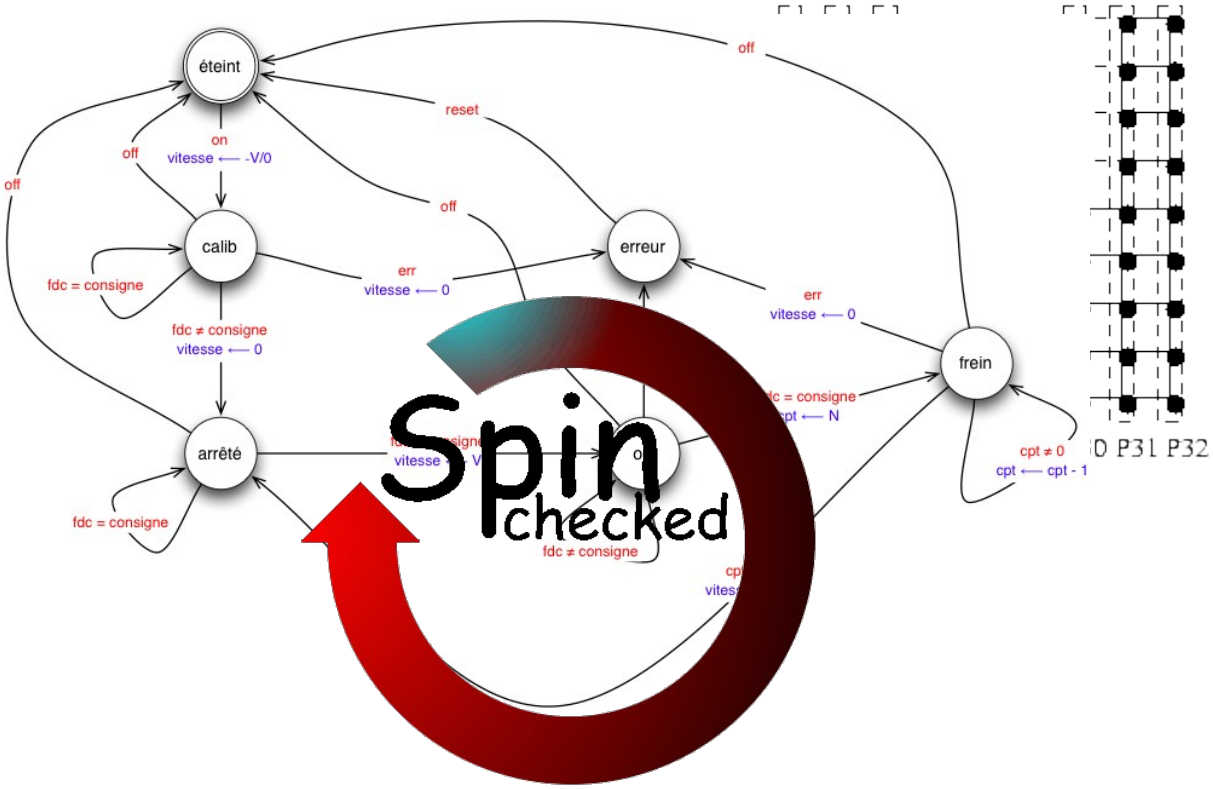
Rémi Mollard
Céline de Roland

25/04/2015

TP INFO 814

PARALLÉLISME



Objectifs du TP :

- Comprendre le comportement d'un système parallèle
- Analyser et manipuler des échanges de données entre processus
- Comprendre les différents types d'opération du parallélisme
- Adapter un problème industriel au parallélisme

TABLE DES MATIÈRES

I) INTRODUCTION, RAPPEL DU SUJET

1) Introduction

L’architecture des ordinateurs, qu’il s’agisse de microprocesseurs ou de supercalculateurs, est fortement influencée par l’exploitation d’une propriété fondamentale des applications : le parallélisme.

Un grand nombre d’architectures présentes dans les sites informatiques sont parallèles.

Ce type d’architecture touche une large gamme de machines depuis les PC biprocesseurs jusqu’aux supercalculateurs.

Aujourd’hui, la plupart des serveurs sont des machines parallèles (des multiprocesseurs).

Pour mieux comprendre l'implémentation des concepts du parallélisme, nous avons implémenté un problème de l'industrie du nucléaire en utilisant le langage Promela.

Promela est un langage de spécification de systèmes asynchrones. L'un des atouts majeur de ce langage est la possibilité de créer dynamiquement des processus et de les faires communiquer entre eux par des variables globales ou en utilisant des canaux de communication.

Le langage Promela est souvent associé à un outil de vérification du logiciel, nous utiliserons l'outil Spin dans le cadre de ce TP. Spin offre la possibilité de pouvoir vérifier de manière formelle des applications multithreads.

2) Rappel du sujet

Il est demandé d'implémenter :

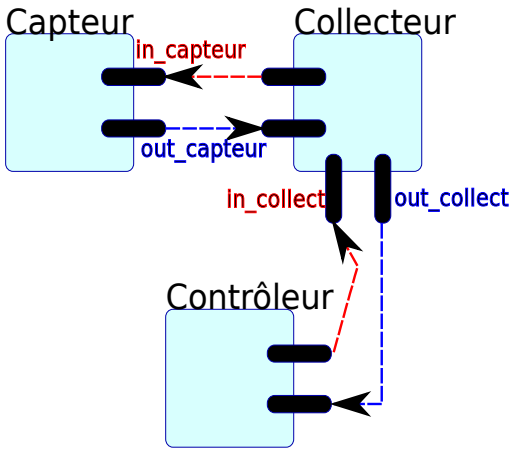
- Quatre capteurs de température pour surveiller le changement de température du cœur.
- Trois collecteurs pour rassembler les valeurs des retournées par les capteurs et émettre un diagnostique sur l'état de la température et des capteurs.
- Un contrôleur afin de faire la synthèse des informations transmises par les collecteurs.

Il sera également nécessaire d'utiliser une horloge pour enclencher la procédure de recueille des valeurs pour les quatre capteurs de manière périodique. Ce signal ne sera pas implémenté dans ce TP, nous avons pris la décision de lancer manuellement les contrôleurs et le collecteur.

Dés lors que les collecteurs seront actifs, ils contacteront chaque capteur pour obtenir une mesure de la température. Suite à la réception de ces valeurs le collecteur va émettre un diagnostic qui dépendra de l'état de la température et de a cohérence des valeurs retournées par les capteurs.

Pour finir le contrôleur fera la synthèse des trois diagnostics des collecteurs pour engager des actions sur les barres et sur le voyant d'état du système.

II) ARCHITECTURE ET FONCTIONNEMENT GÉNÉRAL



Chaque capteur est équipé de son propre canal in (afin de recevoir un ordre de mesure du collecteur) et son propre canal out (afin d'envoyer la valeur demandée au collecteur).

Chaque collecteur est équipé de son propre canal out (afin de recevoir un ordre de diagnostique du contrôleur) et son propre canal out (afin d'envoyer le diagnostique demandé au contrôleur).

Dans notre simulation, le contrôleur envoie un rendez vous sur les canaux in_collect de chaque collecteur lorsque l'utilisateur entre un numéro dans la console (correspondant au numéro du test prédéfini à exécuter, entre 1 et 5). Dans la réalité, cela se passera lorsqu'il recevra l'ordre de l'horloge. Après avoir envoyé toutes les demandes de rendez vous, il attend d'avoir obtenu tous les diagnostiques des collecteurs avant de décider d'abaisser ou non les barres et choisir la couleur du voyant.

Un collecteur, lorsqu'il a reçu le rendez-vous sur in_collect, envoie un rendez vous sur les canaux in_capteur de chaque capteur. Puis il attend que tous aient répondu pour établir son diagnostique et l'envoyer au contrôleur sur son canal out_collect.

Un capteur, lorsqu'il a reçu le rendez-vous sur in_capteur, effectue sa mesure (dans notre simulation, il va chercher la mesure dans le jeux de test sélectionné au début du cycle) puis la renvoie au collecteur sur son canal out_capteur.

III) AGENTS

1) Lanceur

Cet agent sert seulement à lancer les autres agents afin qu'ils soient prêts à effectuer des simulations. On ne lance pas le contrôleur, qui sera lancé par le simulateur à chaque cycle. Le contrôleur ne s'exécute pas en continue : il effectue un cycle puis s'arête, jusqu'à ce qu'il soit relancé par l'utilisateur dans la simulation ou le tic d'horloge dans la réalité.

$$Lanceur \stackrel{D}{=} Capteurs(0-3) \mid Collecteurs(0-2) \mid Simulateur$$

```
active proctype lanceur() {

	int i;
	// Initialisation des jeux de test
	for (i: 0..(NB_CAPTEURS * NB_COLLECTEURS - 1)) {
		test[0].valeur[i] = test01[i];
		test[1].valeur[i] = test02[i];
		test[2].valeur[i] = test03[i];
		test[3].valeur[i] = test04[i];
		test[4].valeur[i] = test05[i];
	}
	// Lancement des capteurs, des collecteurs, et du simulateur
	for (i: 0 .. NB_CAPTEURS - 1) {
		run Capteur(i)
	};
	for (i: 0 .. NB_COLLECTEURS - 1) {
		run Collecteur(i)
	};
	run Simulateur()
}
```

2) Simulateur

Cet agent réagit aux interactions de l'utilisateur avec la console :

- Touche échap (code 4) => quitter l'application
- Touche 1 (code 49) à 5 (code 53) => lancer un cycle en utilisant le jeux de test numéro 1 à 5
- Autre touche => ne rien faire

$$Simulateur \stackrel{D}{=} !\begin{pmatrix}[c=4](quit)\\ +[c\in[49;53]](Contrôleur)\\ +[else](nil)\end{pmatrix}$$

```
proctype Simulateur() {

	int c;

	do
		:: STDIN ? c ->
			if
```

```
                    // Touche échap : quitter
                    :: c == 4 -> break
                    :: (c == 49 || c == 50 || c == 51 || c == 52 || c == 53) ->
                    // Touches 1, 2, 3, 4 ou 5 : lancer le jeux de test
                          numTest = c - 49;
                          run Controleur()
                    // Autre touche : aucune action
                    :: else ->  printf("Illegal %d \n", c)
                    fi
      od
}
```

3) Contrôleur

Cet agent demande leurs diagnostiques aux contrôleurs et décide ensuite quoi faire (barres et voyant), puis s'arête.

$$\textit{Contrôleur} \stackrel{D}{=} \overline{\text{in\_collect[0]}}.\overline{\text{in\_collect[1]}}.\overline{\text{in\_collect[2]}} . \text{out\_collect[0]}(\textit{valeur}[0]).\text{out\_collect[1]}(\textit{valeur}[1]).\text{out\_collect[2]}(\textit{valeur}[2]) . \textit{nil}$$

```
proctype Controleur() {

      int i;
      int retourCollecteur[NB_COLLECTEURS];
      int nbTNormale;
      int nbDefail;
      int nbAlarm;

      // Envoyer les rendez vous aux collecteurs
      for (i: 0 .. NB_COLLECTEURS - 1) {
            in_collect[i] ! 0
      };
      // Recevoir les réponses des collecteurs
      for (i: 0 .. NB_COLLECTEURS - 1) {
            out_collect[i] ? retourCollecteur[i]
      };
      // Prendre une décision
      nbTNormale = 0; nbDefail = 0; nbAlarm = 0;
      for (i: 0 .. NB_COLLECTEURS - 1) {
            if
                  :: (retourCollecteur[i] == ALARME_TEMPERATURE) ->
                        nbAlarm++
                  :: (retourCollecteur[i] == TEMPERATURE_NORMALE) ->
                        nbTNormale++
                  :: (retourCollecteur[i] == DEFAILLANCE_CAPTEURS) ->
                        nbDefail++
            fi
      };
      if
            :: ( nbAlarm > 0 ) ->
                  printf("Baisse les barres, voyant rouge")
            :: ( nbDefail == NB_COLLECTEURS) ->
                  printf("Baisse les barres, voyant rouge clignotant")
            :: ( nbDefail == NB_COLLECTEURS - 1 && nbTNormale == 1) ->
                  printf("Baisse les barres, voyant orange")
            :: ( nbDefail < NB_COLLECTEURS - 1 && nbTNormale > 1
            && nbTNormale != NB_COLLECTEURS) ->
```

```
                printf("voyant orange")
            :: ( nbTNormale == NB_COLLECTEURS) ->
                printf("voyant vert")
    fi
}
```

4) Collecteur

Dès qu'il reçoit un rendez-vous sur son canal in_collect, cet agent demande aux capteurs de prendre une mesure et établit un diagnostique. Il s'exécute en boucle.

$$\begin{aligned} Collecteur(numCollecteur) \stackrel{D}{=} \; & !(\ \text{in\_collect}[\,numCollecteur\,] \\ & .\overline{\text{in\_capteur}[0]}.\overline{\text{in\_capteur}[1]}.\overline{\text{in\_capteur}[2]}.\overline{\text{in\_capteur}[3]} \\ & .\text{out\_capteur}[0](valeur[0]).\text{out\_capteur}[1](valeur[1]) \\ & .\text{out\_capteur}[2](valeur[2]).\text{out\_capteur}[3](valeur[3]) \\ & .\overline{\text{out\_collect}[\,numCollecteur\,]}\langle diagnostique\rangle\ ) \end{aligned}$$

```
proctype Collecteur(int numCollecteur) {

    int i;
    int valeur[NB_CAPTEURS];
    int valeursDifferentes[2] = { 0, 0};
    int nbValeurs[2] = {0, 0};
    int valeurCommune = 0;
    bool dejaVuDeuxiemeValeur = false;
    bool troisCapteursIdentiques = false;

    // Bang
    do
        // Recevoir une demande de rendez-vous
        :: in_collect[numCollecteur] ? _ ->
            printf("Collecteur %d reçu signal\n", numCollecteur);
            // Envoyer demandes de rendez-vous aux capteurs
            for (i: 0 .. NB_CAPTEURS - 1) {
                in_capteur[i] ! 0
            };
            // Recevoir les valeurs des capteurs
            for (i: 0 .. NB_CAPTEURS - 1) {
                out_capteur[i] ? valeur[i]
            };
            // Etablir un diagnostique
            valeursDifferentes[0] = valeur[0];
            valeursDifferentes[1] = 0;
            nbValeurs[0] = 1; nbValeurs[1] = 0;
            dejaVuDeuxiemeValeur = false;
            troisCapteursIdentiques = false;
            for (i: 1..(NB_CAPTEURS - 1)) {
                if
                    :: ( valeur[i] == valeursDifferentes[0] ) ->
                        nbValeurs[0] = nbValeurs[0] + 1
                    :: ( dejaVuDeuxiemeValeur && valeur[i] ==
valeursDifferentes[1]) ->
                        nbValeurs[1] = nbValeurs[1] + 1
                    :: ( !dejaVuDeuxiemeValeur && valeur[i] !=
valeursDifferentes[0]) ->
                        valeursDifferentes[1] = valeur[i];
                        nbValeurs[1] = nbValeurs[1] + 1;
```

```
                                        dejaVuDeuxiemeValeur = true
                                :: else -> skip
                        fi
                };
                if
                        :: (nbValeurs[0] > NB_CAPTEURS - 2) ->
                                valeurCommune = valeursDifferentes[0];
                                troisCapteursIdentiques = true
                        :: (nbValeurs[1] > NB_CAPTEURS - 2) ->
                                valeurCommune = valeursDifferentes[1];
                                troisCapteursIdentiques = true
                        :: else -> troisCapteursIdentiques = false
                fi;
                // Terminer d'établir le diagnostique et
                // Envoyer le diagnostique au contrôleur
                if
                        :: (troisCapteursIdentiques) ->
                                if
                                        :: (valeurCommune < SEUIL) ->
        out_collect[numCollecteur] ! TEMPERATURE_NORMALE
                                        :: else ->
        out_collect[numCollecteur] ! ALARME_TEMPERATURE
                                fi;
                        :: else ->
        out_collect[numCollecteur] ! DEFAILLANCE_CAPTEURS
                fi
        od
}
```

5) Capteur

Dès qu'il reçoit un rendez-vous sur son canal in_capteur, cet agent effectue une mesure de température et l'envoie sur son canal out_capteur. Il s'exécute en boucle.

$$Capteur(numCapteur) \stackrel{D}{=} !(\ \text{in\_capteur}[\,numCapteur\,].\overline{\text{out\_capteur}[\,numCapteur\,]}\langle température\rangle)$$

```
proctype Capteur(int numCapteur) {

        int cpt = 0;
        int valeur = 0;
        do
                // Recevoir une demande de rendez-vous
                :: in_capteur[numCapteur] ? _ ->
                        // Mesurer la température (ie piocher dans le jeu de test)
                        cpt = ((cpt+1) % NB_COLLECTEURS);
                        numCase =  NB_COLLECTEURS*numCapteur + cpt
                        valeur = test[numTest].valeur[numCase];
                        // Renvoyer la température
                        out_capteur[numCapteur] ! valeur
        od
}
```

IV) CONCLUSION

Tout d'abord, nous tenions à vous faire remarquer notre appréciation concernant le lien entre la partie théorique vue en cour et ce TP. La sémantique n'étant pas trivial à appréhender, ce TP nous à permis d'implémenter les définitions vues en cours dans un exemple concret d'actualités.

Le choix des technologies de supports (Promela, Spin et iSpin) nous à semblé vraiment en adéquation avec le sujet. Le langage Promela propose une multitudes de fonction très intéressantes que nous avons découverte en implémentant le système de capteurs. Il est cependant dommage que le temps accorder à ce TP ne nous à pas permis d'approfondir ces fonctions.

Nous tenions également à vous faire remarquer qu'il est vraiment appréciable d'avoir pris en compte notre emploie du temps et nos contraintes de fin d'années pour élaborer ce TP.