# Informatique industrielle A7-19571
# Systèmes temps-réel

*J.F.Peyre*

## Partie 7 : Exemple d'application temps réelle en Ada et en C/Posix

# Plan du cours

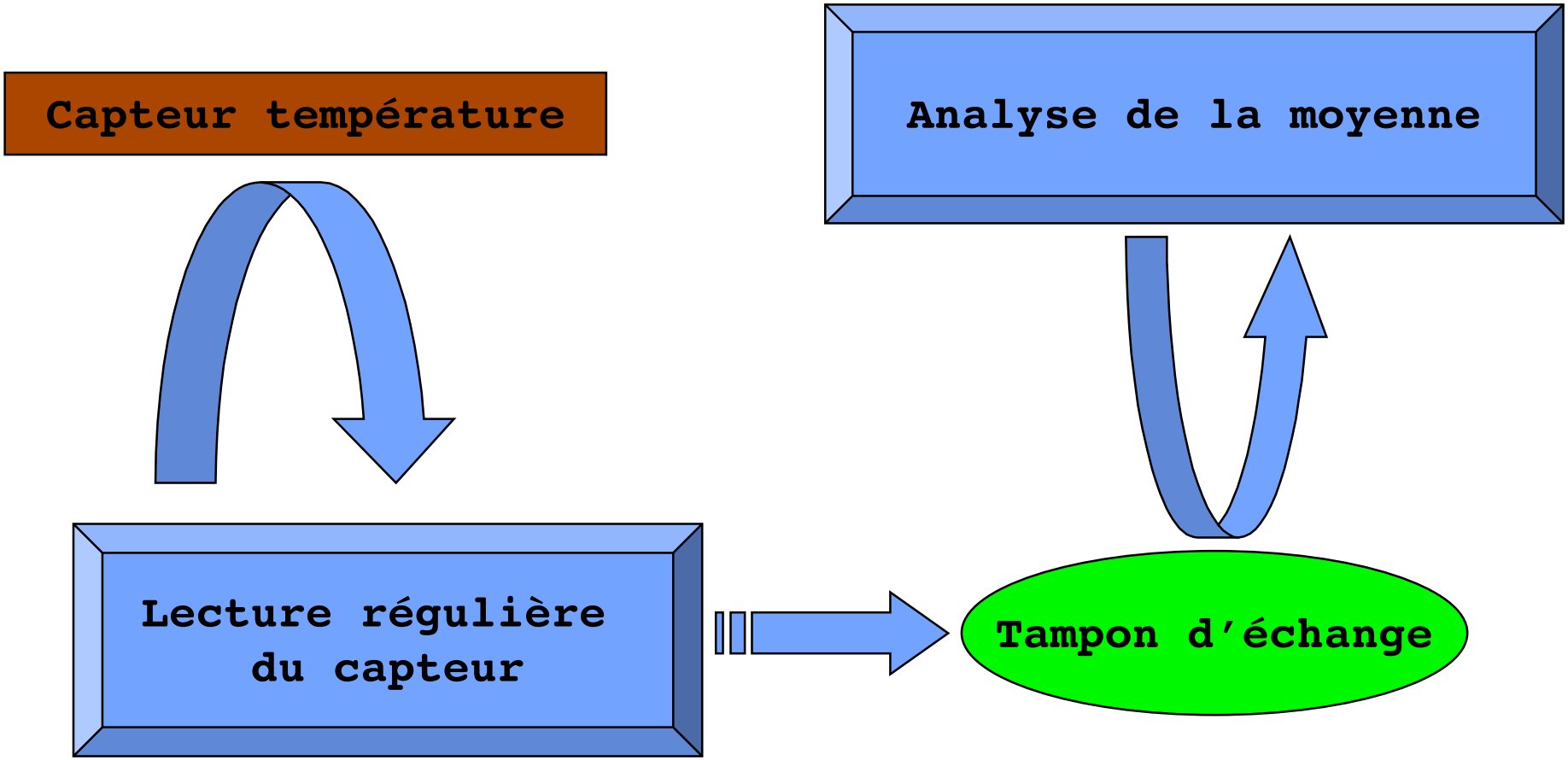
- Présentation d'un problème
- Solution en Ada
- Solution en C/Posix

# Présentation du problème

# Description du problème

- Un robot doit envoyer régulièrement des informations de température
- Il dispose pour cela d'un capteur de température
- On décide d'attribuer à une tâche le travail de lecture régulière du capteur (toute les secondes) et de transmettre à une autre tâche la valeur moyenne sur 10 mesures
- Le seconde tâche fait un pré-traitement de cette moyenne (par exemple calcul du minimum et du maximum de ces données) et envoie à son rythme ces informations vers un central
- On modélise dans ce problème le capteur et les deux tâches; l'interaction du robot et du central n'est pas abordée.

# Schéma de l'application



# Implantation de la solution

- On décide de modéliser le tampon d'échange par un moniteur assurant l'exclusion mutuelle entre la tâche de calcul et celle de lecture du capteur; de plus ce moniteur permettra de gérer l'attente de la moyenne (qui n'est obtenue qu'après 10 lectures du capteur)

- On suppose que le traitement fait par la tâche de calcul est de durée inférieure à 10 secondes (une lecture toute les secondes et 10 lectures pour la moyenne). Dans le cas contraire, il y aurait une perte de donnée entre la tâche de lecture et celle de calcul.

- Le capteur sera modélisé par une variable « mappée » en mémoire ce qui correspond à un cas concret

# Solution du problème en Ada

# Solution Ada *(1/5)*

```
with Ada.Text_IO;                    use Ada.Text_IO;
with System;                         use System;
with System.Storage_Elements;        use System.Storage_Elements;
with Ada.Calendar;                   use Ada.Calendar;
with Ada.Numerics.Discrete_Random;

procedure Appli is

   type Registre_8_Bits is new Integer range 0 .. 255;
   for Registre_8_Bits'Size use 1*8;

   Capteur_Temperature : Registre_8_Bits := 254;

   pragma Atomic (Capteur_Temperature);

   --  for Capteur_Temperature'Address use To_Address(16#FFFF_0040#);

   Periode_Lecture : Duration := 1.0;
   Nombre_Donnees  : Natural := 10;
```

*Valeur sur 8 bits*

*Opérations de lecture et d'écriture atomiques (directive au compilateur)*

*Stocker la variable à l'adresse exprimée en hexadécimal*

# Solution Ada *(2/5)*

```
protected Tampon is
    procedure Met_A_Jour (V : in Registre_8_Bits);
    entry Lecture_Quand_Pret (V : out Registre_8_Bits);
private
    Valeur : Registre_8_Bits;
    Pret : Boolean := False;
end Tampon;
```

```
protected body Tampon is
    procedure Met_A_Jour (V : in Registre_8_Bits) is
    begin
         Valeur := V;
         Pret := True;
    end Met_A_Jour;

    entry Lecture_Quand_Pret (V : out Registre_8_Bits) when Pret is
    begin
         V := Valeur;
         Pret := False;
    end Lecture_Quand_Pret;
end Tampon;
```

# Solution Ada *(3/5)*

*Prochain réveil = heure courante + période*

*Attendre heure prochain réveil*

```
task Lecture_Reguliere;
task body Lecture_Reguliere is
    Somme      : Integer := 0;
    Cpt        : Natural := 0;
    Next_Time  : Time := Clock + Periode_Lecture;
begin
    loop
        delay until Next_Time;

        Somme := Somme + Integer (Capteur_Temperature);
        Cpt := Cpt + 1;
        if (Cpt =  Nombre_Donnees) then
                Somme := Somme / Nombre_Donnees;
                Tampon.Met_A_Jour (Registre_8_Bits (Somme));
                Cpt := 0;
                Somme := 0;
        end if;
        Next_Time := Next_Time + Periode_Lecture;
        Put ('.');
    end loop;
end Lecture_Reguliere;
```

*Calculer heure prochain réveil*

# Solution Ada *(4/5)*

```
task Calcul;
task body Calcul is
    Moyenne_Temp : Registre_8_Bits;
    Le_Min       : Registre_8_Bits := Registre_8_Bits'Last;
    Le_Max       : Registre_8_Bits := Registre_8_Bits'First;
begin
    loop
        Tampon.Lecture_Quand_Pret (Moyenne_Temp);

        if (Moyenne_Temp < Le_Min) then
                Le_Min := Moyenne_Temp;
        end if;
        if (Moyenne_Temp > Le_Max) then
                Le_Max := Moyenne_Temp;
        end if;
        Put ("Valeur temp = " & Registre_8_Bits'Image (Moyenne_Temp));
        Put (" Min = " & Registre_8_Bits'Image (Le_Min));
        Put (" Max = " & Registre_8_Bits'Image (Le_Max));
        New_Line;
    end loop;
end Calcul;
```

*Lecture de la température moyenne*
*Opération bloquante*

*Utilisation de la donnée*
*Opération concurrente à la lecture régulière de la température*

# Solution Ada

*(Simulation de la variation de la température 1/2)*

```
task Simul_Variation_Temperature;

task body Simul_Variation_Temperature is

   package Registre_8_Bits_Random is new Ada.Numerics.Discrete_Random (Registre_8_Bits);
   use Registre_8_Bits_Random;
   Gen : Generator;

   Valeur_Cible : Registre_8_Bits;
   Difference   : Integer;

   Periode_Changement : Duration := 1.5;
   Next_Time          : Time := Clock + Periode_Changement;
```

./...

# Solution Ada

*(Simulation de la variation de la température 2/2)*

```
begin
    Reset (Gen);
    loop
        Valeur_Cible := Random (Gen);

        loop
            Difference := Integer (Valeur_Cible) - Integer (Capteur_Temperature);

            exit when Difference = 0;

            if (Difference > 0) then
              Capteur_Temperature := Capteur_Temperature + 1;
            else
              Capteur_Temperature := Capteur_Temperature - 1;
            end if;
            delay until Next_Time;
            Next_Time := Next_Time + Periode_Changement;
        end loop;
    end loop;

end Simul_Variation_Temperature;
```

# Solution Ada *(5/5)*

*(le main)*

```
begin -- du main
   null;
end Appli;
```

# Solution du problème en C / Posix

# Solution C/Posix *(1/7)*

*Valeur sur 8 bits*

```
#include <stdio.h>
#include <pthread.h>

typedef unsigned char Registre_8_Bits;

Registre_8_Bits *capteurTemperature; // a mapper sur adresse registre

unsigned int periodeLecture = 1000*1000; // 1 seconde = 10⁶ micro secondes

unsigned int nombreDonnees = 10; // 10 lectures avant de transmettre

pthread_mutex_t mutexTampon = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  attenteDonnee; //initialisé dans le principal avant de lancer les taches
```

*Pour la gestion du tampon d'échange*

# Solution C/Posix *(2/7)*

```
// ============================================================================
// le code associé à la gestion du tampon (1/2) : utilisation d'une variable conditionelle
// ============================================================================

Registre_8_Bits valeurTampon = 0;
int valeurPrete = 0;

void Tampon_metAJour(Registre_8_Bits v){

    pthread_mutex_lock( &mutexTampon);
    valeurTampon = v;
    valeurPrete = 1;
    pthread_cond_signal( &attenteDonnee);
    pthread_mutex_unlock( &mutexTampon);

}
```

# Solution C/Posix *(3/7)*

```
// ========================================================================================
// le code associé à la gestion du tampon  (2/2)
// ========================================================================================

void Tampon_lectureQuandPret(Registre_8_Bits *v){

    pthread_mutex_lock( &mutexTampon);

    while ( !valeurPrete){
        pthread_cond_wait( &attenteDonnee, &mutexTampon);
    }

    *v = valeurTampon;
    valeurPrete = 0;
    pthread_mutex_unlock( &mutexTampon);

}
```

# Solution C/Posix *(4/7)*

```
// ============================================================================
// le code associé à la tâche de lecture régulière du registre; version simplifiée pour la
// gestion du temps; une version plus juste devrait faire appel à la notion d'évènemment Posix
// ============================================================================
void codeLectureReguliere(void){
    int somme, cpt  = 0;

    while(1){
        usleep(periodeLecture);
        somme += *capteurTemperature;
        cpt ++;
        if (cpt == nombreDonnees){
                somme /= nombreDonnees;
                Tampon_metAJour(somme);
                cpt = somme = 0;
        }
    }
}
```

# Solution C/Posix *(5/7)*

```
// ============================================================
// le code associé à la tâche de calcul
// ============================================================

void codeCalcul(void){
    Registre_8_Bits moyenneTemp;
    Registre_8_Bits leMin = 255;
    Registre_8_Bits leMax = 0;

    while(1){
        Tampon_lectureQuandPret(&moyenneTemp);
        if (moyenneTemp < leMin)
                leMin = moyenneTemp;
        if (moyenneTemp > leMax)
                leMax = moyenneTemp;
        printf("Valeur temp = %d\n", moyenneTemp);
        printf("min = %d, max = %d", leMin, leMax);
    }
}
```

# Solution C/Posix *(6/7)*

```
int main(){
   pthread_t tacheCalcul;
   pthread_t tacheLecture;

   pthread_cond_init( &attenteDonnee, NULL);
   capteurTemperature = (Registre_8_Bits *) malloc(1*sizeof(Registre_8_Bits));

   pthread_create(&tacheLecture, NULL, (void *(*)()) codeLectureReguliere, NULL);
   pthread_create(&tacheCalcul, NULL, (void *(*)()) codeCalcul, NULL);

   pthread_join(tacheLecture, NULL); // ne doit jamais arriver
   pthread_join(tacheCalcul, NULL); // ne doit jamais arriver

   printf("Bizarre : terminaison des deux tâches -> problemes \n");
   return -1;
}
```

# Solution C/Posix *(7/7)*

*(Simulation (simple) de la variation de la température)*

```
int main(){
    pthread_t tacheCalcul;
    pthread_t tacheLecture;
    int i;

    pthread_cond_init( &attenteDonnee, NULL);
    capteurTemperature = (Registre_8_Bits *) malloc(1*sizeof(Registre_8_Bits));

    pthread_create(&tacheLecture, NULL, (void *(*)()) codeLectureReguliere, NULL);
    pthread_create(&tacheCalcul, NULL, (void *(*)()) codeCalcul, NULL);

    for (i=0; i<100; i++){
         (*capteurTemperature) ++;
         usleep(500*1000);
    }
    for (i=0; i<100; i++){
         (*capteurTemperature) --;
    usleep(500*1000);    }

    return -1;
}
```

# Conclusion

- **Les systèmes temps-réel sont en pleine expansion**
- **Ils ne sont pas toujours simples à appréhender du fait des différentes contraintes de ces systèmes (concurrence et temps concret)**
- **L'utilisation de deux langages ou interface normalisés nous a permis d'aborder les point importants de cette discipline sans entrer dans les spécificités de tel ou tel système**
- **Ce cours peut être approfondie par**
  - une étude plus complète des systèmes (et des réseaux)
  - une étude plus poussée sur la programmation concurrente et temps-réel