J3eA - Vol. 5 - Hors série N°2 (2006)
DOI: 10.1051/j3ea:2006044

Logiciel d'apprentissage du fonctionnement d'un noyau multi-tâche

Christian Meisgny, Hélène Perrin, Véronique Perdereau
Université Pierre et Marie Curie
4 place Jussieu, 75252 Paris Cedex 05
meisgny@ccr.jussieu.fr

RESUME

Dans le cadre d'un enseignement d'informatique sûre et temps réel, il s'est avéré utile de développer un logiciel de simulation du comportement interne d'un noyau multitâche. Les étudiants éprouvent des difficultés pour comprendre les problèmes d'ordonnancement et de synchronisation entre les tâches concurrentes d'une même application. Le support animé dans l'environnement Java permet l'analyse temporelle allant jusqu'au pas à pas des situations rencontrées dans ce type de programmation en temps réel. Ce logiciel offre la possibilité de tester des scénarios simples mettant en œuvre les primitives couramment rencontrées dans les noyaux embarqués. Les scénarios à tester sont écrits dans un langage simple à portée d'un débutant en programmation.

Mots clés : noyau, tâche, ordonnancement, commutation, synchronisation, animation, simulation, Java.

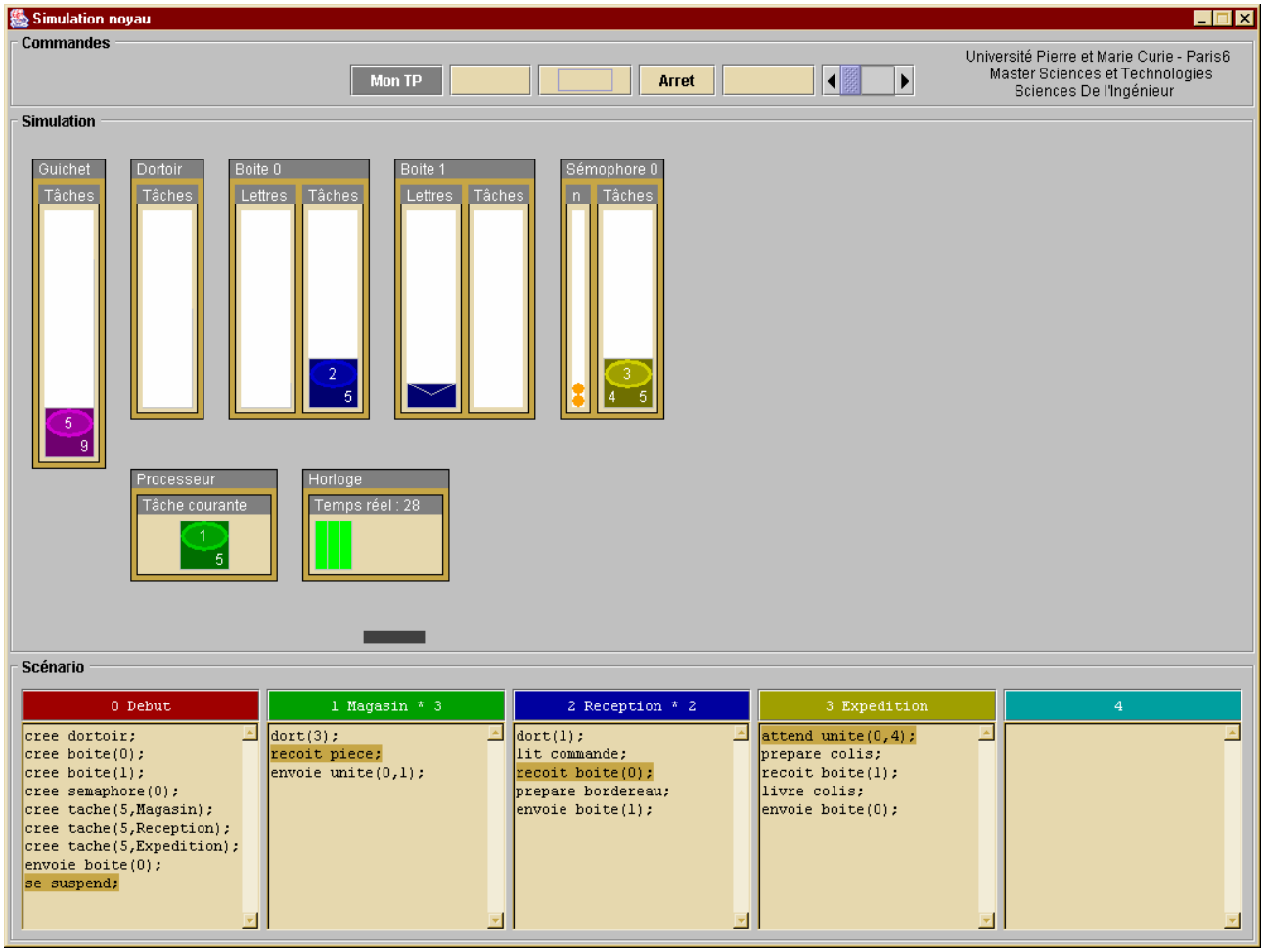

fig 1 : fenêtre graphique

1 INTRODUCTION

Lorsque l'on regarde les réalisations faites dans le domaine des TICE (Technologies de l'Information et de la Communication dans l'Enseignement) pour les matières scientifiques, on trouve beaucoup de produits destinés à l'enseignement de la physique, des mathématiques, de la chimie, …, mais peu de réalisations pour l'informatique. Et pourtant, il peut s'avérer très utile d'apporter une aide aux étudiants pour comprendre certains principes difficiles en leur permettant de faire la part des choses entre l'outil ou le langage, et l'objet de la programmation. Nous devons préciser ici que, quand nous parlons enseignement de l'informatique, il n'est pas question de l'apprentissage des outils tels quels (bureautique, Internet, logiciels divers), ni même d'un langage, mais de concepts fondamentaux tels que le fonctionnement d'un microprocesseur, d'un système d'exploitation, de réseaux de communication … L'outil de visualisation que nous présentons dans cet article a pour objectif d'illustrer le fonctionnement d'un noyau multitâche pour en appréhender de manière simple les fonctions premières.

1.1 Objet

Cet outil pédagogique permet une simulation graphique du comportement d'un noyau multitâche. Il illustre l'évolution temporelle des différentes tâches ainsi que des composants nécessaires à l'ordonnancement et la synchronisation entre ces tâches (*fig 1*). Il était donc légitime de créer une animation simple à comprendre mais suffisamment fine et exhaustive pour mettre en valeurs les concepts fondamentaux de la programmation concurrente.
A l'heure actuelle, il n'existe pas à notre connaissance d'application équivalente. Les animations dans ce domaine sont plutôt dédiées à la visualisation d'un réseau de Pétri [1]. Si les primitives des noyaux industriels commercialisés [2] sont directement utilisables en langage propre ou même en langage évolué, elles nécessitent cependant une bonne connaissance au préalable de leur impact sur le fonctionnement général. Les outils de mise au point [3] associés sont souvent très complexes à mettre en œuvre par des débutants.

1.2 Contexte général

A l'Université Paris 6, une équipe de recherche travaille dans le cadre du projet CoLoS [4] (Conceptual Learning of Science), projet regroupant depuis 1988 un grand nombre d'universités européennes (et aussi américaines et asiatiques). Les partenaires se rencontrent régulièrement pour mettre en commun outils, logiciels et idées pour concevoir des simulations et/ou des visualisations ayant pour but de faire comprendre des concepts difficiles en utilisant le moins possible les modèles mathématiques. Les applications sont essentiellement dédiées à l'enseignement supérieur et les domaines privilégiés du projet CoLoS sont bien évidemment la physique, la mécanique et l'électricité, mais aussi l'informatique. Des produits ont été ainsi réalisés pour l'illustration de concepts tels que les réseaux de neurones, le partage de ressources, la gestion des processus sous Unix, les algorithmes de routage dans les réseaux….

1.3 Cadre pédagogique

Le public visé est un public spécialisé d'étudiants en informatique, électronique ou encore informatique industrielle. On part trop souvent du principe qu'un étudiant dans ces disciplines n'a pas besoin d'illustration des concepts étudiés et que leur programmation suffira à les lui faire comprendre. Il apparaît pourtant que l'assimilation des bases est souvent noyée dans la correction de quantités d'erreurs syntaxiques ou la prise en main de nouveaux logiciels, et finit par passer au second plan. Il est bien évident que l'application proposée n'a qu'un rôle pédagogique et que son utilisation doit être limitée dans le temps mais un étudiant ayant acquis une solide compréhension des bases indispensables aura toutes chances d'être aussi plus performant en programmation pour son travail de développement.
Cette animation sert à illustrer un cours magistral d'informatique temps réel. Elle offre aussi la possibilité aux étudiants d'expérimenter leurs connaissances lors de travaux pratiques. Elle est actuellement utilisée dans un cours d'informatique sûre et temps réel en 2ème année de Master Sciences de l'Ingénieur, Spécialité Informatique Industrielle et Systèmes Automatisés.

2 LES CONCEPTS ILLUSTRES

Dans un comportement multitâche, il faut d'abord montrer l'ordonnancement. Les tâches prêtes font la queue sur un guichet dans l'ordre de priorité ou, si elles sont au même niveau, dans l'ordre d'arrivée. Seule la tâche en tête de file peut s'exécuter sur le processeur. Son exécution consiste alors à interpréter une à une les lignes du programme associé. L'ensemble des programmes pour chacune des tâches de l'application simulée constitue un scénario dont le comportement peut être analysé pas à pas. Ce comportement est entièrement déterministe au vu du jeu d'instructions disponible. C'est à dire qu'une analyse fine du scénario permet de prédire sans ambiguïté le comportement dans le temps.
Les tâches doivent donc cohabiter pour se partager le processeur unique. Il convient donc de prévoir des commutations entre ces différentes tâches pour illustrer l'impression de parallélisme entre les programmes associés. Le premier moyen d'y arriver est d'affecter un quanta de temps à chacune des tâches qui ont la même priorité. C'est le temps partagé réglé par une instruction fixant le nombre de coups d'horloge du système affectés à chaque tâche. Mais cette méthode est à la limite du déterminisme souhaité en informatique temps réel.

Une approche plus raisonnable consiste pour chaque tâche en concurrence en un passage de la main à un moment prévu par le programmeur à l'aide d'une instruction spécialisée. Ces deux premières méthodes de commutation ne fonctionnent pas pour des tâches de niveau de priorité différente. En effet, dans ce cas, la tâche la plus prioritaire monopolise le processeur et crée la pénurie pour les autres.
Il est préférable d'assurer la commutation entre les tâches à l'aide d'objets de synchronisation mis à disposition par le système. La première solution est un dortoir où chaque tâche peut rester bloquée jusqu'à son réveil à heure fixée laissant la place aux autres tâches. La seconde solution consiste à envoyer ou recevoir des marqueurs ou jetons sur un sémaphore à compte. La réception de jetons est bloquante pour la tâche demandeuse tant que le compte demandé n'y est pas. La situation se débloque grâce à une autre tâche qui envoie au sémaphore le nombre de marqueurs nécessaires. Enfin, la dernière méthode proposée est l'utilisation d'une boite aux lettres pour envoyer un message d'une tâche vers une autre. Là encore, la réception est bloquante pour la tâche demandeuse si la boite est vide et ceci jusqu'à l'arrivée d'un nouveau message.
Le mélange de tous ou certains de ces moyens de synchronisation entre les tâches peut conduire à des situations difficiles à envisager sur le papier, situations pouvant aller jusqu'à un blocage complet de l'application. Seul un simulateur comme le nôtre peut montrer pas à pas le processus de blocage des tâches. Il permet également la recherche de solution à des problèmes de ce type ou plus simplement à un comportement anormal du scénario envisagé.

3 LE LOGICIEL D'ANIMATION

3.1 Scénarios et principes généraux

L'utilisateur peut tester des scénarios écrits dans un pseudo langage de programmation. Une tâche boucle sans fin sauf si elle se suspend définitivement. Chaque scénario possède jusqu'à cinq tâches cycliques s'exécutant en parallélisme sur un processeur unique.
La commutation de tâche est assurée par le noyau. Chaque tâche possède un niveau de priorité qui l'oblige à céder le processeur si une tâche plus prioritaire est activée. Dans le cas de tâches ayant le même niveau de priorité, il est possible d'installer un partage du temps automatique au bout d'un nombre de coups d'horloge fixés par programme. Une tâche peut passer volontairement le contrôle du processeur à la suivante si celle-ci est de même priorité. Les tâches peuvent s'endormir et sont réveillées automatiquement lorsque le temps demandé est écoulé.
La synchronisation entre les tâches peut s'effectuer par sémaphore ou boite aux lettres. On dispose d'une réserve de ressources allant jusqu'à six boites et huit sémaphores à compte. Chaque tâche peut envoyer ou recevoir des messages sur la boite de son choix. Elle peut de même envoyer ou recevoir un nombre quelconque de marqueurs sur le sémaphore de son choix.

3.2 Visualisation graphique

La fenêtre graphique permet de visualiser au fil de l'eau l'état interne du noyau multitâche. Le code source de chacune des tâches est montré, avec un curseur sur la dernière instruction exécutée (*fig 2*). Un guichet montre les tâches actives en attente du processeur dans l'ordre de priorité et un dortoir montre la queue des tâches endormies avec le temps restant pour chacune (*fig 3*). On visualise la tâche en cours d'exécution dans le processeur ainsi que l'état de l'horloge interne (*fig 4*). Les boites aux lettres montrent la queue des tâches en attente de courrier ou la file des messages non lus (*fig 5*). Les sémaphores montrent les tâches en attente avec le nombre de marqueurs demandés pour chacune ainsi que les marqueurs non consommés (*fig 6*).

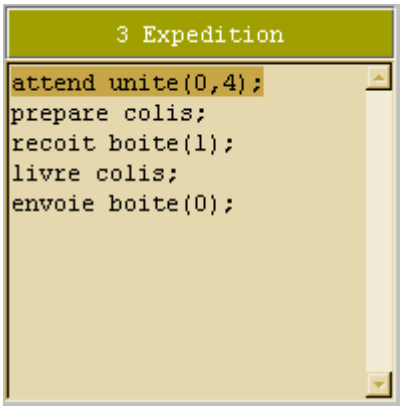

fig 2 : suivi d'une tâche

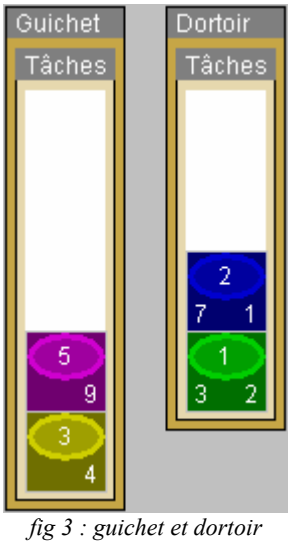

fig 3 : guichet et dortoir

fig 4 : processeur et horloge

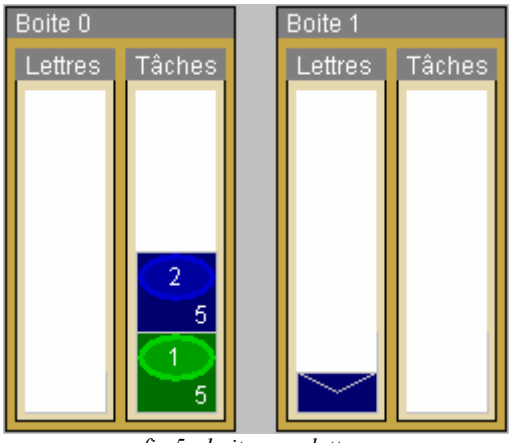

fig 5 : boites aux lettres

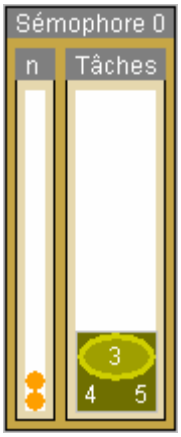

fig 6 : sémaphore

3.3 Programmation

L'écriture d'un scénario doit s'effectuer dans un éditeur non fourni. L'éditeur peut rester ouvert pendant une simulation pour autoriser des modifications sans sortir de l'application. Il faut néanmoins recharger la nouvelle version du scénario pour la prise en compte des modifications.
Le langage de programmation est rustique et comprend les instructions pour créer et manipuler les tâches ainsi que les ressources de base. Aucune structure alternative ou de boucle n'est prévue. En dehors de ce jeu d'instructions élémentaires, l'utilisateur peut écrire des instructions libres pour illustrer les autres activités de la tâche.

3.4 Jeu d'instructions

Les instructions interprétées par le simulateur ont une syntaxe stricte. Toute ligne de programme ne respectant pas cette syntaxe est considérée comme une instruction libre de l'utilisateur dont l'exécution prend un temps de cycle de l'horloge.
La tâche est constituée d'une séquence d'instructions qui seront rejouées en boucle tant que la tâche n'est pas suspendue. Chaque instruction se termine par un point-virgule. La première tâche d'un scénario possède la priorité maximale et sert en général de constructeur pour les autres tâches et les ressources partagées.

"[nomTache]"
: Séparateur indiquant le début d'une tâche

"cree tache(niveauPriorite, nomTache);"
: Construit et lance une nouvelle tâche.

"cree partage(période);"
: Autorise le temps partagé au même niveau.

"cree dortoir;"
: Autorise l'endormissement des tâches.

"cree boite(numero);"
: Construit une boite aux lettres vide.

"cree semaphore(numero);"
: Construit un sémaphore à compte vide.

Chaque tâche utilise le noyau et ses ressources pour se synchroniser avec les autres tâches. Elle peut exécuter des lignes d'instructions libres consommant un coup d'horloge système.

"se suspend;"
: La tâche s'arrête définitivement.

"passe;"
: Donne la main à sa suivante de même niveau.

"dort(durée);"
: S'endort pour la durée indiquée.

"envoie boite(numero);"
: Envoie un message sur la boite désignée.

"recoit boite(numero);"
: Attend un message sur la boite désignée.

"envoie unites(numero,unites);"
: Envoie des unités au sémaphore désigné.

"attend unites(numero,unites);"
: Attend des unités sur le sémaphore désigné.

3.5 Options d'exécution

Chaque instruction d'une tâche occupe le processeur pour une unité d'horloge. La vitesse de la simulation est réglable en marche à l'aide d'un curseur et peut aller jusqu'au pas à pas. Le scénario peut être repris depuis le début ou être remplacé par un autre sélectionné par l'utilisateur (*fig 7*).

fig 7 : bandeau de commande

3.6 Vision comportementale

Le logiciel propose deux modes de fonctionnement. Le mode de marche automatique montre le fonctionnement global et concurrent des tâches du scénario. Le paramètre temporel est respecté. On ne voit pas le détail des opérations internes du noyau. Chaque instruction du programme occupe un coup d'horloge du système. La vitesse de l'horloge est réglable en marche permettant de visualiser de longues périodes de simulation sur un temps d'exécution très court. On a alors une vue macroscopique du comportement multitâche du scénario retenu. Cette vue sur une échelle de temps assez longue révèle des comportements d'ensemble pas toujours prévisibles au moment de l'écriture du programme. A l'opposé, le mode de fonctionnement en

pas à pas décompose un top d'horloge en plusieurs opérations montrant en détail l'ordonnancement interne des tâches. On voit les échanges entre les ressources et le processeur, les mouvements de tâche à l'intérieur des files d'attente, les commutations de tâches et l'avancement du programme dans la tâche finalement élue. Dans ce mode, le timing de l'application n'est plus respecté. Lorsque l'on a bien compris le comportement du système, tout ce qui va se passer dans un pas de simulation est entièrement prédictible, donc entièrement déterministe.
Le logiciel permet soit en démonstration de cours, soit en travaux pratiques d'analyser finement le comportement microscopique d'un scénario particulier. C'est surtout l'aspect du comportement sur des longues périodes que les étudiants ont du mal à cerner.
L'expérimentation de solutions avec ce logiciel sera alors un excellent exercice préalable à toute programmation concurrente.

3.7 Configuration matérielle

Les premières applications réalisées dans le projet CoLoS ont été écrites en langage C dans l'environnement XWindows et ne tournaient que sous Unix.
Le langage Java apparu en 1995 a semblé tout à fait adéquat pour ce genre d'applications. Sa portabilité sur toutes les plate-formes et sa connexion à Internet sous forme d'applets en sont les points forts.
Le logiciel est écrit en Java. Il nécessite donc une plate-forme équipée d'une machine virtuelle java acceptant la bibliothèque "javax.swing". D'autre part, la fenêtre de visualisation nécessite une résolution d'écran d'au moins 1024 * 768 pixels pour avoir une vue globale.
Le logiciel a été testé sous Windows® 2000 et XP de Microsoft® et sous Linux® avec Java 2 SDK, Standard Edition Version 1.4.2 de Sun®.
La livraison comprend un répertoire contenant les fichiers pré compilés "*.class" et une bibliothèque de scénarios de démonstration. Un simple éditeur de texte source externe est nécessaire pour saisir de nouveaux scénarios. Une notice d'utilisation et une documentation sur les scénarios proposés sont également fournies.

3.8 Test et améliorations

Le logiciel a été testé à plusieurs reprises par des enseignants en cours et des étudiants lors de travaux pratiques. Les remarques des uns et des autres ont permis une amélioration sur l'ergonomie et la qualité de la simulation. C'est donc une version améliorée qui est proposée ici. Certaines remarques furent contradictoires et il fallut effectuer des choix stratégiques. Il manque la possibilité de tracer, sous forme de diagramme temporel, le déroulement d'un scénario. On y verrait l'état de chaque tâche et les évènements de synchronisation responsables des commutations. Cette vision temporelle serait visible au fil de l'eau pendant le déroulement de la simulation ou plus tard lors d'un dépouillement hors ligne. Mais cette amélioration constituera une étude logicielle à part entière déjà en cours d'élaboration.

4 CONCLUSION

Ce logiciel offre une animation graphique du déroulement d'un scénario construit dans un environnement multitâche utilisant les concepts couramment rencontrés sur les systèmes d'exploitation embarqués du marché. Il n'y a aucun besoin d'une connaissance d'un langage évolué ou spécifique. Les exercices de travaux pratiques ne sont donc pas « pollués » par des problèmes de syntaxe ou de difficulté d'apprentissage du logiciel ce qui permet de se concentrer sur les difficultés rencontrées en programmation multitâche. L'étudiant peut alors se concentrer sur les concepts fondamentaux ce qui lui permet d'expérimenter les différentes situations telles que les blocages, la pénurie de ressources, les saturations de files d'attentes ou simplement une mauvaise répartition du temps alloué au processeur.

Bibliographie

1. *International Conferences on Application and Theory of Petri Nets*
   University of Aarhus, Denmark
   http://www.daimi.au.dk/PetriNets/.
2. *Embedded Systems Programming magazine*
   http://www.embedded.com/
3. *Tornado II® for VxWorks®*
   Wind River
   http://www.windriver.com/
4. *CoLoS (Conceptual Learning of Science)*
   http://www.colos.org/