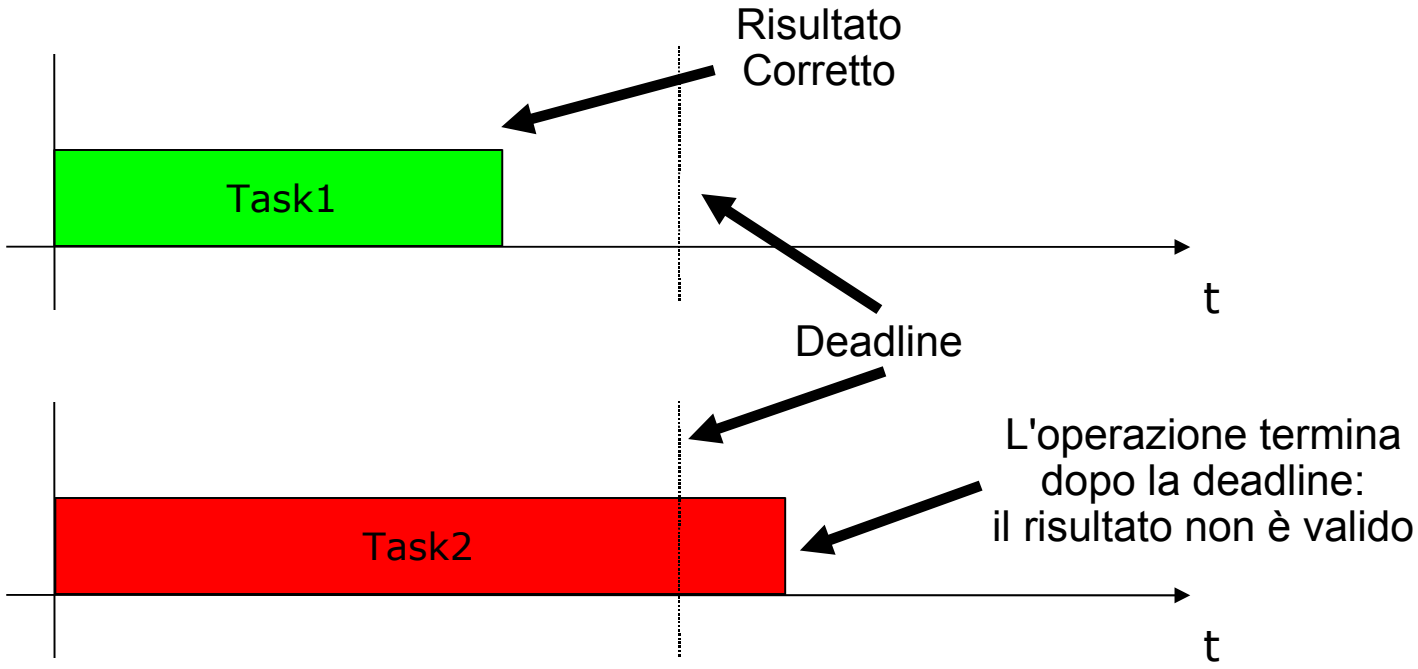
CONTROLLI AUTOMATICI LS
Ingegneria Informatica

INTRODUZIONE A RTAI-LINUX

Ing. Gianluca Palli
DEIS - Università di Bologna
Tel. 051 2093903
email: gpalli@deis.unibo.it
http://www-lar.deis.unibo.it/~gpalli

- Definizione di sistema operativo real time:
  - E' un sistema operativo in cui per valutare la correttezza delle operazioni si considera anche la variabile tempo;
  - Lo scheduler dei processi agisce secondo precise specifiche temporali.



RTOS: Definizioni

- Due tipologie di correttezza devono essere garantite:
  - Correttezza logica: i risultati/risposte forniti devono essere quelli previsti (normalmente richiesta a qualunque sistema di elaborazione)
  - Correttezza temporale: i risultati devono essere prodotti entro certi limiti temporali fissati (deadlines) (specifica dei sistemi real time)
- Tipologie di Sistemi real time
  - Sistemi Hard Real Time:
    - Il non rispetto delle deadlines temporali NON e' ammesso
    - porterebbe a danneggiamento del sistema (safety critical)
  - Sistemi Soft Real Time:
    - il non rispetto delle deadlines e' ammissibile
    - le specifiche temporali indicano solo in modo sommario i tempi da rispettare
    - degrado delle performace accettabile

RTOS: Definizioni

- In generale i sistemi real time complessi saranno ibridi hard/soft
  - Vi saranno deadline inderogabili
    - es: gestione delle condizioni di allarme pericolose
    - controllo digitale in retroazione
    - reazione ad eventi interni
  - Mentre altre non saranno stringenti
    - effettivo avviamento di una macchina dopo il comando ricevuto da un sistema di supervisione
    - salvataggio dei dati su dispositivi di archiviazione
    - interazione con l'uomo

- Attenzione: spesso si confonde il concetto di Hard e Soft Real Time con quello di Real Time "Stretto" e "Largo"
  - Real Time "Stretto": vincoli temporali (deadlines) stretti rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste
  - Real Time "Largo": vincoli temporali (deadlines) larghi rispetto ai tempi di calcolo necessari per eseguire le operazionirichieste

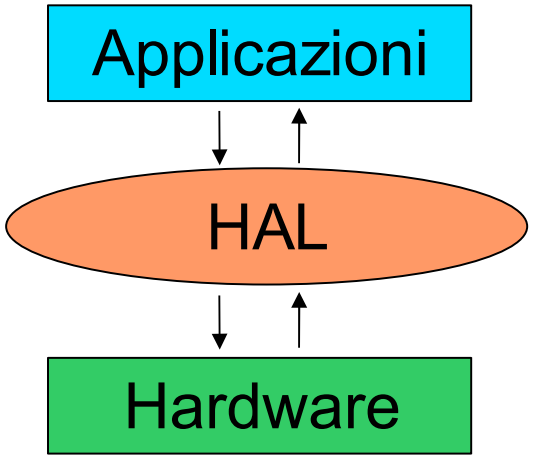
RTOS: Definizioni

- La “larghezza” o “strettezza” di un sistema di elaborazione dell’informazione real time dipende dalla piattaforma HardWare utilizzata
  - tempi di esecuzione dipendono dalla ‘potenza’ dell'unità di elaborazione (velocità + risorse)
- Purtroppo spesso i sistemi Hard Real Time sono anche stretti
  - normalmente le deadlines vicine sono anche inderogabili
  - per motivi di costo si sceglie sempre una piattaforma di elaborazione non sovrabbondante rispetto alle esigenze
- Nel caso di sistemi REAL TIME STRETTI e’ fondamentale la corretta organizzazione delle fasi di elaborazione per il rispetto delle deadlines
  - bisogna organizzare le attività di elaborazione al fine di ottimizzare i tempi
  - massimo sfruttamento delle risorse HW
    - forte legame tra il codice e l’HW di elaborazione
    - difficile abbinare astrazione e ottimizzazione

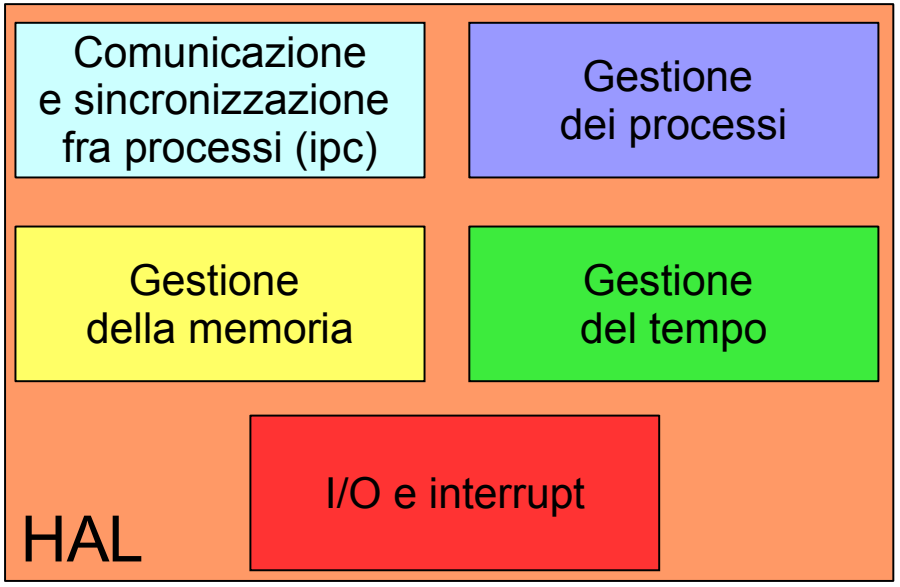
RTOS: Definizioni

- I sistemi real time devono in genere svolgere più attività (task) che possono anche essere completamente indipendenti.
- Le attività sono usualmente innescate a seguito del verificarsi di particolari condizioni (eventi) e devono terminare prima di certe deadlines
  - L'intervallo di tempo tra evento e deadline viene detto "time scope" (finestra temporale) dell'attività
  - I time scope di diverse attività possono essere sovrapposti
    - ESECUZIONE PARALLELA
- La realizzazione parallela delle attività dipende dalla architettura HW utilizzata.
  - Sistemi monoprocessore (parallelismo logico, ma non reale)
  - Sistemi multiprocessore (parallelismo anche reale)
    - ravvicinati (collegamento molto veloce, es.: sulla stessa board)
    - distribuiti (collegamento più lento, es.: via rete ethernet)
- Parallelismo logico gestito con "Programmazione Concorrente"

- I sistemi operativi real time fornisco alle applicazioni un'astrazione dell'hardware (Hardware Abstraction Layer, HAL).



- L'Hardware Abstraction Layer mette a disposizione delle applicazioni una serie di servizi (primitive real time).



RTOS: Applicazioni

- Campi di applicazione dei sistemi operativi real time:
  - Macchine automatiche
  - Automobili
  - Computers
  - Sistemi di telecomunicazione
  - Centrali Elettriche
  - Avionica
  - ecc...

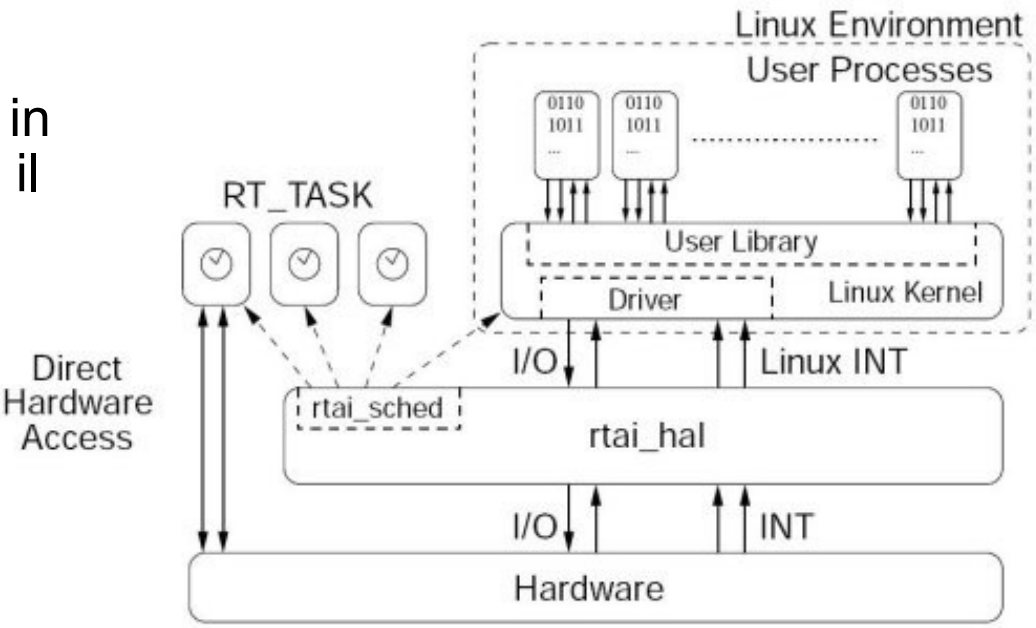
- I sistemi operativi real time sono “nascosti”:
  - La loro principale applicazione è in sistemi embedded;
  - Non sono studiati per interagire con l'utente (uomo);
  - La programmazione di questi ambienti avviene solitamente con l'ausilio di altri sistemi operativi non real time.

- La maggior parte dei sistemi operativi real time consiste in soluzioni proprietarie studiate appositamente per l'hardware su cui deve funzionare
- Alcuni dei sistemi operativi real time “generici”:
  - Soluzioni proprietarie:
    - VxWorks
    - QNX
    - RTLinuxPro
    - ...
  - Soluzioni opensource:
    - RTLinuxFree (solo per kernel 2.4)
    - **RTAI**
    - ...
- RTLinux e RTAI sono soluzioni basate sul kernel di Linux
  - Possono sfruttare tutte le applicazioni e l'ambiente del sistema operativo di partenza.

- Progetto originale del Politecnico di Milano
  - http://www.aero.polimi.it/~rtai/
  - http://www.rtai.org/

- L'ambiente real time è realizzato tramite moduli kernel:
  - ogni modulo gestisce un particolare servizio;
  - disponibilità di diversi scheduler;
  - possibilità di attivare e disattivare il supporto alle applicazioni real time;
  - modifica trasparente alle applicazioni non real time.

- Aggiunge al kernel di Linux le primitive real time
  - Linux (kernel+applicazioni) diventa un processo di RTAI eseguito quando la CPU non è occupata nell'esecuzioni di processi real time
    - ATTENZIONE!!!: Se i processi real time occupano la CPU al 100%, le applicazioni non real time risulteranno bloccate...
  - Le primitive originali di Linux (syscall) e gli interrupt vengono filtrati in modo da non disturbare il funzionamento del sistema real time
  - La programmazione avviene tramite i tools standard di Linux



Caratteristiche di RTAI

- Infrastruttura hard real time sia in kernel space che in user-space:
  - programmazione in user space più semplice;
  - implementazione in kernel space più efficiente ma potenzialmente più pericolosa per il sistema.

- Funzionalità real time avanzate:
  - supporto al C++ (anche in kernel space per kernel 2.4);
  - supporto al calcolo in virgola mobile in kernel space
  - possibilità di backtrace delle applicazioni real time;
  - disponibilità di una interfaccia grafica;
  - ...

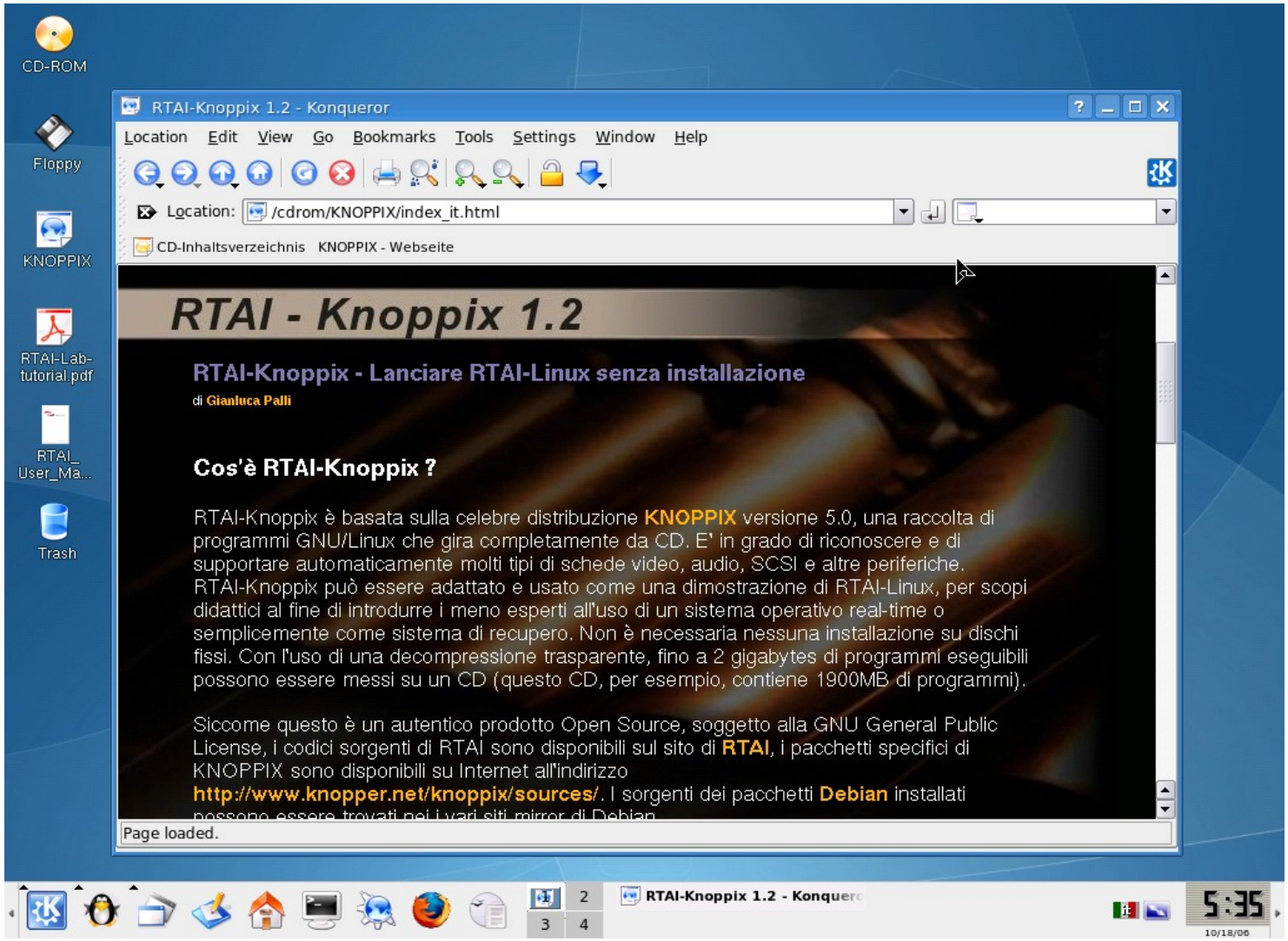
RTAI-Knoppix: boot del sistema

- Si tratta di un LiveCD che mette a disposizione dell'utente un ambiente real time senza la necessità di fare alcuna installazione.
  - si basa sulla distribuzione knoppix-5.0.1
    - la versione del kernel di linux è la 2.6.17
    - la versione di RTAI è la 3.4
- Boot del sistema:
  - impostare il BIOS del proprio PC in modo da effettuare il boot dal CDROM e riavviare il pc con il CD inserito nel lettore;
  - comparirà il seguente messaggio:

premere invio per avviare con le impostazioni di default.

RTAI-Knoppix: schermata iniziale

- Schermata iniziale



Il primo programma in ambiente RTAI

- Una semplice applicazione: rt_hello.c

```
#include <sys/mman.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_lxrt.h>
#include <rtai_math.h>
```

- Inclusione degli header files
  - gestione della memoria
  - definizioni generali di RTAI
  - RTAI scheduler
  - supporto user space RT
  - gestione della FPU

```
int main(void){
  RT_TASK *maintask;
  RTIME period;
  int i;
  float sample_time=1.0;
```

- Funzione principale
  - l'esecuzione del programma inizia dal main()
  - definizione delle variabili locali

```
  rt_allow_nonroot_hrt();
  mlockall(MCL_CURRENT | MCL_FUTURE);
```

- Funzioni di accesso alle risorse
  - acceso alle primitive real time ad utenti generici (non root)
  - blocco della memoria per evitare che l'applicazione venga “swappata”

```
...
```

Il primo programma in ambiente RTAI


```
...
if( !(maintask = rt_task_init_schmod( nam2num("MAINTASK"),
   50, 0, 0, SCHED_FIFO, 0))) {
    printf( "CANNOT INIT MAINTASK\n");
    exit( -1);
  }
```

```
  rt_task_use_fpu(maintask,1);
  rt_make_hard_real_time();
```

```
  rt_set_oneshot_mode();
  start_rt_timer(0);
```

```
...
```

- Parte principale del programma:
  - rt_task_init... inizializzazione del task e inserimento nello scheduler
  - deve essere inserita in tutti i processi che accedono alle primitive real time
- Alcune funzioni accessorie
  - accesso del task alla FPU
  - restrizione dell'ambiente alle sole primitive real time
- Inizializzazione del timer per la gestione dello scheduler
  - impostazione della modalità del timer
  - attivazione del timer alla massima velocità

Il primo programma in ambiente RTAI


```
...
  period=nano2count(sample_time*1E9);
  rt_task_make_periodic(maintask,rt_get_time()+period,
   period);
  for(i=0;i<10;i++){
    rt_task_wait_period();
    printf("Hello World at time %llu !!!\n",
        rt_get_time_ns());
  }
  rt_task_delete(maintask);
  return 0;
}
```

- Impostazione della periodicità
  - conversione del periodo di campionamento in intervalli del timer di sistema
  - il task viene reso periodico
- Ciclo principale
  - attesa di attivazione da parte dello scheduler
  - esecuzione del corpo del programma
- Chiusura dell'applicazione
  - cancellazione del task
  - uscita dal main

Compilazione di rt_hello.c

- In ambiente Linux, il compilatore più diffuso è sicuramente il gcc (GNU Compiler Collection)
  - In realtà non si tratta di un singolo compilatore ma di una collezione di tool per la creazione degli eseguibili
  - Supporta molti fra i più diffusi linguaggi di programmazione (C, C++, Pascal, Java, Fortran, ecc...)
- Comando per la compilazione:

```
rtaiuser@rtaibox:~$ gcc rt_hello.c -o rt_hello \
    -I/usr/realtime/include -L/usr/realtime/lib \
    -llxrt -lpthread -static
```

  - l'opzione `-I/usr/realtime/include` indica al compilatore dove si trovano gli header files di RTAI
  - l'opzione `-L/usr/realtime/lib` indica al compilatore dove si trovano le librerie di RTAI
- Per approfondimenti sul gcc consultare il manuale (`man gcc`)

Esecuzione di rt_hello

- Caricamento dei moduli per la gestione dei processi realtime (realtime scheduler)

  ```
  root@rtaibox:~$ modprobe rtai_sched
  ```

  - per approfondimenti consultare i manuali (`man modprobe`)
  - per eseguire questo comando è necessario possedere i diritti di superutente
- Esecuzione del programma:

  ```
  rtaiuser@rtaibox:~$ ./rt_hello
  ```

  - esaminare i messaggi del kernel con il comando dmesg
  - verificare cosa cambia togliendo dal codice l'istruzione `rt_task_make_hard_hrt();`
- Problema: l'istruzione `printf()` non è studiata per funzionare in modalità real time:
  - il kernel segnala che non è possibile gestire questa istruzione mantenendo il task in hard real time
  - cambio di modalità forzato

Stampa di messaggi a video

- Soluzione: si delegano le operazioni non real time ad un secondo processo (non real time)
  - necessari meccanismi di comunicazione fra i due processi
  - uso di FIFO real time (eseguire `modprobe rtai_fifos`)
- Il programma rt_hello.c viene modificato nel seguente modo:

```
  ...
#include <rtai_fifos.h>
#define FIFODIM 128
#define DEBUGFIFO 1
  ...
  char buf[FIFODIM];
  ...
  rtf_create(DEBUGFIFO,sizeof(char)*FIFODIM*10);
  for(i=0;i<10;i++){
    rt_task_wait_period();
    sprintf(buf,"Hello World  at time %lld !!!\n",
        rt_get_time_ns());
    rtf_put(DEBUGFIFO,buf,FIFODIM);
  }
  ...
```

- gestione delle comunicazioni
  - creazione della fifo real time per lo scambio dei messaggi fra i due processi
  - scrittura dei messaggi sulla fifo

Stampa di messaggi a video

- Ora tutte le istruzioni vengono eseguite in modalità hard real time
  - la funzione `sprintf()` non effettua chiamate di sistema percui non causa problemi alle applicazioni real time
  - Il secondo processo accede alla FIFO per la lettura dei messaggi

```
int main(void){
    char     buf[ 128];
    int      len;
    if((fd_debug = open("/dev/rtf1", O_RDONLY))==-1) {
        printf("Error opening fifo: %i\n",fd_debug);
        exit(-1);
    }
    while(1) {
       if( 0 > (len = read( fd_debug, buf, 128))) return 0;
        buf[ len] = "\0";
        printf( "%s", buf);
    }
    close( fd_debug);
    return 0;
}
```

- gestione delle comunicazioni
  - apertura della FIFO in sola lettura
- lettura dei messaggi dalla FIFO
- scrittura dei messaggi a video
- chiusura della FIFO

Stampa di messaggi a video

- Uso dei thread per la cattura dei messaggi di debug

```
...
#include <pthread.h>
...
void *debug_logging(void *p){
    ...
    close( fd_debug);
}
```

- Definizione della funzione di gestione del thread
  - i parametri e il valore ritornato dalla funzione devono essere necessariamente di questo tipo
  - Il corpo della funzione rimane inalterato
  - è stata rimossa l'istruzione `return()`

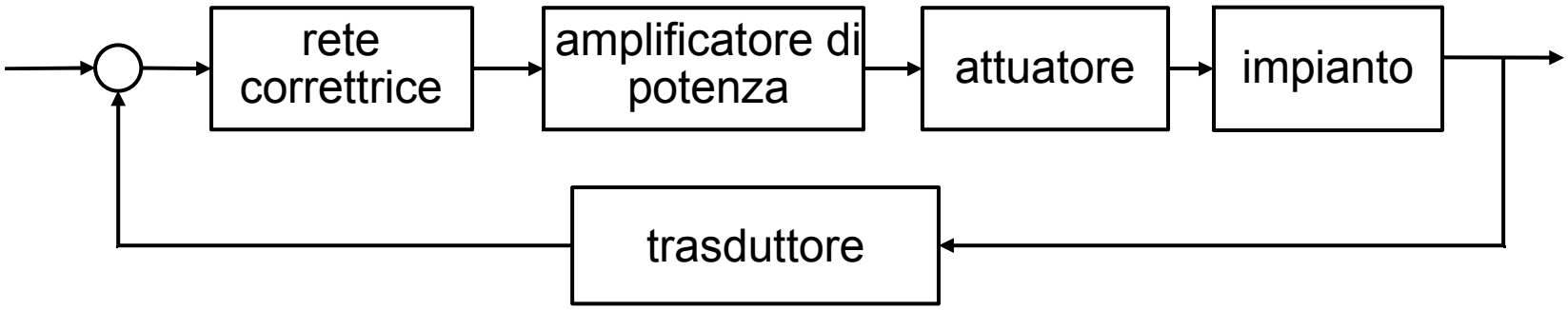
- Creazione del thread all'interno del main

```
...
int main(void){
    ...
    pthread_t debug_thread;
    ...
    if(pthread_create( &thread1, NULL, debug_logging, (void *) NULL)!=0){
      printf("main: Unable to create debug_thread\n");
      exit(-1);}
    ...
}
```

- Generazione del thread
  - definizione della struttura che contiene i parametri del thread
  - generazione del thread

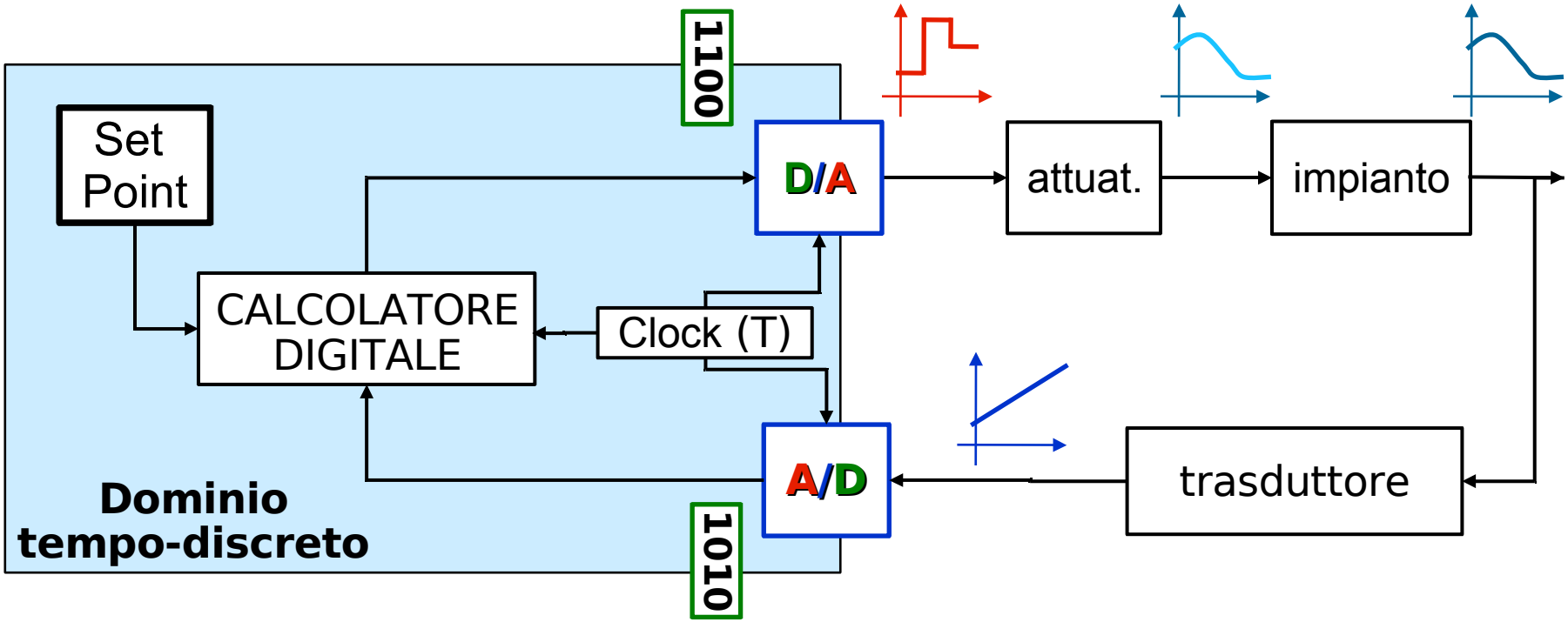
Implementazione degli algoritmi di controllo

- Sistemi di controllo analogici
  - L'elaborazione della legge di controllo è svolta in maniera tempo-continua
    - ad es. attraverso circuiti elettrici o idraulici



Implementazione degli algoritmi di controllo

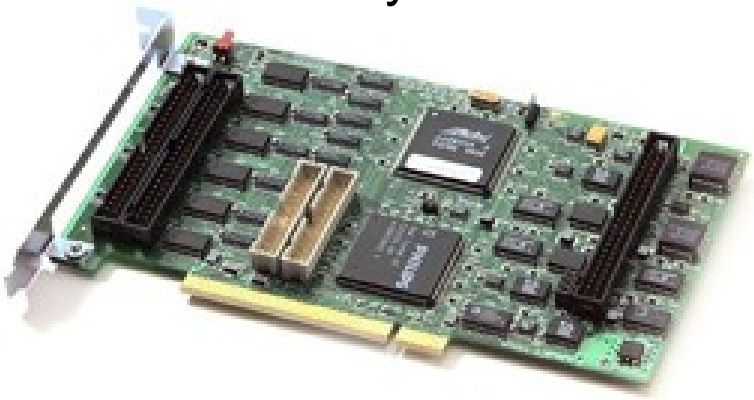
- Sistemi di controllo digitale
  - Presenza di un calcolatore nel loop di controllo
  - **Elaborazione tempo-discreta della legge di controllo**
  - Occorrono dispositivi di interfaccia
    - **tra il dominio tempo-continuo dell'impianto**
    - **e quello tempo-discreto del regolatore**



L'interfaccia D/A e A/D

- Sul calcolatore devono essere presenti dei dispositivi per la conversione di grandezze digitali in analogiche e viceversa
  - Esistono molte soluzioni in commercio con caratteristiche molto diverse anche per PC standard (DAQ board)

Sensoray 626



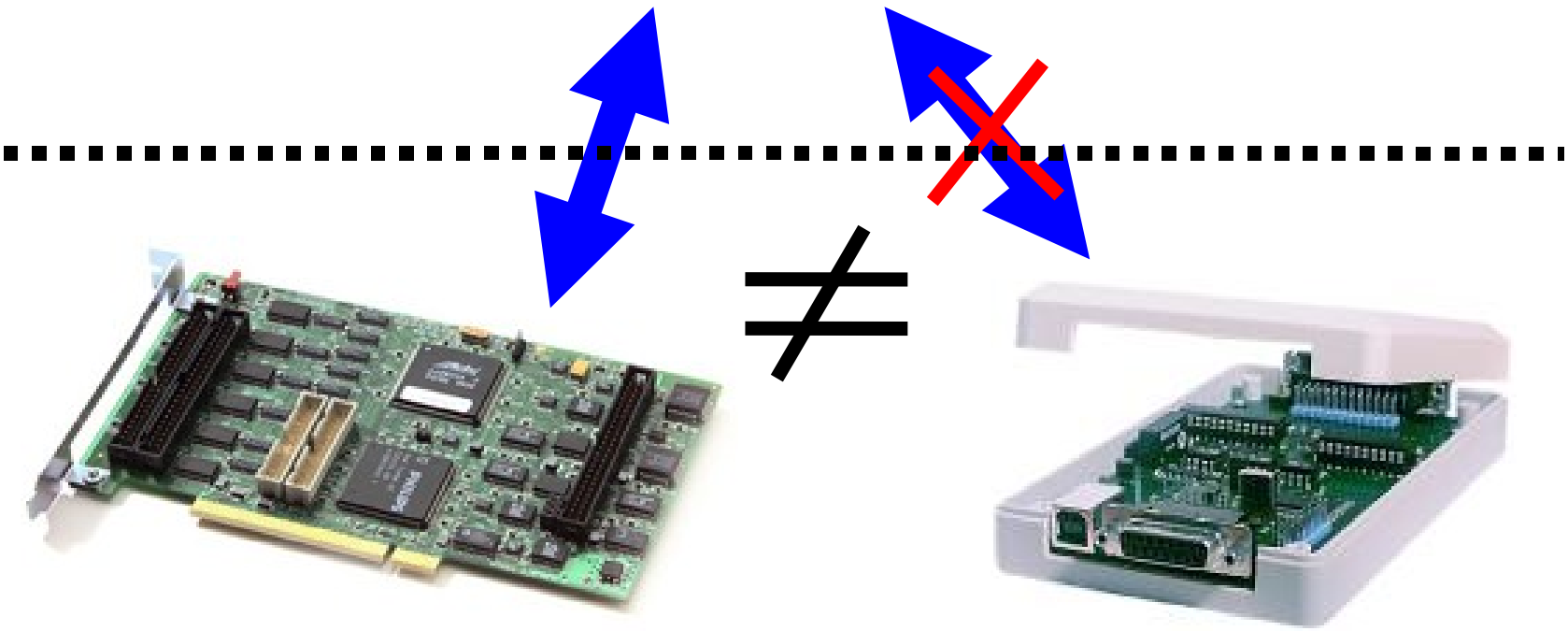
USBDUX

- PROBLEMA: Le interfaccie software con questi dispositivi cambiano radicalmente a seconda del produttore dell'hardware
  - Il codice di controllo deve essere studiato appositamente per comunicare con l'hardware di acquisizione utilizzato

IL CODICE NON È PORTABILE VERSO PIATTAFORME DIFFERENTI

Ambiente
real time

Applicazione
RTAI



Hardware di acquisizione dati

- Le scheda di acquisizione dati (DAQ board) sono accomunate da alcune caratteristiche:
  - N ingressi di conversione analogico/digitale (A/D channels)
    - lunghezza di parola tipicamente compresa fra 8 e 16 bits
    - range +/- 5 V o +/- 10 V
    - ingressi differenziali o single ended
    - ...
  - N ingressi di conversione analogico/digitale (A/D channels)
    - lunghezza di parola tipicamente compresa fra 12 e 16 bits
    - range +/- 5 V o +/- 10 V
    - uscite differenziali o single ended
    - ...
  - N canali di input/output digitali (I/O channels)
    - possibilità di associare eventi o interrupt ai vari canali
  - N contatori con funzioni di encoder/timer (conter channels)
  - ...

Programmazione delle DAQ board

- E' necessario conoscere a fondo l'hardware utilizzato per realizzare applicazioni efficenti
  - non sempre i costruttori forniscono driver per linux
  - non sempre i driver forniti sono studiati per funzionare in ambienti real time

- Soluzione: Realizzazione di una libreria standard per la gestione dei dispositivi di acquisizione dati
  - che fornisca una interfaccia unificata
  - che si adatti ai vari hardware
    - COMEDI: The Control and Measurement Device Interface
      - sviluppata per Linux
      - supporta nativamente RTAI e RTLinux
      - supporta centinaia di DAQ board
      - possibilità di realizzare il supporto per nuovi hardware
      - COMEDI project homepage: http://www.comedi.org/

L'interfaccia COMEDI

- Sfruttando l'interfaccia fornita da COMEDI, il codice prodotto non risente del particolare hardware di acquisizione utilizzato
  - facilità di programmazione dovuta all'unificazione dell'interfaccia
  - il codice risulta portabile su hardware differenti
- L'accesso ai dispositivi di acquisizione avviene tramite file
  - un file per ogni dispositivo installato nel sistema
  - possibilità di gestire contemporaneamente più dispositivi anche di tipo diverso tramite la stessa interfaccia
- Configurazione di tipo dinamico
  - è possibile cambiare “al volo” l'associazione tra file e dispositivo
  - si possono creare hardware virtuali che simulano il comportamento di sistemi fisici
    - utile per testare algoritmi di controllo in fase di progetto
    - è possibile passare al controllo del sistema reale semplicemente cambiando il dispositivo associato al file di interfaccia

Ambiente real time

Applicazione RTAI

COMEDI

Hardware virtuale

Utilizzo della libreria COMEDI

- Una semplice applicazione che accede alla scheda di acquisizione:

```
...
#include <rtai_fifos.h>
#include <comedilib.h>
...
   comedi_t *cf;
   lsampl_t ai_data,ao_data=0xfff;
...
   cf=comedi_open("/dev/comedi0");
   comedi_data_write(cf, 1, 0, 0,
         AREF_GROUND,ao_data);
...
   for(i=0;i<10;i++){
     rt_task_wait_period();
     comedi_data_read(cf, 0, 0, 0,
        AREF_GROUND,&ai_data);
     sprintf(buf,"Data from chan 0 = %d !!!\n", ai_data);
     rtf_put(DEBUGFIFO,buf,FIFODIM);
   }
...
```

- Inclusione degli header files
  - gestione delle FIFO real time
  - header file di COMEDI
- Dichiarazione delle variabili
  - descrittore del dispositivo COMEDI
  - variabili di ingresso e uscita
- Comunicazione con il dispositivo
  - apertura del file di interfaccia
  - scrittura sul canale di uscita (input del sistema)
  - lettura dal canale di ingresso (uscita del sistema

Configurazione di COMEDI

- Per la gestione delle FIFO real time è necessario caricare il modulo rtai_fifo

```
root@rtaibox:~$ modprobe rtai_fifos
```

- Il supporto alla libreria COMEDI di RTAI si attiva caricando l'opportuno modulo

```
root@rtaibox:~$ modprobe rtai_comedi
```

- Dopo aver caricato il supporto alle comedi, è necessario impostare la gestione dell'hardware
  - nel nostro caso, caricheremo il modulo del sistema virtuale

```
root@rtaibox:~/esercitazione$ insmod ./dcmotor.ko
```

  - un file per ogni dispositivo installato nel sistema

```
root@rtaibox:~$ comedi_config /dev/comedi0 dcmotor
```

Messa in scala delle variabili

- Problema: i dati letti e scritti da COMEDI sono interi senza segno
  - rappresentazione dei registri dei dispositivi di acquisizione
  - necessità di effettuare la messa in scala delle variabili
    - il valore 0 indica il valore minimo per il dispositivo
    - il valore massimo dipende dalla lunghezza di parola
  - COMEDI mette a disposizione delle funzioni per effettuare la scalatura dei valori di ingresso ed uscita

```
...
   int ai_maxdata, ao_maxdata;
   comedi_range *ai_rangetype, *ao_rangetype;
   float ai_volts, ao_volts;
...
   ai_maxdata=comedi_get_maxdata(cf, 0, 0);
   ai_rangetype=comedi_get_range(cf, 0, 0, 0);
   ao_maxdata=comedi_get_maxdata(cf, 1, 0);
   ao_rangetype=comedi_get_range(cf, 1, 0, 0);
...
```

- Variabili per la conversione
  - valori massimi del dispositivo
  - descrittore del range
  - grandezze fisiche
- Inizializzazione delle variabili
  - lettura dei valori massimi
  - lettura dei range

Messa in scala delle variabili


```
...
  ao_volts=1.0;
  ao_data=comedi_from_phys(ao_volts,
       ao_rangetype,ao_maxdata);
...
  ai_volts=comedi_to_phys(ai_data,
       ai_rangetype,ai_maxdata);
  sprintf(buf,"Data from chan 0
       = %3.3f !!!\n", ai_volts);
  rtf_put(DEBUGFIFO,buf,FIFODIM);
...
```

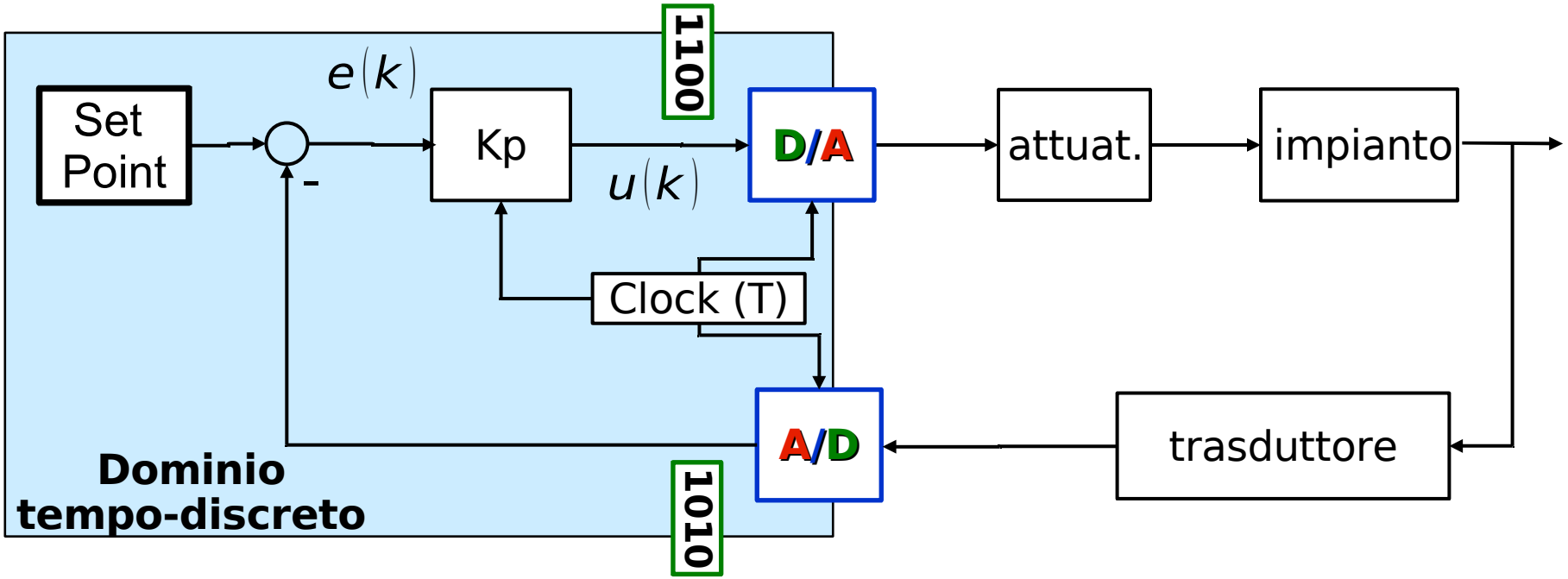
- Funzioni per la messa in scala
  - conversione del valore d'uscita
  - conversione del valore d'ingresso

- Le funzioni di messa in scala restituiscono il valore delle grandezze fisiche di ingresso e di uscita
  - tipicamente espresse in volt [V]
    - sono le tensioni lette dai sensori e applicate agli attuatori
    - modificando opportunamente il range dei canali è possibile ottenere scalature di grandezze fisiche diverse

Un esempio di controllo in retroazione

- La legge di controllo in retroazione più semplice da implementare è il controllo proporzionale
  - si calcola l'errore come differenza fra setpoint e uscita
  - l'azione di controllo si ottiene moltiplicando l'errore per una costante Kp detta guadagno del controllore

$$u(k) = Kp\ e(k)$$



Un esempio di controllo in retroazione

- Vediamo come realizzare questa semplice legge di controllo

```
...
  float setpoint=1.0;
  float ctrl_gain=5.0;
...
  while(isrunning){
    rt_task_wait_period();
    comedi_data_read(cf, 0, 0, 0,AREF_GROUND, &ai_data);
    ai_volts=comedi_to_phys(ai_data,ai_rangetype,ai_maxdata);

    ao_volts=ctrl_gain*(setpoint-ai_volts);

    ao_data=comedi_from_phys(ao_volts,ao_rangetype,ao_maxdata);
    comedi_data_write(cf, 1, 0, 0, AREF_GROUND, ao_data);
  }
...
```

- Acquisizione degli ingressi
  - Lettura dei valori dal dispositivo
  - messa in scala degli ingressi

$$u(k) = Kp\ e(k)$$

- Attuazione delle uscite
  - messa in scala delle uscite
  - scrittura del valore sul dispositivo

Dalla Z-trasformata all'equazione alle differenze

- Data una funzione di trasferimento tempo discreta nella forma

$$G(z)=\frac{y(z)}{u(z)}=\frac{b_0+b_1 z^{-1}+b_2 z^{-2}}{1+a_1 z^{-1}+a_2 z^{-2}}$$

si scrive

$$y(z)\left[1+a_1 z^{-1}+a_2 z^{-2}\right]=u(z)\left[b_0+b_1 z^{-1}+b_2 z^{-2}\right]$$

- Passando al dominio tempo discreto esplicitando l'operatore di ritardo si ottiene l'equazione alle differenze

$$y(k)+a_1 y(k-1)+a_2 y(k-2)=b_0 u(k)+b_1 u(k-1)+b_2 u(k-2)$$

da cui

$$y(k)=-a_1 y(k-1)-a_2 y(k-2)+b_0 u(k)+b_1 u(k-1)+b_2 u(k-2)$$

- Per l'implementazione in C si fa uso di buffer a scorrimento per salvare i valori degli ingressi e delle uscite

L'equazione alle differenze in C

- Vediamo come implementare l'equazione alle differenze

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2)$$

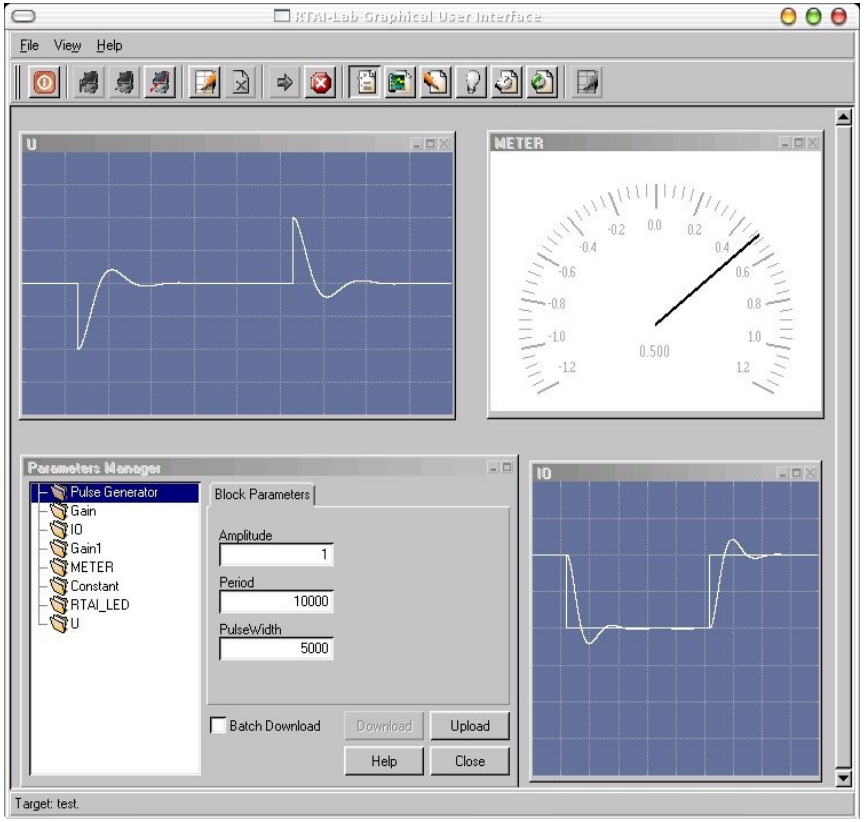
in linguaggio C

```
...
  float a[]={0.9, 0.5};
  float b[]={0.99, 0.7, 0.5};
  float y[]={0.0, 0.0};
  float u[]={0.0, 0.0, 0.0};
...
  while(isrunning){
...
    u[2]=u[1];
    u[1]=u[0];
    u[0]=ai_volts;
    ao_volts=-a[0]*y[0]-a[1]*y[1]+b[0]*u[0]+b[1]*u[1]+b[2]*u[2];
    y[1]=y[0];
    y[0]=ao_volts;
...
  }
```

- Definizione delle variabili
  - coefficenti dell'equazione alle differenze
  - vettori degli ingressi e delle uscite passate
- implementazione dell'equazione
  - aggiornamento degli ingressi
  - calcolo della nuova uscita
  - aggiornamento delle uscite

Interfacciamento con la grafica

- xrtailab è una interfaccia grafica disponibile in RTAI con la quale è possibile monitorare i processi real time
  - dispone di vari oggetti grafici per il plottaggio dei dati
    - scope, meter, led ecc...
  - possibilità di salvare i dati campionati su file
  - permette di modificare alcuni parametri del sistema
  - è possibile monitorare processi su host remoti

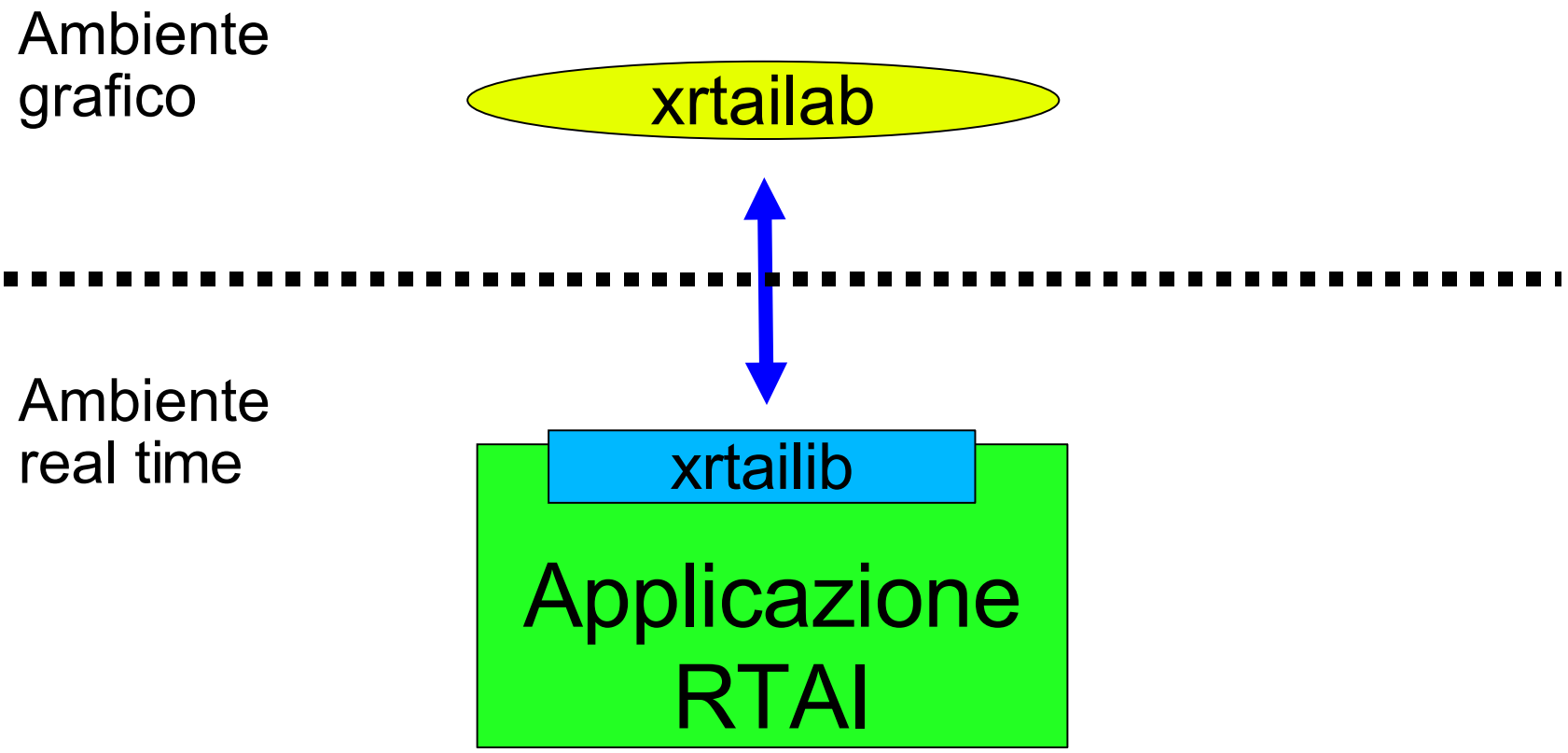


- Disponibiltà di una libreria per l'interfacciamento delle applicazioni
  - semplifica notevolmente la comunicazione con xrtailab
  - ancora in fase di sviluppo

Interfacciamento con la grafica

- Per utilizzare xrtailab è necessario caricare i seguenti moduli realtime:

```
root@rtaibox:~$ modprobe rtai_netrpc
root@rtaibox:~$ modprobe rtai_msg
```



Interfacciamento con la grafica

- Esempio di applicazione che fa uso della grafica

```
#include "xrtailib.h"
...
  double *data[2];

  rt_interface* my_interface;
  my_interface = new rt_interface;
...
  my_interface->add_scope(&ao_volts,0.01,
        output_traces,"Input");
  my_interface->add_scope(&ai_volts,0.01,
        input_traces,"Output");

  my_interface->add_param("Input",rt_SCALAR,
        SS_DOUBLE,1,1,&ao_volts);
  my_interface->add_param("Output",rt_SCALAR,
        SS_DOUBLE,1,1,&ai_volts);

  data[0]=&ao_volts;
  data[1]=&ai_volts;
...
```

- Dichiarazione delle variabili dell'interfaccia
  - puntatori alle variabili da modificare e ai relativi descrittori
  - puntatore all'interfaccia
  - creazione dell'interfaccia
    - Inizializzazione dei gestori degli oggetti grafici
      - vengono creati due scope
- Inizializzazione dei descrittori delle variabili modificabili
  - inizializzazione dei descrittori
  - puntatori alle variabili

Interfacciamento con la grafica


```
...
  while(isrunning){
...
  }
...
delete my_interface;
...
```

- Distruzione dell'interfaccia
  - Dopo il ciclo principale
  - Chiusura dell'interfaccia

- Nel file xrtailib.cpp e xrtailib.h è disponibile la documentazione sulle funzioni dell'interfaccia ad xrtailab
  - interfaccia in linguaggio C++
    - semplifica notevolmente la programmazione
  - possibilità di modificare il codice (anche quello di xrtailab)

CONTROLLI AUTOMATICI LS
Ingegneria Informatica

INTRODUZIONE A RTAI-LINUX
FINE

Ing. Gianluca Palli
DEIS - Università di Bologna
Tel. 051 2093903
email: gpalli@deis.unibo.it
http://www-lar.deis.unibo.it/~gpalli