IN405 – Système d'exploitation
TD 8 – Exercices avancés

S. Gougeaud

2017/2018

Exercice 1 – Passage de jeton dans un réseau distribué

L'objectif de l'exercice est de créer un réseau de processus indépendants où la communication est assurée par des tubes. Un jeton, possédant une durée de vie définie, parcourt ce réseau jusqu'à que ce que sa vie se termine. Vous pouvez supposer le réseau présenté figure 1 pour vos tests.

L'exercice s'effectue en plusieurs étapes :

1) Construction du graphe

Chaque noeud du graphe correspond à un flux d'exécution d'un programme, et chaque arête à l'existence d'un moyen de communication bi-directionnel entre ces deux flux. Implémentez le programme correspondant à ce graphe à l'aide de processus pour les flux d'exécution et de tubes anonymes pour les moyens de communication. *On dit que le noeud 0 est le processus père du programme.*

2) Travail d'un noeud

Pour exécuter une instruction, un noeud du graphe doit détenir un jeton. L'instruction en question est l'affichage de la phrase "Je suis i et je possède le jeton !" dans laquelle i est le numéro du noeud. Une fois l'instruction effectuée, le jeton est envoyé aléatoirement à un autre noeud. Implémentez la fonction qui se met en attente d'un jeton, fait l'affichage puis renvoie le jeton.

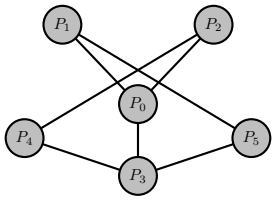

Figure 1: Réseau distribué test

3) Initialisation et mise à jour du jeton

On suppose que le jeton est détenu au début du programme par le noeud 0, et qu'il possède une durée de vie égale à 20. A chaque fois que le jeton est utilisé (ie. qu'une instruction est effectuée par le noeud possédant le jeton), sa durée de vie décrémente d'un point. Implémentez l'initialisation du jeton par le processus père, et la mise à jour du jeton après chaque affichage réalisé.

4) Fin de vie du jeton et destruction du graphe

Lorsque la durée de vie du jeton est nulle, le programme s'arrête. Il faut alors prévenir l'ensemble des noeuds du graphe qu'il faut terminer l'exécution. A cet effet, le noeud recevant le jeton dont la durée de vie est nulle, prévient tous les noeuds qu'il a dans son entourage que le programme se termine (par un message contenant l'entier -1 par exemple). A la réception de ce message, chaque noeud préviendra à son tour tous les noeuds de son entourage, ainsi de suite jusqu'à ce que tous les noeuds aient été prévenus et soient terminés. Implémentez ce phénomène.

Exercice 2 – Chasse au trésor dans un fichier partagé

L'objectif de l'exercice est de créer un jeu dans lequel un chasseur de trésor creuse à différents endroits d'une île afin d'emplir ses poches de richesse. Le programme est composé de trois acteurs différents :

- L'île, qui connaît la topographie des lieux et sait où se trouve le trésor.
- Le chasseur, qui cherche à récupérer le trésor en creusant aléatoirement (ou non ?) sur l'île.
- La volcan, qui peut à tout moment détruire l'île, son trésor et ses occupants s'il y en a.

L'appel au programme se fait par l'appel suivant :

```
$ ./hunt [i/h/v] [arg1] [arg2]
```

L'argument `[i/h/v]` détermine le rôle d'acteur du processus créé, entre l'île (island), le chasseur (hunter) et le volcan (volcano). Les arguments `arg1` et `arg2` dépendent de l'acteur créé. La suite de l'énoncé décrit le comportement de chaque acteur.

1) L'île – *The island*

L'île est constituée de $m \times n$ cases où chaque case est l'un des éléments indiquée dans la table 1. La constitution initiale de la carte se fait à la création du processus, où une case possède une probabilité p d'être de l'élément indiqué. Creuser une case pour chercher le trésor demande un apport e en énergie devant être fournie par le chasseur. La topographie de la carte est seulement connue de l'île.

type	p	e
eau	10	1
sable	50	1
roche	10	3
palmier	15	2
épave	10	5
volcan	5	4

Table 1: Paramètres de l'île

L'île partage avec le chasseur un plan sommaire où sont référencées les cases ayant déjà été creusées. Ainsi, une case ne peut pas être creusée plus d'une fois.

L'emplacement du trésor est défini à la création du processus, mais ne peut pas se trouver sur une case d'eau ou de volcan. Si le trésor a été trouvé par le chasseur ou si le volcan entre en éruption, alors l'île s'éteint.

L'appel au programme pour la création de l'île est : `$ ./hunt i m n`

2) Le chasseur – *The hunter*

Le chasseur possède une énergie initiale E. Il demande à son initialisation la taille de l'île et commence à creuser là où des recherches n'ont pas encore été effectuées. S'il trouve le trésor, alors il repart victorieux. Si un autre chasseur trouve le trésor avant lui, alors il repart défait. Si son énergie tombe à 0, alors il meurt d'épuisement. Si le volcan entre en éruption, alors il meurt tout court.

L'appel au programme pour la création du chasseur est : `$ ./hunt h E`

3) Le volcan – *The volcano*

Le volcan a pour seule vocation d'entrer en éruption. Lorsqu'il est créé, l'île et les chasseurs sont détruits, brûlés au 42ème dégré.

L'appel au programme pour la création du volcan est : `$ ./hunt v`

4) Consignes d'implémentation

- La topographie de la carte est uniquement possédée par l'île.
- La carte sommaire, indiquant quelles cases ont déjà été creusées est partagée en mémoire dans un fichier. Pour ceci, aidez-vous de l'exercice sur `mmap()` du TD 3.
- La communication demandée est bi-directionnelle : dans un premier temps, le chasseur indique quelle case il cible, puis l'île lui répond en lui donnant le montant d'énergie qu'il a perdu et/ou s'il a trouvé le trésor. Il faut donc au moins un tube pour les messages à destination de l'île, et un autre tube pour les messages à destination d'**un** chasseur.

- Pour permettre à l'île de 'tuer' les chasseurs se trouvant en son sein lorsque le volcan entre en éruption, il est préférable de garder en mémoire une liste des chasseurs.