Rapport du TP de Systèmes d'Exploitation Multitâches: Le Festival

Vincente NATTA & Salomé COAVOUX

21 novembre 2013

Table des matières

1 Version 0

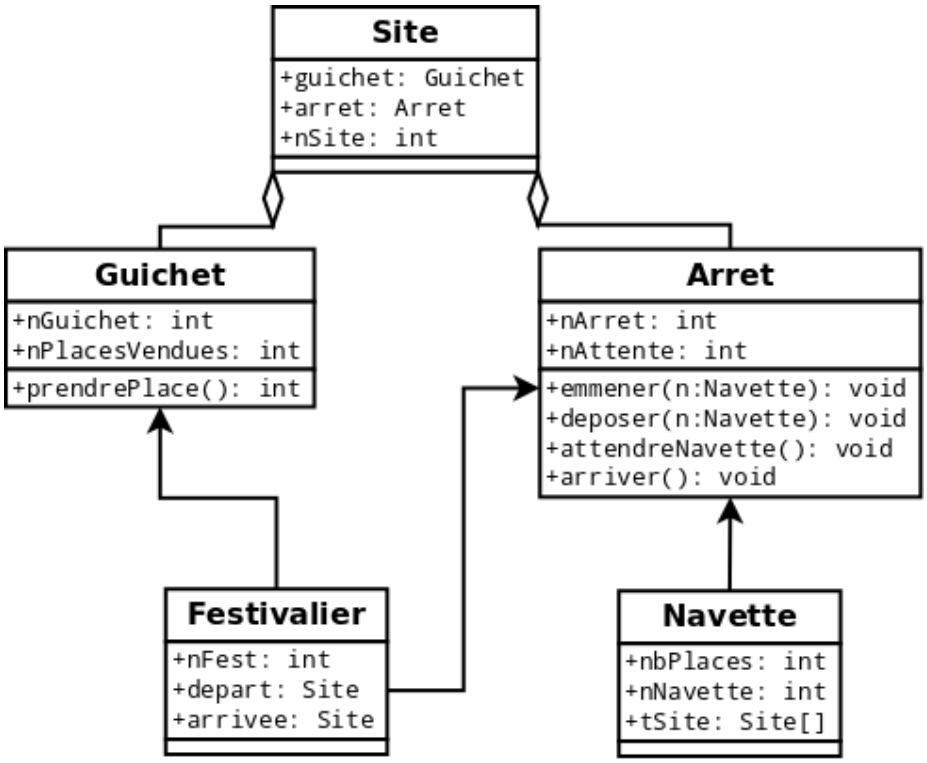

FIGURE 1 – Diagramme de classes de la version 0

Cette version est une amorce de la première version. En effet, c'est une simulation correcte du comportement de tous les *Threads* (*Festivaliers* et *Navettes*), et qui agit exactement comme le fait la version 1. Cependant, bien que les objets partagés soient les *Arrêts* et les *Guichets*, un seul moniteur est nécessaire ici, sur le dernier *Arrêt* (celui du *Festival*).

Pour réaliser cette version, nous avons détourné l'utilisation des moniteurs. Ainsi, après avoir acheté un billet, chaque *Festivalier* s'endort à l'*Arrêt* du *Festival*. Quand une *Navette* arrive, elle "prend" autant de *Festivaliers* que possible en réduisant leur nombre à l'*Arrêt*, et le nombre de places dans la *Navette*. Quand la *Navette* arrive au dernier *Site*, elle notifie au hasard autant de *Festivaliers* qu'il y avait de places prises dans la *Navette*.

L'avantage de cette implémentation est la simplicité du code : on n'utilise pas de tableaux pour savoir quel *Festivalier* est dans quelle *Navette*. L'inconvénient principal est cependant qu'il est très difficile de passer de cette version à la version finale, puisque tous les *Threads* fonctionnent indépendamment les uns des autres. Cette version n'est fonctionnelle que parce que seuls les *Festivaliers* utilisent un moniteur, et qu'ils sont forcés de l'utiliser : il n'y a aucune condition pour s'arrêter, et donc aucune condition pour retourner dormir si ce n'était pas lui qu'il fallait réveiller. La contrainte d'une seule *Navette* par *Arrêt*, par exemple, n'aurait pas pu être implémentée avec des moniteurs, puisque le fait de notifier une *Navette* en attente aurait réveillé aussi les *Festivaliers* endormis à l'*Arrêt*.

Cette version étant très simple, un seul problème de synchronisation, qui était plutôt une erreur d'inattention, s'est posé. Au début, les *Festivaliers* n'avaient pas d'attribut *arrivée*, qui représente le *Festival*. Ainsi, ils attendaient sur leur *Site* de départ respectifs, et étaient réveillés sur le dernier *Site*. Le programme tournait donc en boucle, puisque le moniteur où s'effectuait l'attente et celui où s'effectuait la notification n'étaient pas les mêmes.

2 Version 1

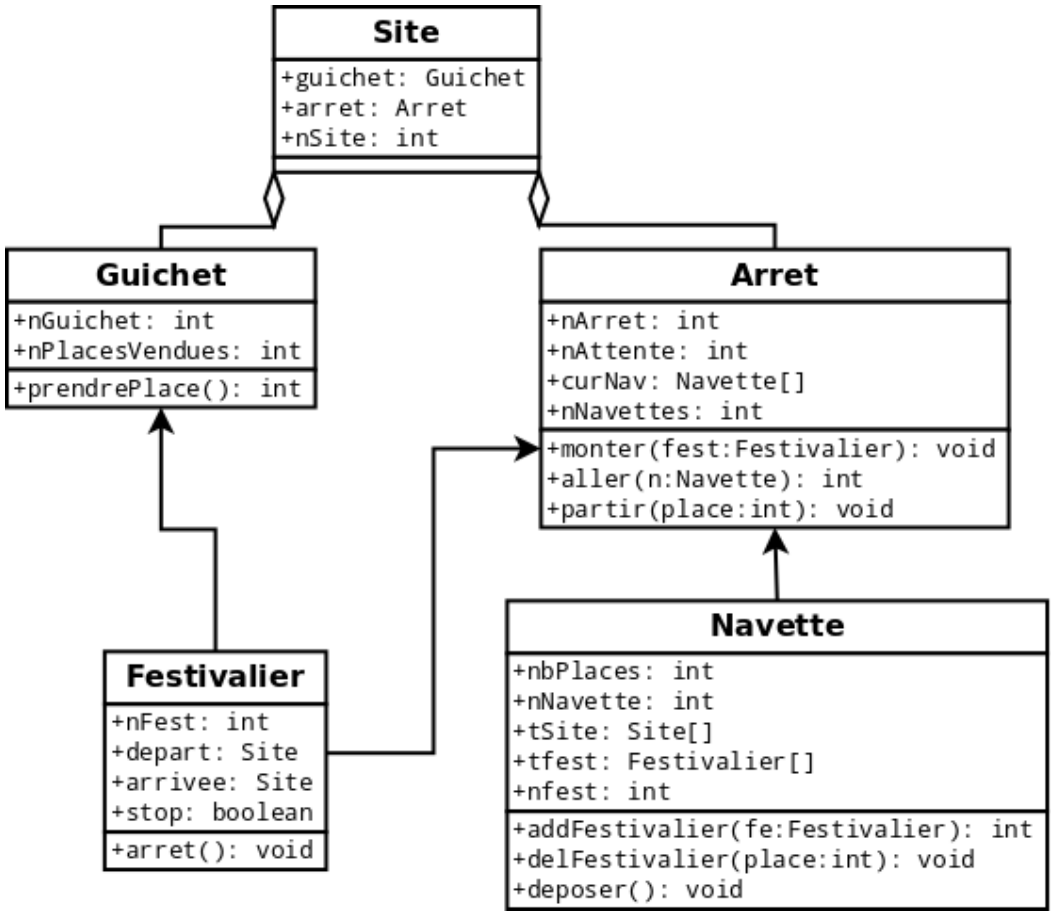

FIGURE 2 – Diagramme de classes de la version 1

Pour implémenter une première version correcte, nous avons ajouté aux *Arrêts* un tableau des *Navettes* courantes, et aux *Navettes* un tableau des *Festivaliers* présents. On ajoute alors aux objets partagés les *Navettes*, en plus des *Arrêts* et des *Guichets*.

Ainsi, une fois leur billet acheté, les *Festivaliers* essaient de monter dans une navette courante en cherchant une place libre dans le tableau des *Navettes*. Ensuite, grâce à un attribut booléen dans la classe *Festivalier*, celui-ci s'endort régulièrement jusqu'à ce que la *Navette* dans laquelle il est change la valeur de ce booléen. De cette manière, aucun *Thread Festivalier* ne s'arrête avant son arrivée au *Festival*.

Le plus gros problème pour cette version a été de trouver les conditions d'arrêt pour les *Festivaliers*. Quand un *Festivalier* veut monter dans une *Navette*, il doit d'abord vérifier qu'une *Navette* attend à l'*Arrêt*, puis parcourir toutes les *Navettes* pour trouver une place libre. Ce problème a été réglé grâce à l'ajout d'un booléen *in* dans la méthode *monter()*, qui permet de savoir si le *Festivalier* a trouvé ou non une place. De cette manière, les *Festivaliers* attendent à un *Arrêt* tant qu'il n'y a aucune *Navette*, ou qu'ils ne trouvent pas de place dans celles-ci.

3 Version 2

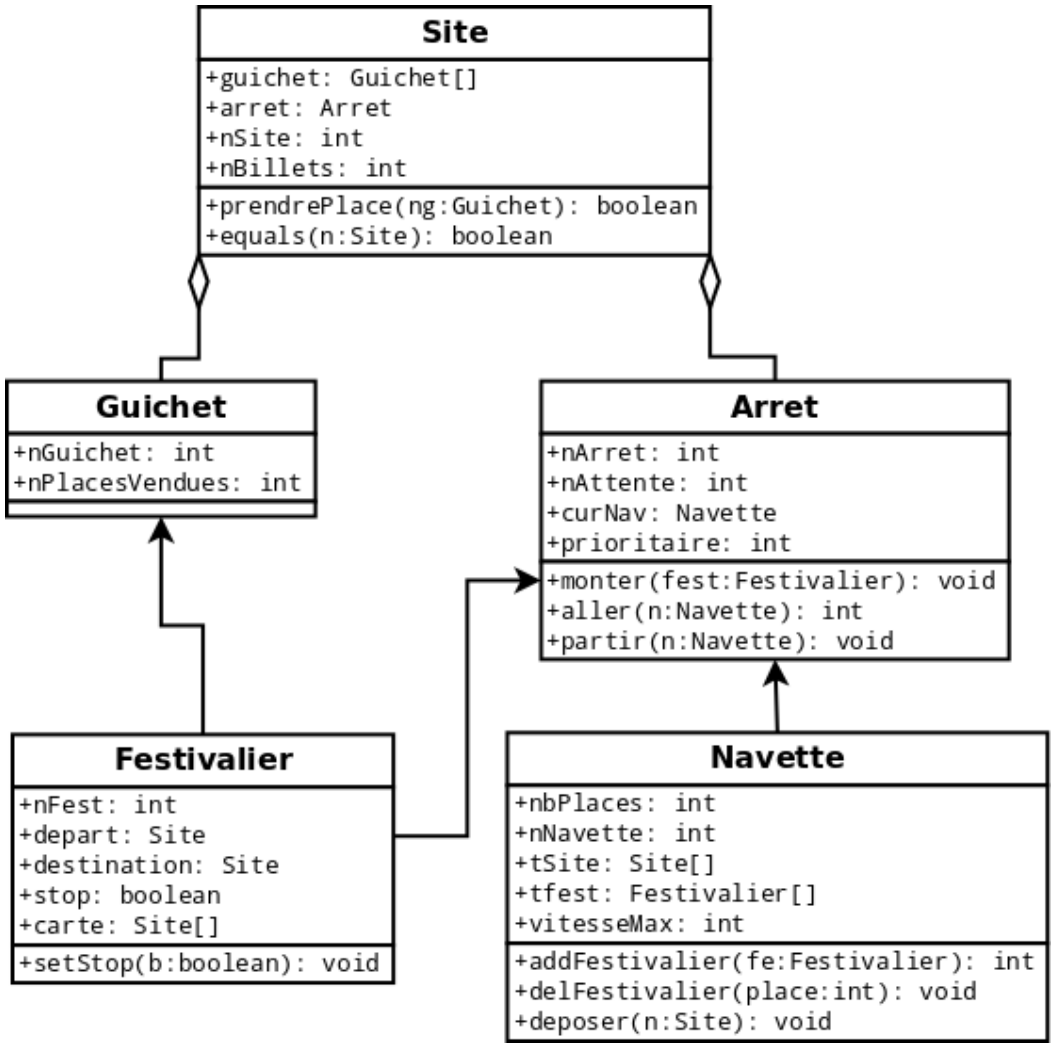

FIGURE 3 – Diagramme de classes de la version 2

La version 2 a été dans un sens plus facile à réaliser. La contrainte d'une seule *Navette* à l'*Arrêt* à la fois, notamment, a grandement simplifié le code. En effet, pour satisfaire cette hypothèse, il suffit de transformer le tableau de *Navettes*, attribut des *Arrêt*, en une seule *Navette* courante. Cela évite ainsi le parcours du tableau quand un *Festivalier* cherche à monter dans une *Navette*.

De même, autoriser plusieurs *Guichets* par *Site* n'a consister qu'à changer l'attribut *Guichet* des *Sites* en un tableau de *Guichets*. Les *Festivaliers* choisissent alors un *Guichet* au hasard. Le nombre limité de billets a entraîné le transfert de la méthode *prendrePlace* dans la classe *Site* afin d'accéder au nombre des billets accessibles sur ce *Site*.

En revanche, afin de rendre la simulation plus réaliste, un *Festivalier* n'ayant pu avoir son billet sur le *Site* de départ décide de se rendre en *Navette* jusqu'au *Site* suivant pour essayer d'acheter son billet. S'il n'y a plus de billet nulle part, le *Festivalier* parcourt tous les *Sites*, puis, revenu au *Site* de départ, décide de rentrer chez lui.

Le problème du service de ramassage a été réglé grâce à un attribut *prioritaire* ajouté aux *Arrêts*. En effet, chaque *Navette* a maintenant une vitesse propre. La synchronisation aurait donc pu être réalisée avec

un sémaphore dont la file d'attente est de type *FIFO*. Cependant, cela aurait nécessité une harmonisation des vitesse sur la vitesse la plus basse, et de modifier notre implémentation pour être sûr de garder l'ordre des *Navettes*. Le code aurait alors été similaire au problème du *Producteur/Consommateur* vu en TD. Nous avons donc préféré garder les moniteurs et ajouter une contrainte d'arrêt aux *Navettes*.

Les *Navettes* commencent donc maintenant toutes au même *Arrêt*, et se suivent dans l'ordre de 0 à n. La *Navette* 0 a priorité partout, c'est donc à elle de débuter le service de ramassage. Avant de partir d'un *Arrêt*, chaque *Navette* n donne la priorité à la *Navette* n+1. Ainsi, concrètement, les *Navettes* peuvent se doubler sur le chemin entre deux *Arrêts*. Cependant, une *Navette* ayant doublé ne pourra aller à l'*Arrêt* que lorsque la *Navette* doublée sera arrivée et repartie.

Cette version n'est donc plus tout à fait réaliste, mais permet quand même une évaluation du temps d'exécution de la simulation en fonction des paramètres (par exemple le nombre de *Navettes* ou leur vitesse).

4 Etude du service de ramassage

Pour les mêmes paramètres, deux simulations peuvent avoir un temps d'exécution très différent. Cela est dû à l'utilisation de données aléatoires, comme le nombre de billets par *Site* ou le nombre de *Festivaliers* au départ d'un certain *Site*. Pour cette raison, seuls le nombre de *Navettes*, leur vitesse maximale et minimale ont été changés d'une simulation à l'autre. De plus, chaque ensemble de paramètres a été testé une vingtaine de fois afin d'avoir des valeurs cohérentes.

A titre indicatif, chaque simulation s'est passée sur 7 *Sites*, avec 100 *Festivaliers*, le nombre de *Guichets* sur chaque *Site* varie entre 1 et 13, et le nombre de billets disponibles par *Site* entre 1 et 50. Les valeurs sont données en millisecondes. Le tableau complet des valeurs est disponible en annexe.

Navettes	Vitesse min	Vitesse max	Minimum	Maximum	Moyenne	Médiane basse	Médiane haute
3	30	90	16206	26772	19828,8	18723	18924
6	30	90	16175	25327	18247,75	17705	17718
10	30	90	17739	32534	20908,05	19129	19832
6	80	90	15796	26220	18485,15	17567	17614
3	80	90	15837	24399	18677,65	17686	17703
3	30	50	18090	25873	20255,05	19747	19940
4	50	90	15895	22584	17860,9	17514	17551
1	110	110	23280	31387	25330,35	23333	23336

Nous pouvons bien voir ici que les meilleurs résultats sont obtenus avec un nombre de *Navettes* inférieur au nombre des *Sites*, à vitesses élevées, et avec une fourchette de vitesses assez petite. En effet, il suffit d'une seule *Navette* avec une faible vitesse pour ralentir l'intégralité du *Festival*. Par exemple, une configuration de 4 *Navettes* avec une vitesse de 50 à 90 donne de très bons résultats par rapport aux autres configurations, pour un *Festival* étendu sur 7 *Sites*.

5 Annexes

Navettes	3	6	10	6	3	3	4	1
Vitesse min	30	30	30	80	80	30	50	110
Vitesse max	90	90	90	90	90	50	90	110
	16206	16175	17739	15796	15837	18090	15895	23280
	16564	16365	17765	15847	15903	18309	16024	23280
	16773	16424	18034	15884	15969	18426	16356	23281
	17723	16906	18198	15890	15971	18940	16485	23284
	18432	16969	18252	15990	15997	18941	16896	23291
	18442	17026	18424	16060	16073	19089	17092	23295
	18446	17097	18493	16093	17592	19094	17153	23298
	18587	17563	18827	17454	17629	19261	17451	23314
	18671	17619	18935	17522	17642	19428	17498	23329
	18723	17705	19129	17567	17686	19747	17514	23333
	18924	17718	19832	17614	17703	19940	17551	23336
	19038	17943	20048	17630	17709	19941	17603	23337
	19129	18200	20843	17673	17743	20598	17621	23345
	19263	18201	20866	17693	17795	20656	17869	23346
	20614	18506	21255	17721	17815	20825	18220	23347
	21090	19405	22283	17753	21290	20826	18238	31375
	21515	19764	22724	24281	24141	21269	18485	31382
	25581	19860	24943	24480	24310	22191	19132	31382
	26083	20182	29037	24535	24349	23657	21551	31385
	26772	25327	32534	26220	24399	25873	22584	31387
Minimum	16206	16175	17739	15796	15837	18090	15895	23280
Maximum	26772	25327	32534	26220	24399	25873	22584	31387
Moyenne	19828,8	18247,75	20908,05	18485,15	18677,65	20255,05	17860,9	25330,35
Médiane basse	18723	17705	19129	17567	17686	19747	17514	23333
Médiane haute	18924	17718	19832	17614	17703	19940	17551	23336

FIGURE 4 – Résultats de l'analyse du service de ramassage