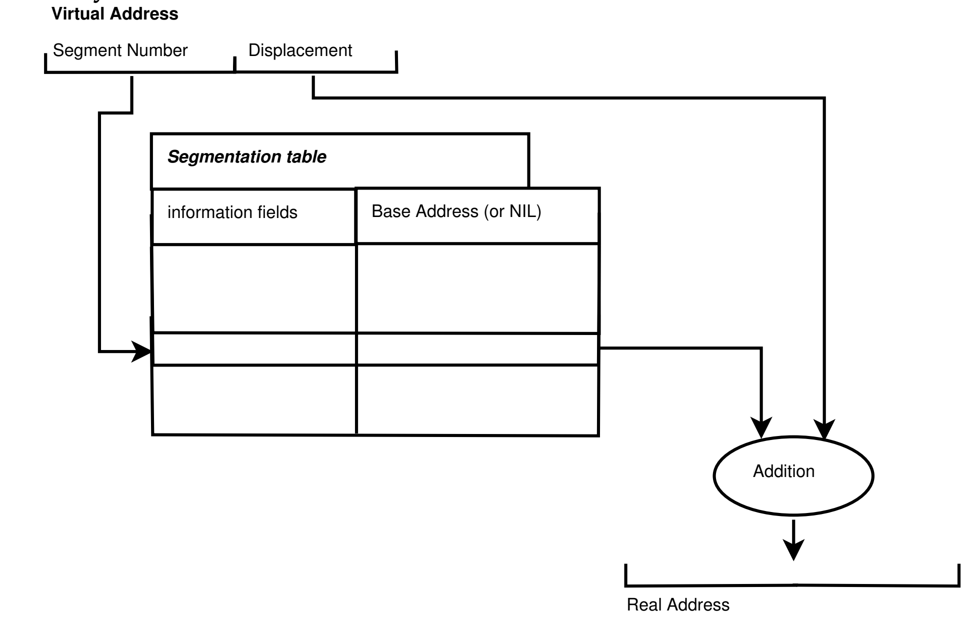
Kernel address space

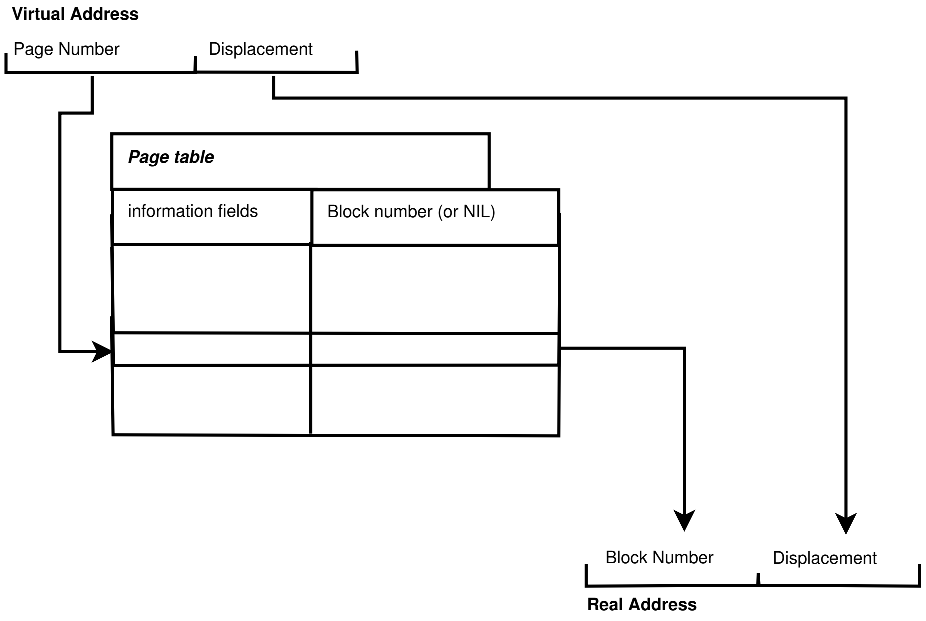
- linear view of the RAM no segments
- no check of illegal accesses

User address space

- virtual dont represent real memory
- private for each process
- in advance allocation real allocation when really in use
- no size limit except hardware ones sometimes software if it doesnt fit the hardware (ex: DOS)

Memory Translation Virtual-Real





Process

Elementary Execution Unit of the Operating System
Two kinds

- **process** based on a copy of resources coming from father process improved with the use of copy on write mechanisms or vfork
- **thread** share the maximum part of the resources coming from the main process ⇒ context switch accelerated

Scheduling

Pre-emptive Multi-task :
Kernel is pre-emptive itself (from 2.6 version);
UNIX Policy to share CPU time between each processes;
Real-Time API in POSIX.1b ;
Linux is nevertheless a real-time kernel :

- Execution time not **predictive**;
- non-determinist system. Priorities management (41 normal levels + 99 real-time levels).

Interruption management

Linked to a device (ex : bus, interface, timer...) ;
Material interruptions can pre-empt a process even if it work in kernel mode;
Two phases computing :

- top-half : high-speed part and critical (ex : acknowledge an interruption for a device), and
- bottom-half : slow part in a queue of tasks that is emptied by the scheduler (ex : processing associated to the interruption). Possibility to share IRQ.

Mutual Exclusion

Linux Kernel is re-entrant and pre-emptive;
There is consequently many possible concurrencies :

- between each CPU (SMP architecture) ;
- between Interruption Controller and System Calls ;
- between different System Calls (because of explicit or implicit pre-emption) ;

⇒ Blocking mechanism (locking) for protection :

- global variables,
- non-re-entrant functions,
- critical hardware accesses.

File Systems

Fundamental of every UNIX based systems ;
Two main kind :

- real : Associated to a real storage device (ex. hdd, CDROM ...) ;
- virtual : emulates a storage device to have access to advantages of files mechanisms (ex : drivers, /proc ...).

File System

Abstraction layer to implement writing mechanism with file system (VFS) :

- can be based on every bloc device ;
- basic mechanism are present : * open()/close(), read()/write(), lseek()... ; *inheriting generic mechanisms if not implemented.
- need of specific tools (ex : formatting, check, tuning...)

Network

One of the big asset of Linux system ; Many devices and protocols are supported ; Divided in two set of sub-systems :

- driver of interfaces that fit with physical level (1) and Data level (2) of OSI Model (ex : Ethernet, Token Ring, PPP...) ;
- Protocols corresponding to layers Net (3) and Transport (4) (ex : TCP, IP, IPX, Appletalk...).

Network

Interface drivers use device drivers to have access to them ; Communication between device drivers, interface drivers and protocols is done with packets (ex : structure sk_buff defined in <linux/skbuff.h>) ;
IP layer integrate route tables and name resolution.

Modules

Dynamic Kernel Modules : object files (**.o/.ko**) implementing a functionality of the kernel ;
Used to reduce the size of the Kernel and bring it more flexibility ;
We can load and unload in dynamic way (during run-time) a new functionality (ex : driver, protocol, file system ...)
Module Dependencies management with **modprobe** tool ;
Management of dynamic link edition with **insmod**.

Device drivers

Piece of software plugged in the kernel to bring an interface between kernel and hardware devices;

There must be two different interfaces :

- one for users that is generic and stable ;
- one to communicate with the kernel .

With Linux, user interface is provided as a special file ;

So we are able to use classic system call like **open()**, **read()**, **write()** ;

Can be implemented as a module dynamically linked "on demand" or statically compiled in the kernel.

4.3.3 Developing environment

Distribution

Sources archives in files linux-X.Y.Z.tar.bz2

- X = VERSION (~ 4 5 years) : number of version ;
- Y = PATCHLEVEL (~ 1 2 years) : number of sub-version (even ⇒ stable, odd ⇒ unstable) ;
- Z = SUBLEVEL (~ 1 6 months) : number of sub-sub-version.

Update patches in files like patch-X.Y.Z+1.bz2 ;[6mm] Files .sign to check signature of archives and patches Ex for signature :

```
gpg -keyserver wwwkeys.pgp.net -recv-keys 0x517D0F0E
gpg -verify linux-X.Y.Z.tar.bz2.sign linux-X.Y.Z.tar.bz2
```

TreeView

Documentation/ : documentation related to the kernel and its sub-systems (ex : sound/, DocBook/...) ;

arch/ : specific part for architectures (ex : alpha/, arm/, i386/...) ;

drivers/ : device drivers (ex : usb/, net/...) ;

fs/ : file systems (ex : xt2/, fat/...) et VFS ;

include/ : headers files (ex : asm-i386/, linux/, net/...) ;

init/ : initialisation process (boot) of the kernel ;

TreeView

ipc/ : inter-processes communication mechanisms (ex : shared memories, semaphores...) ;

kernel/ : core of the system (ex : scheduler, signals...) ;

lib/ : mini C library to be used in the kernel (ex : strcmp(), sprintf()...) ;

mm/ : memory management ;

net/ : interfaces and network protocols (ex : ethernet/, ipv4/...) ;

scripts/ : useful scripts for configuration and compilation of the kernel.

TreeView
In 2.6 version, new directories : crypto : generic API cryptography and their implementations ; security : security mechanisms running in the kernel et implementations (SELinux) ; sound : sound sub-systems (in drivers before).

4.4 Tools

4.4.1 Dev. tools

Compilation
Only supported tools for compilation :

- GCC (with its specific optimization set...) ;
- GNU Make (syntax...).

Lower versions of tools ⇒ Documentation/Changes ;
Use optimization flag -O of GCC to have inline in headers interpreted.

Ex Makefile
Ex : Makefile

```
CC = gcc
CCOPTS = -Wall -Wstrict-prototypes -O2
KSRC = /lib/modules/\$(shell uname -r)/build
MODVERSIONS = -DMODVERSIONS
MODVERSIONS += -include $(KSRC)/include/linux/modversions.h
CFLAGS = -I$(KSRC)/ ... $(CCOPTS) -DMODULE $(MODVERSIONS)
CFLAGS += -D__KERNEL__

%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@
```

C Language and C library
No libC for kernel programming ;
Useful Functions in the directory lib/ of kernel sources ;
Coding standards described in Documentation/CodingStyle ;
It is advised to split big C files into many smaller files with well defined functions inside (ex : software layer) ;
Object files (.o) can then be merged in one module (one **module_init()**...) with the **linker** ;
Ex :

```
>ld -r part1.o part2.o -o module.o
```

Patches
Collecting modifications in one or more files ;
Must keep the original file or directories ;
Ex : creation of a patch

```
>diff -urN <orig> <dest> > patch_<description>
```

Ex : using the patch

```
>cd <origine>
>patch -p1 < patch_<description>
```

4.4.2 Debugging

Console
Kernel function **printk()** to send information in standard output (console) ;
Multiple levels of debugging are defined in <linux/kernel.h> (ex : KERN_EMERG, KERN_INFO...) ;
Messages are logged by syslogd deamon (entry kern.*) ;
dmesg software display and check the round buffer used by **printk()** ;
Prototype :

- int printk (const char *fmt, ...) ;

KGDB
kgdb patch for the kernel (http ://kgdb.sourceforge.net) ;
Need of a second machine :

- link the target with a serial connection ;
- use GDB-client to debug the target.

Easy to use and quick way to debug (source mode debugging).

KDB
Embedded in the kernel debugging tool ;
Developed by SGI (http ://oss.sgi.com/projects/kdb/) ;
It includes :

- A set of user commands that allow to debug the kernel and real-time kernels ;
- a kernel debugger in assembly mode (not in source mode)

Useful debugger but hard to use because of its complexity.

Profiling
It is possible to analyse time spent by each function of the kernel :

- boot parameter : *profile=n* ;
- binary information in */proc/profile* ;
- can become readable with **readprofile** tool.

Allow to identify :

- structural misbehaviour ;
- to be optimised parts of the driver.

2.6 kernel provides Oprofile , a more powerful kernel/user profiler.

Other possibilities
On request Debugging :

- virtual file in */proc* ;
- system call **ioctl()**.

GDB debugging read only mode the virtual file */proc/kcore* ;
Using Linux User Mode for hardware independent parts ;
Analysing outputs of the software with **strace** ;
In oops case, analysing et interpretation with **ksymoops** tool (use the file System.map).

4.4.3 Practical

Practical

- Get sources of a linux kernel on www.kernel.org
  - Extract kernel sources : tar xvfj linux-X.Y.Z.tar.bz2
  - Go to the root of the sources : cd linux-X.Y.Z
- Apply some patches : patch -p1 < /path/to/patch-foo
- Configure the kernel : make xconfig
  - Compile (2.4 kernel) : make dep clean bzImage modules
  - Compile (2.6 kernel) : make
- Install : make modules_install install
- Reboot.

4.5 User Space Mechanisms

4.5.1 Review

- Threads
- Signals
- Systems Calls
- Hardware access

4.5.2 System Calls

ioctl

ioctl : Devices Control

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

int ioctl(int d, int request, ...)
```

Example :

0x00000606	LPGETIRQ	int *
0x00000608	LPWAIT	int
0x00000609	LPCAREFUL	int
0x0000060A	LPABORTOPEN	int

syscall

```
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
int main(void) {
long ID1, ID2;
/*-----------------------------*/
/* direct system call*/
/* SYS_getpid (func no. is 20) */
/*-----------------------------*/
ID1 = syscall(SYS_getpid);
printf ("syscall(SYS_getpid)=%ld\n", ID1);
/*-----------------------------*/
/* "libc" wrapped system call */
/* SYS_getpid (Func No. is 20) */
/*-----------------------------*/
ID2 = getpid();
printf ("getpid()=%ld\n", ID2);
return(0);
}
```

Extract from "/usr/include/bits/syscall.h"

```
...
#define SYS_chmod __NR_chmod
#define SYS_chown __NR_chown
#define SYS_chroot __NR_chroot
#define SYS_clock_getres __NR_clock_getres
#define SYS_clock_gettime __NR_clock_gettime
...
```

4.5.3 Hardware access

Input/Output Ports and Authorization

Address range used by present devices on IO bus (ex : controlling registers);

We can access these ports from the kernel with functions :

- in{b,w,l}()/out{b,w,l}() : read/write 1, 2 or 4 octets on an IO port,
- in{b,w,l}_p()/out{b,w,l}_p() : read/write 1, 2 or 4 octets on an IO port and pause until the I/O completes,
- ins{b,w,l}()/outs{b,w,l}() : read/write sequences of 1, 2 or 4 octets on IO ports.

To be authorize to access data in user space you have to ask for permission with the functions :

- int ioperm(unsigned long from, unsigned long num, int turn_on) : only the first 0x3ff I/O ports can be specified in this manner.
- int iopl(int level) : changes the I/O privilege level of the calling process, Warning !! not advised.

Prototypes

Prototypes x86 (<asm/io.h>) :

- unsigned char inb (unsigned short port) ;
- void outb (unsigned char byte, unsigned short port) ;
- unsigned char insb (unsigned short port, void *addr, unsigned long count) ;
- void outsb (unsigned short port, void *addr, unsigned long count) ;

Mapping addresses

Memory

- mmap
- mremap
- mprotect

IO In some cases, it is mandatory to map addresses range of IO in linear address space of the kernel (ex : PCI ⇒ addresses > PAGE_OFFSET) :

- ioremap() : map physical addresses range to a linear address range (similar to vmalloc()),
- iounmap() : free an address range already mapped .

Memory Mapping

```
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[]){
    char *addr;
    int fd;
    struct stat sb;
    off_t offset, pa_offset;
    size_t length;
    ssize_t s;
    fd = open(argv[1], O_RDONLY);
    if (fstat(fd, &sb) == -1)     /* To obtain file size */
        offset = atoi(argv[2]);
    pa_offset = offset & ~(sysconf(_SC_PAGE_SIZE) - 1);
    /* offset for mmap() must be page aligned */
```

Memory Mapping

```
    if (argc == 4) {
      length = atoi(argv[3]);
      if (offset + length > sb.st_size)
         length = sb.st_size - offset;
       /* Can't display bytes past end of file */
    } else {
      /* No length arg ==> display to end of file */
      length = sb.st_size - offset;
    }
    addr = mmap(NULL, length + offset - pa_offset,
       PROT_READ, MAP_PRIVATE, fd, pa_offset);
    s = write(STDOUT_FILENO,
                addr + offset - pa_offset, length);
    exit(EXIT_SUCCESS);
}
```

Kernel Services

/sys : system / driver status information

```
> cat /sys/bus/usb/devices/usb4/4-2/product
USB-PS/2 Optical Mouse
```

/proc : Hardware Status Information

```
> cat /proc/iomem
...
fec00000-fec00fff : IOAPIC 0
...
fee00000-fee00fff : Local APIC
...
>cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-0060 : keyboard
...
```

4.6 Kernel/Hardware Interface

4.6.1 Devices

IO Port Reservation

Drivers can access IO ports that are not there own devices (ex : probing) ;
In order to avoid conflicts, kernel can reserve ports for drivers :

- int request_region() : reserve a IO port ;
- check_region() : check if the port is not yet reserved (deprecated in 2.6) ;
- release_region() : free an IO port.

List of ports already reserved appears in /proc/ioports.

Memory Reservation

To avoid conflicts, kernel can reserve IO memory region for some device driver :

- request_mem_region() : reserve one memory range of adress,
- check_mem_region() : check if the region is already reserved (deprecated in 2.6),
- release_mem_region() : free this region.

Listing of memory regions reserved appears in /proc/iomem ;

Prototypes
Prototypes (<asm/io.h> and <linux/ioport.h>) :

- void * ioremap (unsigned long offset, unsigned long size) ;
- void iounmap (void *addr) ;
- struct resource * request_{mem_}region (unsigned long start, unsigned long n, const char *name) ;
- int check_{mem_}region (unsigned long start, unsigned long n) ;
- void release_{mem_}region (unsigned long start, unsigned long n) ;

Hardware access functions
It is then possible to access to shared memory of IO with the following functions :

- readb,w,l()/writeb,w,l() : read/write respectively 1, 2 ou 4 octets in IO memory ;
- memcpy_{from,to}io() : read/write octet continuous block in IO memory ;
- memset_io() : fill a specific range of data with a fixed value ;
- virt_to_bus()/bus_to_virt() : translate between a virtual linear address and real address on the bus .

Prototypes
x86 prototypes (<asm/io.h>) :

- char readb (void *addr) ;
- void writeb (char byte, void *addr) ;
- void * memcpy_{from,to}io (void *dest, const void *src, size_t count) ;
- void * memset_io (void *addr, int pattern, size_t count) ;
- unsigned long virt_to_phys (volatile void *addr) ;
- void * phys_to_virt (unsigned long addr) ;

Kernel memory
Many methods exists in kernel to allocate memory :

- kmalloc()/kfree() : allocate/de-allocate a real continuous memory block in the kernel (GFP_KERNEL $\Rightarrow$ can sleep, GFP_ATOMIC $\Rightarrow$ atomic),
- __get_free_pages()/free_pages() : allocate/de-allocate an integer number (2^n) of continuous real memory pages (idem previous but for biggest needs - limit : 128 ko),
- vmalloc()/vfree() : allocate/de-allocate a virtual memory block composed by many discontinuous memory blocks of real memory.

Prototypes
Prototypes (<linux/slab.h> et <linux/vmalloc.h>) :

- void * kmalloc (size_t size, int flags) ;
- void kfree (const void *addr) ;
- unsigned long __get_free_pages (int gfp_mask, unsigned long order) ;
- void free_pages (unsigned long addr, unsigned long order) ;
- void * vmalloc (unsigned long size) ;
- void vfree (void *addr) ;

4.6.2 Interruptions and events

Interruptions and events
Supervision of Input/Output events, I/O often asynchronous and unpredictable (ex : keyboard) ;
Two methods to supervise I/O :

- active waiting (polling) ;
- interruptions.

In polling mode, CPU is released with schedule() function after each unsuccessful test ;

Interruptions et events
Ex : polling

```
for ( ; ;)
{
  if (read_state(carte) & ETAT_END)
     break ;
  schedule() ;
}
```

In interruptible mode, calling process is asleep and put in a waiting queue ;

Interruption manager will have to wake the process (or processes according to its policy)

Interruptions et events

Interruption manager is called interrupt handler.

Properties :

- must be fast ;
- can't call sleeping function (ex : kmalloc() non atomic).

For these reasons, interruption management is divided in 2 parts :

- interrupt handler : fast and uninterrupted part, and
- bottom-half : slowest part, manage waiting queue of tasks.

First part can exist without second part but inverse is not possible ;

Interruptions et events

Interrupt handler is declared with function request_irq() ;

It is freed by free_irq() ;

Prototypes (<linux/sched.h>, <linux/interrupt.h>) :

- int request_irq (unsigned int irq, irqreturn_t (*handler)(int, void *, struct pt_regs *), unsigned long flags /* SA_SHIRQ */ , const char *device, void *dev_id) ;
- void free_irq (unsigned int irq, void *dev_id) ;

Interruptions et events

When irq arises, function handler() is called ;

dev_id field is used to identify devices when IRQ is shared (SA_SHIRQ) ;

It can be used to send specific structure to driver ; List of already declared IRQ is available in /proc/interrupts ; Returning code of request_irq must be IRQ_NONE or IRQ_HANDLED.

Interruptions et events

bottom-halves

Kernel functions used to manage asynchronous tasks ;

Executed after every return of system call, exception or interrupt handler ;

Two possible "implementations" :

- tasklets : can run on different CPU but one instance at one time
- softirqs : can run on different CPU and many instances can work in parallel

Interruptions et events

Ex :

```
void my_routine_bh(unsigned long)
{
/* ... bottom-half code  ... */
}

DECLARE_TASKLET(ma_tasklet, ma_routine_bh, 0) ;

void my_handler_irq(int irq, void *dev_id,
                                  struct pt_regs *regs)
{
 ...
 tasklet_schedule(& ma_tasklet) ;
 ...
}
```

4.6.3 Waiting queues

Concepts

A process can wait for a kernel event (ex : data) ;[6mm] Process is going in a waiting queue and release the CPU ;[6mm] When the event arises, processes in the corresponding queue are awaking.

Implementation

Manipulation methods on waiting queue are as follow :

- wait_event_interruptibletimeout() : put the running process in sleep mode, in interruptible way or not and with/without time out limit, sleeping process is put in waiting queue

Interruptible Implementation

'interruptible' version check if a signal has been sent to a process and return an error code in this case (-ERESTARTSYS).

This error code should be propagated as a return of the system call. Initialisation of a waiting queue (type wait_queue_head_t) :

- during declaration ⇒ DECLARE_WAIT_QUEUE_HEAD(), or
- during execution (runtime) ⇒ init_waitqueue_head().

Prototypes (<linux/wait.h>) :

- DECLARE_WAIT_QUEUE_HEAD (name) ;
- void init_waitqueue_head (wait_queue_head_t *wq) ;

Prototypes
Prototypes (<linux/sched.h>, <linux/wait.h>) :

- void wait_event (wait_queue_head_t *wq, condition) ;
- int wait_event_interruptible (wait_queue_head_t *wq, condition) ;
- int wait_event_interruptible_timeout (wait_queue_head_t *wq, condition, long timeout) :
- void wake_up_interruptible (wait_queue_head_t *wq) :
- void wake_up_interruptible_nr (wait_queue_head_t *wq, int nr) :
- void wake_up_interruptible_all (wait_queue_head_t *wq) ;

Example
Ex :

```
static DECLARE_WAIT_QUEUE_HEAD(ma_file) ;

int function_reading (void *addresse) {
  ...
  ret = wait_event_interruptible(& ma_file, readb() != 0) ;
  if (ret $<$ 0)
    return ret ;
  ...
  }
void my_handler(int irq, void *priv, struct pt_regs *regs)
{
  ...
  wake_up_interruptible(& my_file) ;
  ...
}
```

4.7 Driver (Module)

4.7.1 Manipulation

Manipulation
Set of tools to manipulate modules ⇒ modutils (kernel 2.4) or module-init-tools (kernel 2.6) :

- lsmod : listing of modules loaded in the kernel ;
- ins,rmmod : manual insertion/extraction of a module in the kernel ;
- modprobe : automatic insertion/extraction of a module in the kernel with dependency resolving (loading all needed modules) ;

Manipulation

- modinfo : general information about a module (ex : author, description, parameters...) ;
- depmod : generate a list of dependencies for a set of modules ;

Modules are automatically loaded when it is need according to hotplug mechanism that run modprobe in user mode.

4.7.2 Basic Routines

Basic Macros
Descriptions and parameters
managed by dedicated macros (<linux/module.h>) :

- MODULE_AUTHOR() ;
- MODULE_DESCRIPTION() ;
- MODULE_LICENSE() ;
- MODULE_PARM() ;
- MODULE_PARM_DESC().

In kernel 2.6, MODULE_PARM() is replaced by (<linux/moduleparam.h>) :

- module_param(name, type, perm) ;

Ex : MODULE_PARM(my_integer, "i") ; MODULE_PARM_DESC(my_integer, "My integer parameter") ;

Initialisation and release
Macros defined in file <linux/init.h> :

- module_init() : declaration of initialisation function (type int foo(void) ;) ;
- module_exit() : declaration of quitting function (type void foo(void) ;).

MODULE is a constant that allow the compiler to make the difference between module versus in kernel compilation (no use for you!).

Implementation
The initialisation function must contain all necessary information and computation to initialise a module (ex : allocations, hardware initialisations, resources reservation...) ;
In the opposite, exiting function must undo all the things done by initialisation one (ex : freeing reserved memory, hardware deactivation, freeing resources...) ;

routines

Linux uses optimisations of GCC to free in memory some unused part of the code after they have been instantiated (<linux/init.h>) :

- __init : initialisation functions (freed after finishing module_init()) ;
- __exit : exit functions (ignored if statically linked to the kernel) ;
- __initdata : used data in initialisation functions (freed after finishing module_init()) ;
- __exitdata : used data in exit functions (ignored if statically linked to the kernel) ;

Ex :

```
int __init my_function_init(void)
{
/* ...code... */
}

module_init(my_function_init) ;
```

Usage counter

Each module is associated to a usage counter ;
Avoid that a module is unloaded while being used ;
Counter modification is done when the module is called ;
The module can be manipulated with functions described in (<linux/module.h>) :

- MOD_INC_USE_COUNT : increase the usage counter ;
- MOD_DEC_USE_COUNT : decrease the usage counter ;
- MOD_IN_USE : bring the value of the usage counter.

Kernel 2.6 provides instead (<linux/module.h>) :

- try_module_get(THIS_MODULE) : try to increase the usage counter.
- module_put(THIS_MODULE) : decrease the usage counter.
- module_refcount(THIS_MODULE) : bring the value of the usage counter.

Symbols

Exported symbols by the kernel are available in /proc/ksyms (/proc/kallsyms in 2.6) ;
Used by insmod to do dynamical linking ;
Most of the symbols are exported in kernel/ksyms.c ;
Modules can declare new symbols according to macro EXPORT_SYMBOL() (<linux/module.h>) ;
ksyms tool give a more precise and readable view of /proc/ksyms.

Integration to kernel sources

Choose an appropriate directory in kernel tree ;

Create a sub-tree if necessary (ex : many elements) ;

In 2.4, edit Config.in and Makefile files of upper directory ;

En 2.6, edit Kconfig et Makefile files of upper directory ;

Add an entry inspired by the other ;

Add a help entry in Documentation/Configure.help (2.6 only, in 2.6 content is in Kconfig)

Syntax is defined in Documentation/kbuild/... ;

Implementation basic routines

In kernel 2.4, while a statical compilation, parameters of a module must be defined by macro __setup() (in <linux/init.h>) :

void __setup (char *append, int (*extract) (char *)) ;

In kernel 2.6, module_param() automatically do this, parameters are directly given as inline parameter with :[6mm] module.param=valeur

Practical

Creation of the simplest module (HelloWorld) ;

Add some parameters to this module ;

Link it to the kernel in a static point of view.

4.7.3 Char Devices

File systems

You can access to drivers with the use of specials files (device nodes also called inode) ;

A structure file_operations provide a way to define any entry point (<linux/fs.h>) ;

Access classic functions can be overloaded by driver own access functions;

Structure Associated

Initialisation of structure variables is done with the new C syntax (.var = value) (in old times a specificity of GCC)

Some not implemented methods are replaced by default functions (ex : open(), close())

Others not implemented functions return -EINVAL ;

Ex :

```
static struct file_operations mondriver_fops = {
    .owner = THIS_MODULE,
    .read = mondriver_read,
    .write = mondriver_write,
    .open = mondriver_open,
    .release = mondriver_release};
```

Majors et minors

Special files are created with command mknod ;

A *inod* file is described by a unique identifier (i_rdev, of type kdev_t, size 16 bits in 2.4, 32 bits in 2.6) separated in two numbers (8+8 / 12+20) :

- major : identifier of the driver kind (ex : IDE, /dev/hd* ⇒ Major=3),
- minor : identifier of one driver in a set (ex : IDE, /dev/hda1 ⇒ minor=1).

Majors 0 to 255 are reserved ;

Majors and minors

List of majors and minors already used is updated in Documentation/devices.txt and <linux/major.h> ;

Macros MAJOR() and MINOR() allow you to know what are these information of your inode ;

Ex :

```
static int mondriver_open(struct inode *inode,
                                        struct file *file)
{
unsigned int minor = MINOR(inode-$>$i\_rdev) ;
/* or file-$>$f\_dentry-$>$d\_inode-$>$i\_rdev */
}
```

Kernel 2.6 have also functions : imajor(inode) and iminor(inode)

Majors and minors

Classical use, a major describes one driver for a specific device ;

Minors are used to identify different devices of the same kind or controlled sub-set ;

Can manage same maje identical cards with this habit:

- structures that describes each device (ex : buffers, mutex...) ;
- array of structures indexed with minors ;
- **n** files in /dev with the same major and different minors.

Returning Value

Negative means error defined in <linux/errno.h> and <asm/errno.h> ;

Positive or zero means working ;

This standard is used in every functions in the kernel; Error values transmitted between each functions ;

Returning Value

Ex : invalid argument

```
if (arg $>$ 3) return -EINVAL ;
```

Ex : error transmission

```
int ret ;

ret = fonction\_noyau() ;

if (ret $<$ 0) return ret ;
```

4.7.4 Memory Management

Data Transfer kernel/user

Kernel uses physical or virtual memory ;

Processes use their own virtual memory space ;

Need to transfer data from/to kernel memory space to/from calling process memory space ;

Main functions are :

- get,put_user() : transfer a variable from/to user memory space (using with lvalue), and
- copy_{from,to}_user() : transfer a buffer from/to user memory space.

Data Transfer kernel/user

These functions are all calling function access_ok() that checks validity of a user buffer ;

This call can be avoided if we used those functions prefixed by __ (not advised). Prototypes (<asm/uaccess.h>) :

- int get_user (lvalue, addr) ;
- int put_user (expression, addr) ;
- void copy_{to,from}_user (unsigned long dest, unsigned long src, unsigned long len) ;
- int access_ok (int type, unsigned long addr, unsigned long size) ;

4.7.5 Xenomai a RTOS example

4.7.6 Description

Motivations

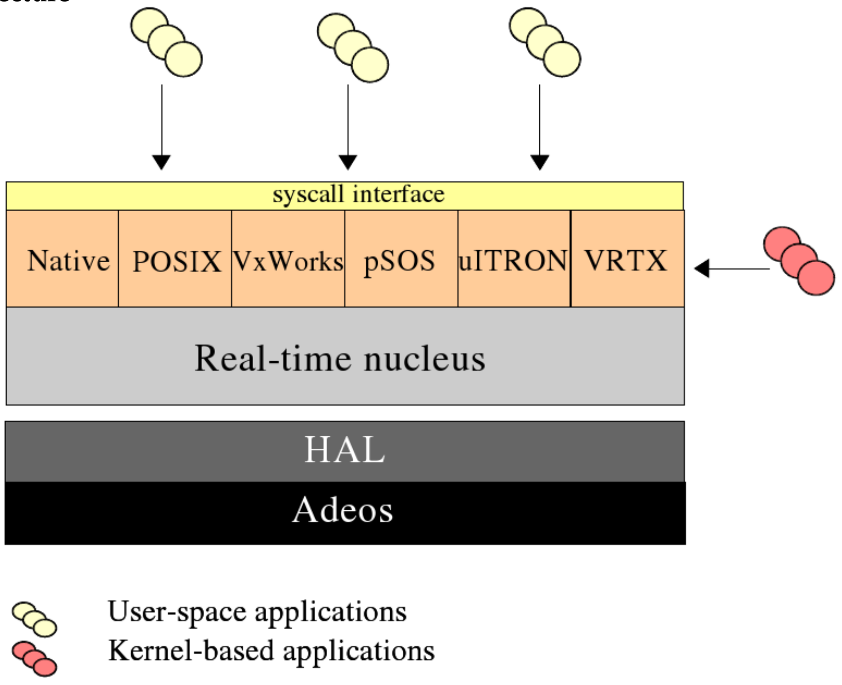
- Linux based (GPL)
- Developing tools (Multi-Interfaces)

- Dynamic community (Forum and mailing list[1])

Main Structure

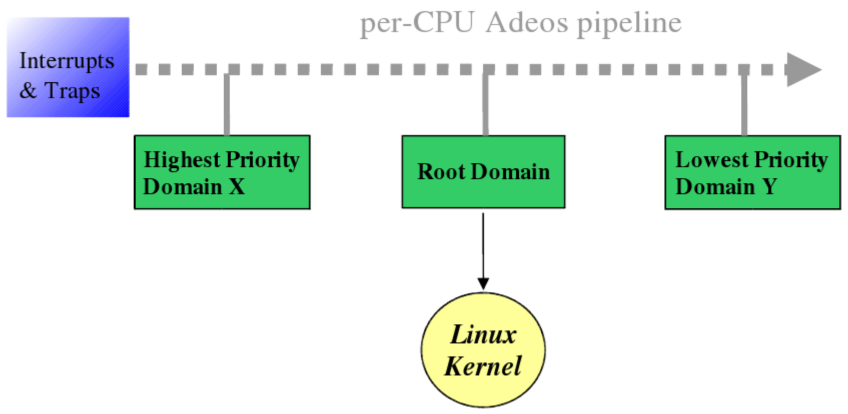
- Patched Kernel (ADEOS)
- HAL
- nucleus
- Multi-skins
  - RTAI,
  - VXWorks,
  - POSIX,
  - RTDM,
  - pSOS+,
  - VRTX,
  - uITRON ...

Architecture

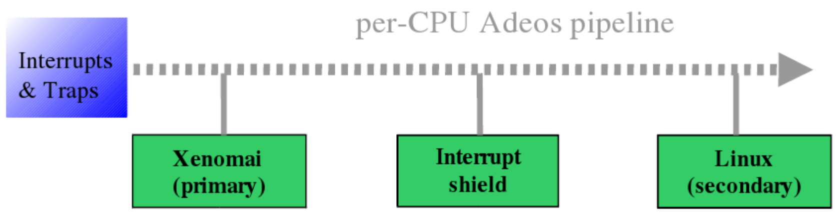


[1]Quality to be observed when we want to use a free tool in long term point of view

ADEOS - Virtual Domains



ADEOS - Applied to Xenomai



ADEOS - HAL to manage communications

```
void realtime_eth_handler (unsigned irq, void *cookie)
{
rthal_irq_enable(irq);
rthal_irq_host_pend(irq);
/* This irq has been marked as pending for Linux */
}
void linux_eth_handler (int irq, void *dev_id,
                          struct pt_regs *regs)
{
/* process the IRQ normally. */
}
```

NATIVE API - Main

- Compactness : <100 services
- Orthogonality :
  - developer memory :)

  - Non-ambiguity
  - More functions combination than more complex functions

- Context independent :
  - Kernel Space
  - **User space** : privileged

- Small Semantic for localisation : Same names
  - Kernel (ex. xeno_native.ko),
  - User (ex. libnative.so)

NATIVE API - Services

- Tasks Management
- Time Service
- Synchronisation tools
- Messages and communication
- Control I/O
- Registers

NATIVE API - Services

- Tasks Management
  - Increasing Priorities 1-99
- Time Service
  - Timers,
  - Time,
  - **Alarm**...
- Synchronisation tools
  - Counting Semaphores,
  - Mutex,
  - Conditional Variables,
  - Event Flags

NATIVE API - Services

- Message and communication
  - Inter-tasks synchronous Messages
  - Messages queue
  - Pipes : Message streaming
- I/O control
  - Interruptions
  - Heap memory
  - Memory Access of a device in user space
- Registers
  - Small semantic, developer basis
  - /proc/xenomai/registry/...

NATIVE API - Compilation Flags

```
CFLAGS=$(shell xeno-config --xeno-cflags)
LDFLAGS= -lnative $(shell xeno-config --xeno-ldflags)

myapp: myapp.c
    $(CC) -o $@ myapp.c $(CFLAGS) $(LDFLAGS)
```

4.7.7 Main Mechanisms

Common Functions

- Create / Delete
- Bind / Unbind ⇒ Registry and binding
- Inquire ⇒ Inquiries

Registry and binding
Multiple processes unified sharing of objects.
In Kernel Space **and** in User Space (T_PRIMARY).

- "/proc/xenomai/registry"
- int **rt_object_bind** (RT_OBJECT *obj, const char *name, RTIME timeout)
- static int **rt_object_unbind** (RT_OBJECT *obj)

Ex :

```
main1.c:
RT_TASK t;
rt_task_create(&t,"myTask",0,0,0);

main2.c:
RT_TASK *tt;
rt_task_bind(tt, "myTask", (RTIME)10000);
... Execution ...
rt_task_unbind(tt);
```

Inquiries
State informations of any object.

- int **rt_object_inquire** (RT_OBJECT *obj, RT_OBJECT_INFO *info)

Ex:

```
...
RT_TASK *curtask;
RT_TASK_INFO curtaskinfo;
curtask=rt_task_self();
rt_task_inquire(curtask,&curtaskinfo);
rt_printf("Task name : %s \n", curtaskinfo.name);
...
```

RTDK
Introduces a collection of utilities aimed at forming a Real-Time Development Kit for userland usage.

```
int rt_vfprintf(FILE *stream, const char *format, va_list args);
int rt_vprintf(const char *format, va_list args);
int rt_fprintf(FILE *stream, const char *format, ...);
int rt_printf(const char *format, ...);

int rt_print_init(size_t buffer_size, const char *name);
void rt_print_cleanup(void);
void rt_print_auto_init(int enable);
const char *rt_print_buffer_name(void);
```

4.7.8 RTDM

Real Time Driver Model RTDM

- User API
- Driver Development API
  - Inter-Driver API
  - Device Registration Services + Synchronisation Services

  - Clock Services
  - Task Services
  - Timer Services
  - Synchronisation Services
  - Interrupt Management Services
  - Non-Real-Time Signalling Services
  - Utility Services

- User API
- Driver Development API
  - Inter-Driver API
  - Device Registration Services + Synchronisation Services
  - Clock Services
  - Task Services
  - Timer Services
  - Synchronisation Services
  - Interrupt Management Services
  - Non-Real-Time Signalling Services
  - Utility Services

Real Time Driver Model

- Device Profiles
  - CAN Devices
  - Serial Devices
  - Testing Devices

5 Application

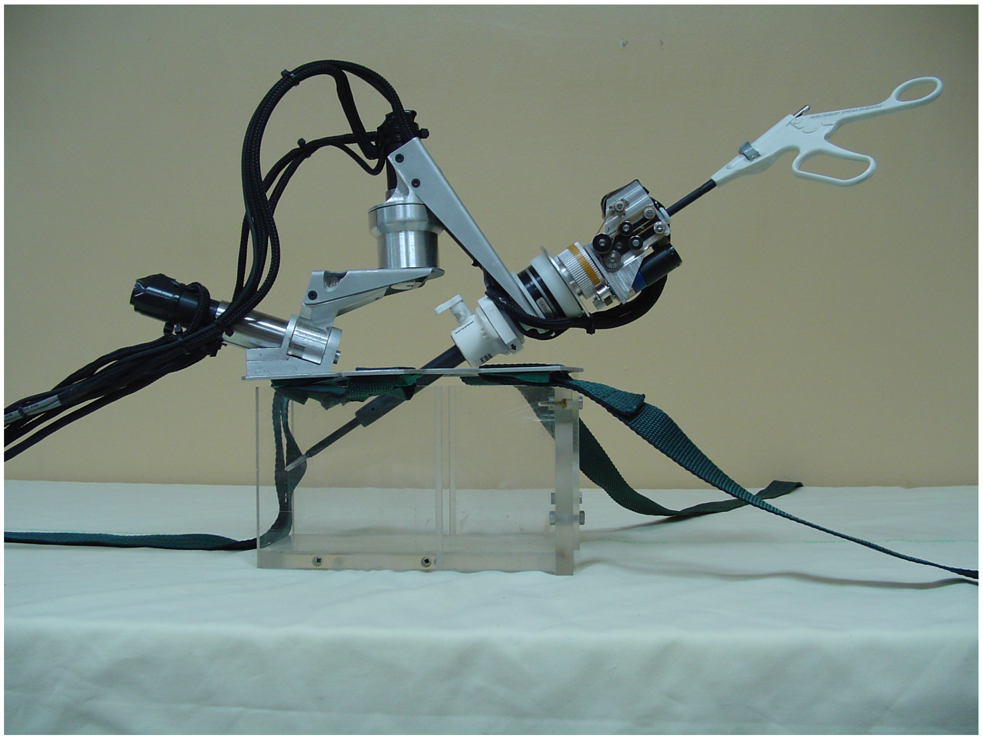
5.1 Examples

5.1.1 Examples

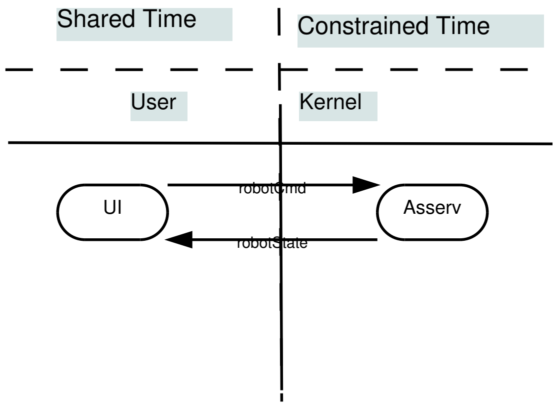
MC2e

- Aim :
  - Surgical Robotics
  - 3rd Hand

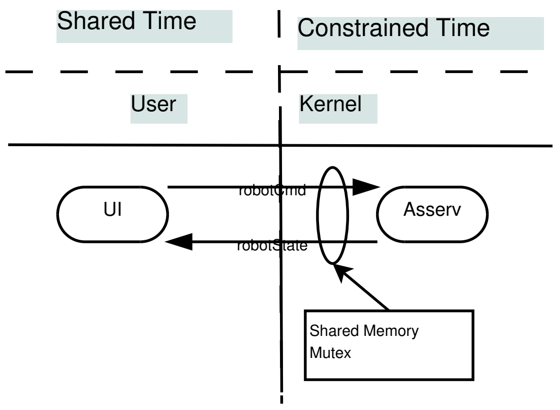
  - Minimal invasive operation
  - Co-manipulation
- Technical notes :
  - Spherical kinematics
  - Force based robot



MC2e : Tasks Architecture



MC2e : Synchro Solution

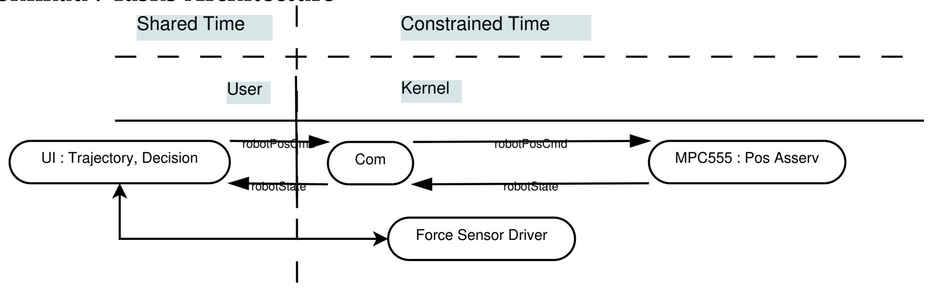


Only one shared memory with all the state and command information. $\Rightarrow$ leads to only one synchronization mechanism

Monimad : Description

- Aim :
  - Sit-To-Stand
  - Walking
  - disbalance recover
  - Old and Diseased people
  - Assistance

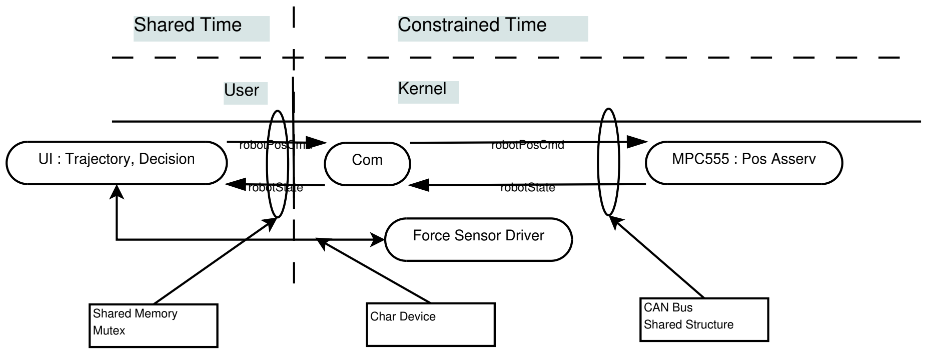
Monimad : Tasks Architecture



Elements :

- CAN Bus
- Force Sensors (Serial, Acquisition card)
- UI

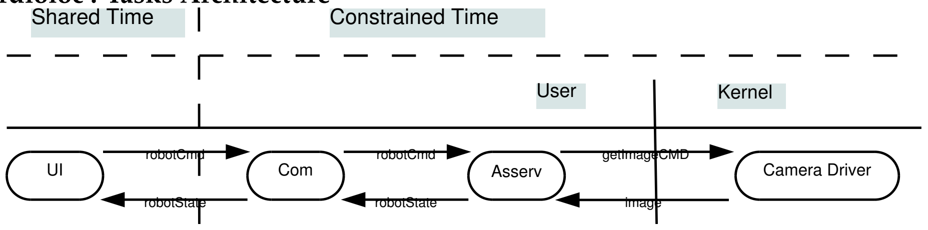
Monimad : Synchro Solutions



Cardioloc : Description

- Aim :
  - Heart Surgery
  - Compensation
  - Visual Servoying
- technical information :
  - 2000Hz
  - PiezoElectric Actuator
  - High frequency camera

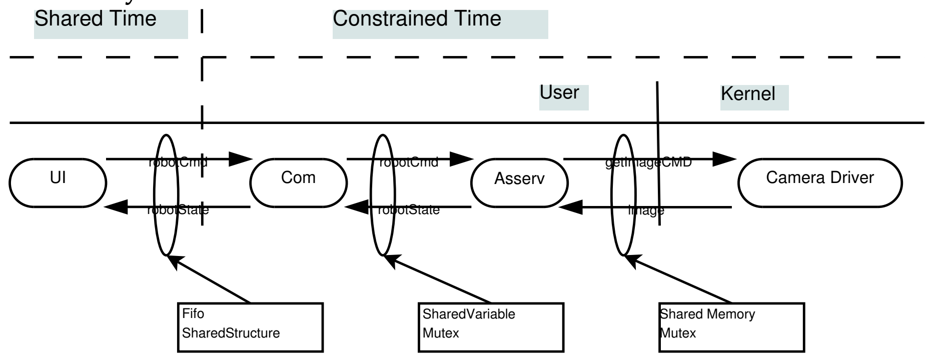
Cardioloc : Tasks Architecture



Elements :

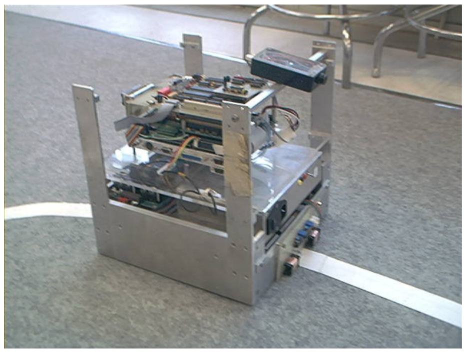
- GUI
- Camera Driver
- Threads

Cardioloc : Synchro Solutions

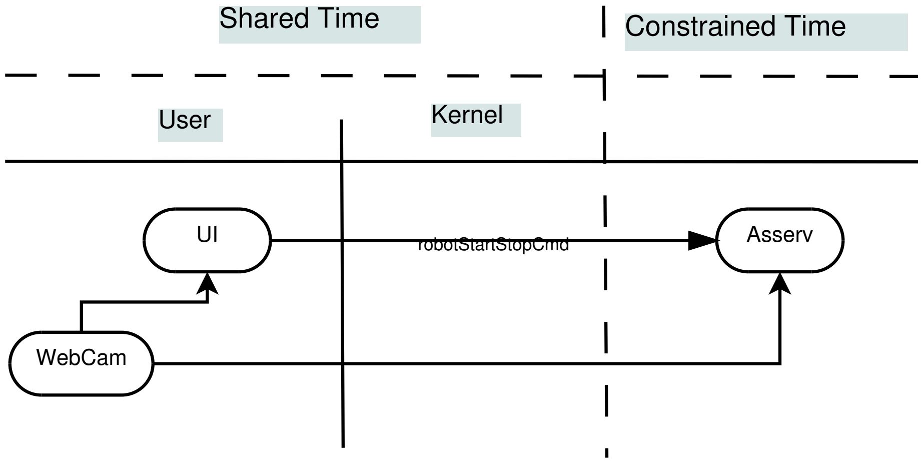


RobModex : Description

- Aim :
  - Pedagogic Solution
  - Polytechnique School of Engineering (Palaiseau)
- Technical
  - Fixed Wheel Structure
  - Usb Camera



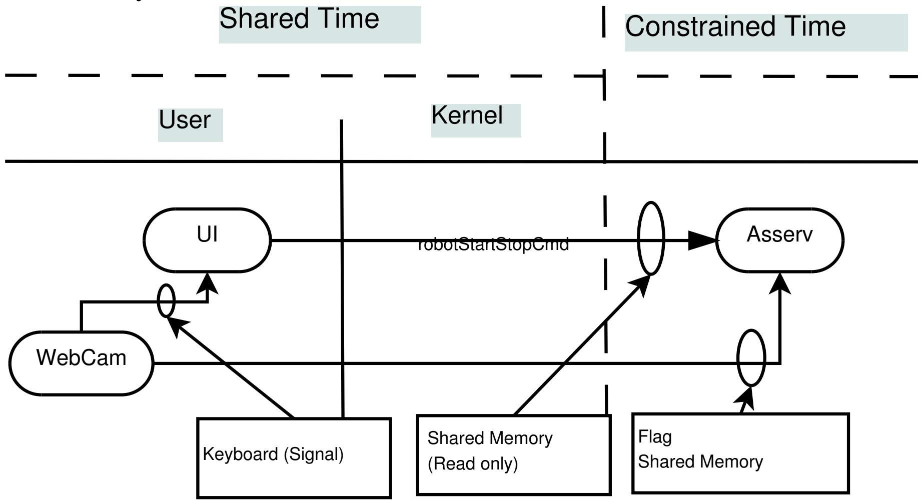
RobModex : Tasks



Elements :

- USB Driver
- Threads

RobModex : Synchro Solution



Notice : **There is no Synchronization needed**

6 Documentations

6.1 Doc

6.1.1 Interface with community

Documentation

The Documentation/ directory in kernel sources ;
The documentation Linux project

- LDP "`http://www.tldp.org`" : guides, HOWTOs, FAQs, manpages... ;

Personal web sites of maintainers (see newsgroups) ;
Other documents from web (`http://www.google.com/linux/`).

Information
Specialised Mailing Lists (newsgroups `http://www.uwsg.indiana.edu/search/`) :

- linux-kernel,
- kernel-newbies,
- linux-net... ;

Week summary of discussions of linux-kernel mailing list :

- Kernel Traffic (`http://kt.zork.net/kernel-traffic/`), and
- Kernel Notes (`http://www.kernelnotes.org`).

Interface with community
FAQ Linux kernel (http ://www.tux.org/lkml/) ;
Discussion Forums : comp.os.development.kernel... ;
Personal email maintainers or authors of a part of source code.

6.1.2 References

References
Use the source, Luke !
The answer of all your questions are inside the kernel sources...

References
Books - Linux Device Drivers 2nd edition (O'Reilly) `http://www.xml.com/ldd/chapter/book/index.html` :

- Understanding the Linux Kernel (O'Reilly) :
- "Programmation Linux 2.0 "(Eyrolles).

References
E-Books - Linux Kernel Internals (kernel 2.4) [`http://www.tldp.org/LDP/lki/`] :

- Linux Kernel Module Programming Guide (kernels 2.0 et 2.2) [`http://www.tldp.org/LDP/lkmpg/`] :
- Linux Kernel Hackers' Guide (kernels 2.0 et 2.2) [`http://www.tldp.org/LDP/khg/`] :
- The Linux Kernel (kernel 2.0) [`http://www.tldp.org/LDP/tlk/`] :
- The Linux Programmer's Guide (API user space) [`http://www.tldp.org/LDP/lpg/`].

References
Articles and documents

- Articles on how import drivers from 2.4 to 2.6 [`http://lwn.net/Articles/driver-porting/`]
- Links of kernelnewbies [`http://www.kernelnewbies.org/links/`] :
- Articles of kerneltrap [`http://kerneltrap.org/`] :
- Conceptual architecture of Linux Kernel [`http://plg.uwaterloo.ca/~itbowman/CS746G/a1/`] :
- Concrete architecture of Linux kernel [`http://plg.uwaterloo.ca/~itbowman/CS746G/a2/`].

References

- `http://www.xenomai.org`
- `http://www.xenomai.org/index.php/Publications`
  - Native-API-Tour-rev-C.pdf
  - Life-with-Adeos-rev-B.pdf
  - RTDM-and-Applications.pdf
- `http://www.xenomai.org/index.php/API_documentation`

6.1.3 Licences

GPL

GNU General Public License (GPL) Maintained et defended by Free Software Foundation (FSF) ;

Give the right (and it is also a duty !) to copy, modify et redistribute the software only under the same licence terms (GPL).

This rule includes also software linked to codes licensed under GPL ;

No free of charge are described, only freedom in using ;

It is possible to sell a software under GPL but not to forbid them to modify it and redistribute (free of charge or not).

LGPL

- GNU Lesser General Public License (LGPL) Maintained and defended by Free Software Foundation (FSF) ;
- Less restricted than GPL ; Possibility to call LGPL code from a non-free program (ex : software under licence incompatible with GPL) ;
- Used mainly for libraries (ex : GNU libC).

Others

- Licence of Linux kernel is GPL with added terms ;
- Core of the kernel is GPL ;
- But using and redistribution of binaries only inside are allowed ;
- Must fit with standard interfaces of modules ;
- A lot of work has been done to clarify the GPL/non GPL frontier : EXPORT_SYMBOL_GPL, MODULE_LICENSE...

Licences

Technically very difficult to provide compatible binary modules because of all possible configurations of the kernel (ex : versions, external patches...) ; Legal but **not** advised (no support from community).

6.1.4 Glossary

- API , Application Programming Interface :
- AVL , Adelson-Velskii and Landis : binary tree structure always balanced that allow researches in O(log n) where a classic research is in O(n) :

- BE , Big-Endian : bytes organization where most significant byte (MSB) are stored before less significant bytes (LSB), :
- DMA , Direct Memory Access : Mechanism in Computer that bring the ability to avoid the CPU to be used when we want data transfer between central memory and devices;

- FAQ , Frequently Asked Questions : the document to be read before asking any question on a mailing list :
- FSF , Free Software Fundation : association that aim to defend and promote free-ware (on GPL and LGPL Licenses) :
- GPL , General Public Licence : free license of Free Software Fundation :
- IP , Internet Protocol : (level 3 in OSI Model) based best effort principle,
- IRQ , Interrupt ReQuest : ;

- ISDN , Integrated Services Digital Network : numerical network that can transport voice or data :
- LE , Little-Endian : bytes organization where less significant byte (LSB) are stored before most significant bytes (MSB), :
- LGPL , Lesser General Public Licence : free licence of Free Software Fundation mainly used for libraries :
- OSI , Open System Interconnect : layer model bringing a conceptual and standardize for exchanging between different operating systems ;

- POSIX , Portable Operating System for Computer Environment : Unix Standard from IEEE (number 1003.1) that specify system kernel ; it is composed of extensions for real-time kernel (1003.1-b) and threads (1003.1-c) :
- PLIP , Parallel Line Internet Protocol : communication protocol on parallel port used to embed IP communications through parallel lines :
- PPP , Point to Point Protocol : communication protocol used to embed communications on serial lines (ex : modem) :
- SLIP , Serial Line Internet Protocol : communication protocol used to embed IP communications on serial lines (ex : modem) ;

- SMP , Symmetric Multi-Processing : use of multiple same CPU that share the same resources (memory...) and devices in one operating system $\Rightarrow$ transparent for users :
- TCP , Transmission Control Protocol : (4th level in OSI model) working in connected mode :
- VFS , Virtual File System : abstraction layer in high level part of the kernel to enable fast writing on file systems ;