# LSINF1252 - Factorisation de nombres

Monnoyer Charles et Paris Antoine

21 avril 2015

## 1 Architecture globale

Pour structurer une application qui réalise des calculs, il est courant d'utiliser des producteurs/consommateurs[1]. Pour cette application, il est peut-être même plus intéressant d'utiliser deux buffers et donc d'utiliser des producteurs/ consommateurs-producteurs/consommateurs.

Dans notre cas, les producteurs seraient chargé d'extraire les nombres à factoriser des fichiers passés en ligne de commande et de les placer dans le premier buffer. Les consommateurs-producteurs seraient quant à eux chargé de factoriser les nombres contenus dans le premier buffer pour ensuite insérer les facteurs premiers de ces nombres dans le second buffer. Enfin, les consommateurs permettrait de vider le second buffer pour stocker les facteurs premiers dans une structure données (qui sera discutée en section 4.

## 2 Threads utilisés

Pour bien fonctionner, notre application nécessite donc aux minimum 3 threads : un extracteur, un calculateur et un sauveur de résultats.

## 3 Mécanismes de synchronisation

Pour implémenter un simple problème de producteurs- consommateurs, il faut utiliser un mutex pour protéger le buffer et deux sémaphores pour gérer les producteurs/ consommateurs. Dans ce cas-ci, il faudra utiliser deux mutex pour protéger les deux buffers et quatre sémaphores.

## 4 Principale structures de données

Les deux buffers seront représentés par des tableaux d'entiers. Les facteurs premiers seront ensuite déplacé du second buffer vers une liste châinée de structure. Ces structures contiendront un facteur premier et le nombre d'occurence associé à ce facteur.

## 5 Algorithme de factorisation

L'algorithme que nous avons décidé d'implémenter pour ce projet est un algorithme à but général (c'est à dire dont le temps d'éxécution dépend de la taille du nombre à factoriser, et non de la taille de ces facteurs premiers). Il s'agit du *Shanks's square forms factorization algorithm (SQUFOF)*. Nous avons choisis cet algorithme car il possède une bonne complexité temporelle ($\sqrt[4]{n}$, où $n$ est le nombre à factorisé) tout en étant facile à implémenter. Deux contraintes importantes sont cependant à noter, cet algorithme ne fonctionne pas si son entrée $n$ est un carré parfait ou un nombre premier. Cependant, cela ne posera pas problème en pratique. En effet, dans le cas où $n$ est un carré parfait, il suffit de donner $\sqrt{n}$ en entrée à l'algorithme [1] Dans le cas où $n$ est un nombre premier, l'algorithme est inutile et il peut simplement retourner $n$.

## Références

[1] O. Bonaventure G. Detal C. Paasch. *SINF1252 : Systèmes informatiques*. EPL, 2014.

1. Les facteurs premiers de $n$ sont identiques aux facteurs premiers de $\sqrt{n}$.