

Partie 2 : Exercices

Contrôleur de température (& Pression)

Déplacements d'un Robot

Game of Life

CPE - Nov 2022

(Version élèves)

ASG

I Projets CPE : Python Concurrent

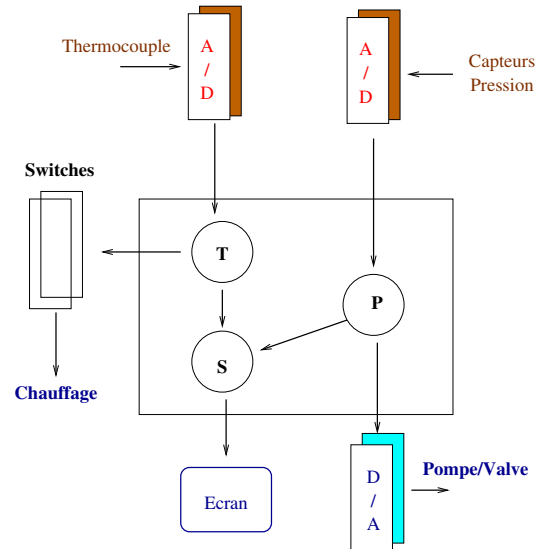
- Ce document contient 3 sujets :
 - Contrôleur de température / Pression
 - Simulation des déplacements d'un Robot
 - Simulation des serveurs/client/cuisiniers d'un restaurant
 - Game of life

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

Difficulté : **/*****, (4 points)

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 coordonnée l'ensemble
 - Par ailleurs, nous utiliserons une zone mémoire partagée pour ...
et protégée par un verrou (un sémaphore est également possible).



<p>Déclarations :</p> <p>Ver : Verrou – cf. TAS</p> <p>Seuil_T, Seuil_P : réel \leftarrow ..</p> <p>go_pompe : bool \leftarrow faux</p> <p>go_chauffage : bool \leftarrow faux</p> <p>mem_xx : mémoire partagée</p> <hr/> <p>☞ mémoire partagée :</p> <p>si <i>thread</i> (ou task ADA) utilisés alors une variable globale si non un <i>shmem</i>.</p> <p>→ Certains langages proposent des variables protégées = variable globale + Verrou mutex</p>	<p>Tâche Contrôleur :</p> <p>Répéter toutes les X secondes</p> <p>verrouiller(Ver);</p> <p>$T \leftarrow Mem_T \quad P \leftarrow Mem_P$</p> <p>libérer(Ver);</p> <p>Si ($T > Seuil_T$)</p> <p> go_chauffage \leftarrow faux – pour le chauffage</p> <p> Si ($P > Seuil_P$)</p> <p> go_pompe \leftarrow vrai – pour la Pompe</p> <p> Sinon go_pompe \leftarrow faux</p> <p> Sinon Si ($T < Seuil_T$)</p> <p> go_pompe \leftarrow vrai</p> <p> go_chauffage \leftarrow vrai</p> <p> Sinon – $T = Seuil_T$</p> <p> go_chauffage \leftarrow faux</p> <p> Si ($P > Seuil_P$) go_pompe \leftarrow vrai</p> <p> Sinon go_pompe \leftarrow faux</p> <p>Fin Répéter</p>	<p>Tâche Chauffage :</p> <p>Répéter toutes les Z secondes</p> <p> Si (go_chauffage) Alors</p> <p> "mettre en route"</p> <p> Sinon "arrêter"</p> <p> Fin si</p> <p>Fin Répéter</p>
---	--	---

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

Tâche Température :	Tâche Pression :	Tâche Pompe :	Tâche Ecran :
<p>Répéter toutes les S secondes</p> <p> lire la valeur V sur le capteur</p> <p> Convertir_AD(V, T)</p> <p> verrouiller(Ver);</p> <p> Ecrire(T, Mem_T)</p> <p> libérer(Ver);</p> <p>Fin Répéter</p>	<p>Répéter toutes les U secondes</p> <p> lire la valeur V sur le capteur</p> <p> Convertir_AD(V, P)</p> <p> verrouiller(Ver);</p> <p> Ecrire(P, Mem_P)</p> <p> libérer(Ver);</p> <p>Fin Répéter</p>	<p>Répéter toutes les Z secondes</p> <p> Si (go_pompe) Alors</p> <p> "mettre en route"</p> <p> Sinon "arrêter"</p> <p> Fin si</p> <p>Fin Répéter</p>	<p>Répéter</p> <p> verrouiller(Ver);</p> <p> $T \leftarrow Mem_T$</p> <p> $P \leftarrow Mem_P$</p> <p> libérer(Ver);</p> <p> écrire T et P</p> <p>Fin Répéter</p>

- 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.

III Simulation des déplacements d'un Robot

Difficulté : ***** / *******, (5 points)

☞ **La partie graphique doit impérativement être réalisée sous TkInter.** Les versions qui existent sur le WEB ne sont pas acceptées (ne sont pas réalisées avec Tkinter)!

- 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  $Vg$  sur le capteur gauche
  lire la valeur  $Vd$  sur le capteur droit
  Convertir_AD( $Vg, Dg$ )
  Convertir_AD( $Vd, Dd$ )
  Si  $Dg < d$  OU  $Dd < d$  Alors
    Drapeau_IR  $\leftarrow$  vrai
    Si  $Dg < d$  ET  $Dd < d$  Alors
      Cmd_IR  $\leftarrow$  "reculer"
    Sinon Si  $Dg < d$  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!!

Un système multi-tâches de simulation d'un restaurant

Difficulté : **/*****, (5 points)

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

1. simule des commandes de clients dans un restaurant
2. un certains nombre de serveurs en salle enregistrent ces commandes et les transmettent à la cuisine pour préparation
3. après leur préparation, les serveurs délivrent ces commandes aux clients

Dans la **version de base**, on n'identifie pas de cuisinier et ce sont les serveurs qui simulent la préparation des commandes (voir plus bas pour la version étendue).

Prévoir :

- s processus *serveur*. P. Ex. $s = 5$
- un processus *clients* qui simulera aléatoirement les commandes des clients selon une loi uniforme. Ce processus émettra une commande aléatoire toutes les p. ex. 3..10 *secondes* à l'adresse des serveurs.
- un processus *major_dHomme* qui s'occupera des affichages à l'écran
- un tampon de taille (p. ex.) 50 contiendra les commandes des clients ; les serveurs prélèvent des commandes de ce tableau
- une commande d'un client sera constituée d'un identifiant client (un entier) et une lettre $A..Z$ qui représentera le menu commandé

En l'absence d'interface graphique, on utilisera le module **curses** de Python que l'on a déjà utilisé dans l'exemple cours de chevaux. On affichera ainsi à l'écran les informations suivants :

- les commandes des clients (les paires $(id, menu)$) dès leur émission
- le serveur qui prend cette commande en charge et simule sa préparation (par un délai)
- le client qui reçoit sa commande préparée

☞ Les informations sont affichées exclusivement par le processus *major_dHomme*.

Un exemple d'affichage à l'écran :

Le serveur 1 traite la commande (id_i, C_i) (ou rien si pas de commande traité par ce serveur)

....

Le serveur s traite la commande (id_j, C_j)

Les commandes clients en attente : $[(id_i, C_i), (id_j, C_j) \dots (id_k, C_k)]$

Nombres de commandes attente : 5

Commande (id_u, U) est servie au client

Aller plus loin (Bonus) :

Ajouter un certain nombre de cuisiniers (en cuisine) qui préparent ces commandes et avertissent les serveurs. Le serveur qui avait enregistré la commande la délivre au client qui a commandée.

Ajouter à l'aversion de base :

- c processus *cuisto*. P. Ex. $c = 2$
- Modifier les affichages et présenter le cuisinier qui traite la commande.

Le contenu de l'écran sera augmenté des lignes :

Le cuisinier 1 prépare la commande $(id_1, A, serveur_1)$ (ou rien si pas de commande traité par ce cuisinier)

....

Le cuisinier c prépare la commande $(id_p, P, serveur_p)$

IV Game of Life

Difficulté : **/*****, (5 points)

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.
-