

Partie 2 : Exercices

Contrôleur de température (& Pression)

Déplacements d'un Robot

Game of Life

CPE - Avril-Mai 2021

(Version élèves)

ASG

I Projets CPE : Python Concurrent

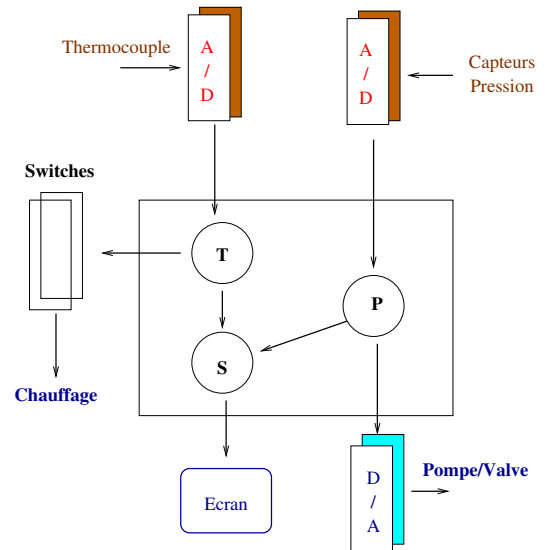
- Ce document contient 3 sujets :
 - Contrôleur de température / Pression
 - Simulation des déplacements d'un Robot
 - Game of life

I-1 Réalisation d'un système multi-tâche de contrôle de Température et Pression

Difficulté : **/*****

On considère le système (*temps réel embarqué*) simple suivant :

- Un **processus T** lit les valeurs d'un ensemble de thermocouples (par l'intermédiaire d'un convertisseur analogique-numérique, CDA).
→ T commande les changements appropriés à un chauffage (par l'intermédiaire d'un commutateur à commande numérique).
 - Le **processus P** a une fonction similaire pour la pression (il emploie un au convertisseur numérique-analogique, DAC).
 - T et P doivent communiquer des données au **processus S**, qui présente des mesures à un opérateur par l'intermédiaire d'un écran.
 - Notez que P et T sont les entités actives ; S est une ressource (il répond juste aux demandes de T et de P) : il peut être mis en application comme ressource protégée ou serveur s'il agit plus intensivement avec l'utilisateur (avec différents réglages et *consignes* possibles).
 - L'objectif global de ce système temps réel embarqué est de maintenir la température et la pression d'un certain processus chimique dans des limites définies.
 - Un vrai système de ce type serait clairement plus complexe, permettant par exemple à l'opérateur de modifier les limites (consignes).
 - Le but de ce système est de conserver la température et la pression d'un processus chimique dans des limites spécifiées.
 - Un vrai système sera plus complexe, par exemple, permettre à un opérateur de modifier ces limites.
 - Deux approches : synchrone (cyclique) et asynchrone.
- T : Thermocouple
P : Pression
S : Écran (screen)
- On distingue plusieurs entités concurrentes :
 - Gestionnaire de la température (T)
 - Gestionnaire de la pression (P)
 - Gestionnaire du Chauffage
 - Gestionnaire de la Pompe
 - La tâche Ecran (S)
 - Un contrôleur pour coordonner l'ensemble
 - Par ailleurs, nous utiliserons une zone mémoire partagée pour ...
et protégée par un verrou (un sémaphore est également possible).



Déclarations :

Ver : Verrou – cf. TAS
 Seuil_T, Seuil_P : réel \leftarrow ..
 go_pompe : bool \leftarrow faux
 go_chauffage : bool \leftarrow faux
 mem_xx : mémoire **partagée**

☞ mémoire **partagée :**

si *thread* (ou task ADA) utilisés alors
 une variable globale si non un *shmem*.
 → Certains langages proposent des variables **protégées** = variable globale +
 Verrou mutex

Tâche Contrôleur :

Répéter toutes les X secondes
 verrouiller(Ver);
 $T \leftarrow Mem_T$ $P \leftarrow Mem_P$
 libérer(Ver);
 Si ($T > Seuil_T$)
 go_chauffage \leftarrow faux – pour le chauffage
 Si ($P > Seuil_P$)
 go_pompe \leftarrow vrai – pour la Pompe
 Sinon go_pompe \leftarrow faux
 Sinon Si ($T < Seuil_T$)
 go_pompe \leftarrow vrai
 go_chauffage \leftarrow vrai
 Sinon – $T = Seuil_T$
 go_chauffage \leftarrow faux
 Si ($P > Seuil_P$) go_pompe \leftarrow vrai
 Sinon go_pompe \leftarrow faux
 Fin Répéter

Tâche Chauffage :

Répéter toutes les Z secondes
 Si (go_chauffage) Alors
 "mettre en route"
 Sinon "arrêter"
 Fin si
 Fin Répéter

- En général, le contrôleur crée les tâches après sa propre création.

Tâche Température :

Répéter toutes les S secondes
 lire la valeur V sur le capteur
 Convertir_AD(V, T)
 verrouiller(Ver);
 Ecrire(T, Mem_T)
 libérer(Ver);
 Fin Répéter

Tâche Pression :

Répéter toutes les U secondes
 lire la valeur V sur le capteur
 Convertir_AD(V, P)
 verrouiller(Ver);
 Ecrire(P, Mem_P)
 libérer(Ver);
 Fin Répéter

Tâche Pompe :

Répéter toutes les Z secondes
 Si (go_pompe) Alors
 "mettre en route"
 Sinon "arrêter"
 Fin si
 Fin Répéter

Tâche Ecran :

Répéter
 verrouiller(Ver);
 $T \leftarrow Mem_T$
 $P \leftarrow Mem_P$
 libérer(Ver);
 écrire T et P
 Fin Répéter

- La gestion par les booléennes *go_pompe*, *go_chauffage* peut être remplacée par le mécanisme d'évènement (*Attendre*, *Signaler*) :
 → la tâche Pompe fera *Attendre(go_pompe)* conjugué avec *Signaler(go_pompe)* effectué par le Contrôleur.
- Ces booléennes n'ont pas besoin d'un accès en *mutex* car le *contrôleur* y écrit et Pompe (ou Chauffage) lisent.

I-2 Simulation des déplacements d'un Robot

Difficulté : ***/*****

- Un robot avec les caractéristiques suivants
 - Pas de but particulier : avancer et éviter les obstacles
 - Plusieurs capteurs : infra rouge (IR) sur les 2 côtés, sonar (US) frontal, de contact (Bumper) frontal
 - Les actions sur les servo moteurs : *avancer*, *reculer*, *tourner à gauche/droite*
 - Le comportement par défaut est : *avancer*
 - Un écran d'affichage de l'état
- Principes : lecture des capteurs

Déclarations :

Ver : Verrou – cf. TAS
 les Distances : réel \leftarrow ..
 les Drapeaux : bool \leftarrow faux
mem_xx : mémoire partagée

☞ mémoire **partagée** :

si *thread* (ou task ADA) utilisés alors
 une variable globale si non un *shmem*.
 → Certains langages proposent des variables **protégées** = variable globale + Verrou mutex

Tâche Controleur :

Répéter toutes les *X* secondes
 Commande \leftarrow "avancer"
 Drapeau \leftarrow faux
 Si (Drapeau_IR) Alors
 Commande \leftarrow Cmd_IR
 Drapeau \leftarrow Drapeau_IR
 Si (Drapeau_US) Alors
 Commande \leftarrow Cmd_US
 Drapeau \leftarrow Drapeau_US
 Si (Drapeau_BU) Alors
 Commande \leftarrow Cmd_BU
 Drapeau \leftarrow Drapeau_BU
 Transmettre *Commande* aux servos
 verrouiller(Ver);
mem_Cmd \leftarrow *Commande*
mem_Flag \leftarrow *Drapeau*
 libérer(Ver);
 Fin Répéter

Tâche Ecran :

Répéter toutes les *A* secondes
 verrouiller(Ver);
C \leftarrow *mem_Cmd*
F \leftarrow *mem_Flag*
 libérer(Ver);
 écrire *C* et *F*
 Fin Répéter

- En général, le contrôleur crée les tâches après sa propre création.

Tâche IR :

```

Répéter toutes les  $S$  secondes
  lire la valeur  $V_g$  sur le capteur gauche
  lire la valeur  $V_d$  sur le capteur droit
  Convertir_AD( $V_g, D_g$ )
  Convertir_AD( $V_d, D_d$ )
  Si  $D_g < d$  OU  $D_d < d$  Alors
    Drapeau_IR  $\leftarrow$  vrai
    Si  $D_g < d$  ET  $D_d < d$  Alors
      Cmd_IR  $\leftarrow$  "reculer"
    Sinon Si  $D_g < d$  Alors
      Cmd_IR  $\leftarrow$  "à gauche"
    Sinon Cmd_IR  $\leftarrow$  "à droite"
  Sinon Drapeau_IR  $\leftarrow$  faux
Fin Répéter

```

Tâche US :

```

Répéter toutes les  $K$  secondes
  lire la valeur  $V$  sur le capteur
  Convertir_AD( $V, D$ )
  Si  $D < d$  Alors
    Drapeau_US  $\leftarrow$  vrai
    Cmd_US  $\leftarrow$  "reculer"
  Sinon Drapeau_US  $\leftarrow$  faux
Fin Répéter

```

Tâche Bumper :

```

Répéter toutes les  $Z$  secondes ( $Z$  petit)
  Si (contact=1) Alors
    Drapeau_BU  $\leftarrow$  vrai
    Cmd_BU  $\leftarrow$  "reculer"
  Sinon Drapeau_BU  $\leftarrow$  faux
Fin Répéter

```

Remarque sur "Répéter toutes les X milli/micro/nanosecondes" :

Un moyen simple d'implanter ce délai :

```

Next  $\leftarrow$  temps actuel (clock)
Répéter
  Actions
   $Next \leftarrow Next + X$ 
   $delay$  until next
Fin Répéter

```

- Si $delay$ non disponible :

```

Temps  $\leftarrow$  temps actuel (clock)
Répéter
  Actions
  Next  $\leftarrow$  temps actuel (clock)
  Reste  $\leftarrow X - (Next - Temps)$ 
  Attendre(Reste) – e.g. usleep/sleep
  Temps  $\leftarrow$  Next
Fin Répéter

```

→ Bien entendu, $Reste > 0$ sinon, le système n'est pas RT!!

I-3 Game of Life

Difficulté : **/*****

Réaliser le jeu suivant dans une version **concurrente** avec les mécanismes de base graphique (*screen* comme dans la course Hippique).

Il s'agit d'une grille (matrice de taille d'au moins 15×15) dont les cases représentent soit un "être" vivant soit rien. L'état d'une case peut être modifié en fonction de son voisinage selon les règles décrites ci-dessous.

Extrait de l'énoncé d'origine :

- The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead.
- Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur :
 - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
 - Any live cell with two or three live neighbours lives on to the next generation.
 - Any live cell with more than three live neighbours dies, as if by overcrowding.
 - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
- The initial pattern constitutes the seed of the system.
- The first generation is created by applying the above rules simultaneously to every cell in the seed-births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one).
- The rules continue to be applied repeatedly to create further generations.

ASG, Avril-Mai 2021