Exercice 5

Difficulté : 130 points-virgules

On désire réaliser un programme pour réaliser le calcul du polynôme $P(x) = \sum_{i=0}^{i=n} a_i x^i$ pour toutes les valeurs de x comprises entre 1 et k (on ne considère que des entiers dans cet exercice). Le calcul direct étant trop simple, on souhaite utiliser un fichier, des processus, des tubes et des signaux.

`poly` *k f* a_0 `...` a_n

La méthode que l'on demande d'utiliser est la suivante :

1. le processus père génère $n+1$ processus fils (c_0 à c_n) et écrit dans le fichier de nom f les valeurs n, i (initialement 0), x (initialement 1), p (initialement 0), puis les identificateurs des processus c_1 à c_n et enfin son propre identificateur de processus ;
2. au moment de sa création, le processus père communique à chaque fils le descripteur d'ouverture du fichier f et le coefficient a_i (à l'exclusion de toute autre valeur) ;
3. le processus fils attend l'arrivée d'un signal `SIGUSR1` ou `SIGUSR2` ;
4. lorsqu'un fils reçoit le signal `SIGUSR1`, il lit les valeurs n, i, x et p dans le fichier, calcule $a_i x^i$ en utilisant la commande `expr` (voir plus bas) et l'ajoute à la valeur de p ;
5. puis le fils réécrit les nouvelles valeurs $n+1$ et p (vous pouvez réécrire les 4 pour simplifier) ;
6. enfin, le fils envoie le signal `SIGUSR1` au processus suivant $n+1$ ou au père, puis il se remet en attente de réception d'un nouveau signal ;
7. lorsqu'un fils reçoit le signal `SIGUSR2`, il se termine avec un code de retour nul ;
8. lorsque le père reçoit le signal `SIGUSR1`, il récupère dans le fichier la valeur p (qui est maintenant égale à $P(x)$, l'affiche, puis commence le cycle de calcul de $P(x+1)$ tant que $x < k$.
9. lorsque toutes les valeurs de $P(x)$ ont été calculées, le processus père envoie `SIGUSR2` à tous les fils, puis il attend leur terminaison et efface le fichier.

Par exemple :

```
> ./poly 3 toto 1 -2 3
2                       # P(1) = 1 × 1^0 − 2 × 1^1 + 3 × 1^3
9                       # P(2) = 1 × 2^0 − 2 × 2^1 + 3 × 2^2
22                      # P(3) = 1 × 3^0 − 2 × 3^1 + 3 × 3^2
```

Chaque calcul de $a_i x^i$ ($0 \leq i \leq n$) doit être réalisé avec la commande `expr` qui affiche le résultat sur la sortie standard (qui peut être redirigée vers un tube). Par exemple, le calcul de 2×3^2 doit être réalisé avec la commande `expr 2 * 3 * 3` (si vous utilisez cette commande interactivement avec le Shell, il faut neutraliser le caractère spécial « * » (« `expr 2 \* 3 \* 3` » ou « `expr 2 "*" 3 "*" 3` »).

Pour rédiger votre programme, il est impératif de respecter les contraintes suivantes :

— on supposera que les entiers sont codés sur 32 bits, donc compris entre -2^{31} et $2^{31}-1$ (environ 2 milliards), et on ne traitera pas les cas de débordement de ces valeurs ;
— vous ne devez utiliser que les primitives système (ou assimilées comme telles) ; vous pouvez toutefois utiliser les fonctions de bibliothèque pour les affichages et les manipulations de chaînes de caractères ou de mémoire ;
— pour des raisons d'efficacité, vous ne ferez pas d'appels redondants à des fonctions lentes (primitives système ou autres) ;
— vous vérifierez soigneusement les débordements de tableau ;
— lorsqu'une erreur est détectée, le programme doit s'arrêter aussitôt avec un code de retour indiquant l'erreur, sans attendre la terminaison de tous les processus fils en cours d'exécution : on suggère pour cela de détecter la terminaison prématurée d'un fils à l'aide d'un signal ;
— votre programme doit retourner un code de retour nul (`exit(0)`) si tout s'est déroulé sans erreur ou un code de retour non nul (`exit(1)`) si une erreur a été rencontrée ;
— si votre programme est appliqué avec un nombre d'arguments incorrect, il doit afficher le message : `"usage : poly k f a0 ... an"`.

— vous apporterez un soin particulier à la mise en forme de façon à rendre un code lisible et commenté à bon escient. Référez-vous au document « Conseils pour réussir vos TP et projets » mis à votre disposition sur Moodle et, si besoin, utilisez l'utilitaire `clang-format` avec la configuration donnée dans ce document ;
— votre programme doit compiler avec les options `-Wall -Wextra -Werror -pedantic` sur `gcc` version 9.4 minimum (la version disponible sur la machine `turing.u-strasbg.fr`). Alternativement, vous pouvez utiliser l'image Docker `pdagog/refc` (version de `gcc` 13.2) Les programmes qui ne compilent pas au moins sur `turing` avec ces spécifications **ne seront pas examinés**.

Un script de test est mis à votre disposition sur Moodle. Celui-ci exécute votre programme sur des jeux de tests qui serviront de base à l'évaluation de votre rendu. La commande suivante permet de lancer les tests : `sh test5.sh`.

Vous devrez rendre sur Moodle un *unique* fichier nommé `poly.c`.

Cet exercice est **individuel**. On rappelle que la copie ou le plagiat sont sévèrement sanctionnés.