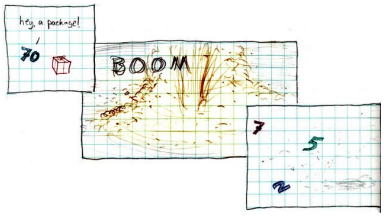
TP n° 3: Crible partagé

Exercice 1.

Blown into prime factors



source : xkcd

Le but de cet exercice est de calculer tous les nombres premiers inférieurs à n, n étant un argument du programme. La méthode choisie est le *Crible d'Ératosthène*, qui est une méthode qui a l'avantage d'être simple et de pouvoir être parallélisée sans aucun mécanisme de synchronisation mémoire.
On représentera le crible par un tableau unidimensionnel de `char`. Soit c le crible, et c_i la valeur du crible pour la *i-ème* case (i.e., `c[i]` en C). Si $c_i = 0$ alors i est premier, sinon si $c_i = 1$ alors i n'est pas premier. Au début, $\forall i,\ c_i = 0$.
Soit p le nombre de processus qui met à jour en parallèle le crible, on a $p < n$ (on rappelle que n est la taille du crible).
Comme on veut pouvoir paralléliser le processus, on doit mettre en place un mécanisme de communication inter-processus. Dans notre cas via `mmap`. Le crible sera stocké dans une zone mémoire créée à l'aide de `mmap`. Bien sûr, aucune question ne sera tolérée sans avoir lu au préalable `man 2 mmap` ☺.

Dans la suite, les fonctions retournant un `int` renvoient un *code de retour*, ici 1 si tout s'est bien passé, et 0 en cas d'erreurs.

1. Dans un premier temps on fixe $p = 0$, c'est-à-dire que le père va créer un crible à l'état initial via `mmap`, puis calculer lui-même le crible.
   - **(a)** Créer une zone mémoire partagée grâce à `mmap` qui représentera le crible.
   - **(b)** Écrire une fonction `int init(char* crible, int n)` qui initialise le crible.
   - **(c)** Écrire une fonction `int update_k(char* crible, int n, int k)` qui prend un pointeur sur le crible et qui met à 1 les multiples de k dans le crible. Cette fonction sera appelée par le père sur le crible pour $2 \leq k \leq \sqrt{n}$.
   - **(d)** Écrire une fonction `int affiche(char* crible, int n)` qui affiche les nombres premiers inférieurs à n.
   - **(e)** Écrire une fonction `int libere(char* crible, int n)` qui libère la mémoire occupée par le crible.
   - **(f)** Résoudre le crible.
2. On peut maintenant résoudre le crible avec un processus. L'un des (nombreux) intérêts de `mmap` est de pouvoir partager la zone mémoire allouée, on voudrait pouvoir définir un nombre p arbitraire (avec $p < n$) et donc pouvoir faire travailler p fils sur le crible afin d'améliorer les performances. Maintenant le père ne va plus travailler sur le crible, il va seulement initialiser le crible puis attendre ses fils (modèle *maitre/esclave* ou *fork/join*). C'est toujours au père d'afficher la liste des nombre premiers, à la fin.

   Note : Vous n'avez pas besoin de mécanisme de synchronisation ici (on ne cherche pas à faire un crible hautement efficace).
   - **(a)** Comment diviser le travail à faire par les fils ?
   - **(b)** Résoudre le crible avec 1 fils.
   - **(c)** Résoudre le crible avec p fils.
   - **(d)** *(Optionnelle)* Quelles sont les «bonnes» valeurs pour p en fonction de n ?

3. Maintenant on veut utiliser des signaux [1] pour avertir le père de l'activité des fils et ne plus utiliser des fonctions comme `wait`.

   (a) Étudiez la structure `struct sigaction` et son champ `sa_handler` définie dans `signal.h` ainsi que la fonction `int sigaction(.......);`.

   (b) Les fils doivent envoyer le signal `SIGCHLD` au père quand ils ont fini.
   Écrivez une fonction `void handler(int signal)` qui sera appelé par le père quand il recevra un signal `SIGCHLD`.

   (c) Pour éviter les zombies, le père doit tuer lui-même tous les fils qui ont terminé le travail avec le signal approprié.

4. *(Optionnelle)* Pour le moment, si un nombre n'est pas premier on le test quand même. Il pourrait être intéressant de regarder si $c_i = 1$ (i.e., c_i n'est pas premier) avant de tester c_i.

   (a) En utilisant les primitives de synchronisation appropriés, testez si la valeur a déjà été testé par un autre processus (i.e., si la valeur est à 1).

   (b) Quel est l'impact sur les performances ?

1. `http://man7.org/linux/man-pages/man7/signal.7.html`