Systèmes d'exploitation centralisés

mai 2021

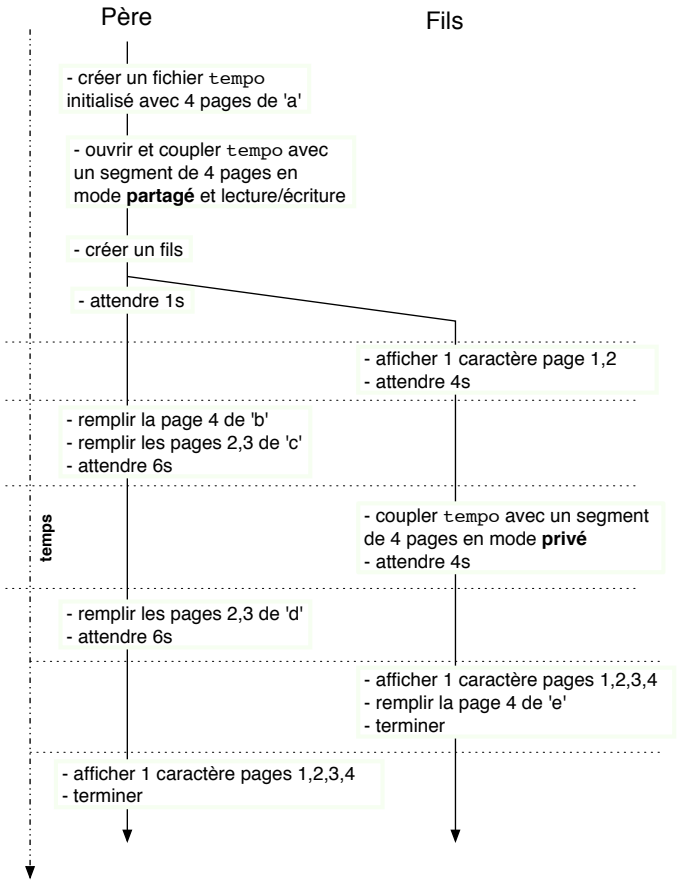
- *Documents autorisés* : 2 feuilles de notes manuscrites originales (pas de photocopie) format A4, recto-verso. Tous appareils électroniques interdits.
- *Durée* : 3 heures
- *Barème* : toutes les questions ont le même poids. **Les réponses non justifiées ne seront pas comptées.** La clarté et la qualité de la rédaction seront prises en compte.
- *Conseil* : il est préférable de **lire attentivement** l'énoncé avant de répondre.

1 Questions brèves

1. On considère un processus sous Linux, qui, successivement
   - crée un fichier `tempo` contenant quatre pages de caractères 'a',
   - ouvre ce fichier en lecture/écriture,
   - couple un segment de taille 4 pages à ce fichier, en mode partagé et lecture/écriture,
   - crée un processus fils,
   - attend 1 seconde,
   - remplit la page 4 du segment de caractères 'b',
   - remplit les pages 2 et 3 du segment de caractères 'c',
   - attend 6 secondes,
   - remplit les pages 2 et 3 du segment de caractères 'd',
   - attend 6 secondes,
   - affiche le premier caractère de chaque page du segment,
   - et termine.

   Le processus fils
   - affiche le premier caractère des pages 1 et 2 du segment,
   - attend 4 secondes,
   - (re)coupler le fichier `tempo` en mode privé, et lecture/écriture,
   - attend 4 secondes,
   - affiche le premier caractère de chaque page du segment,
   - remplit la quatrième page de caractères 'e',
   - et termine.

   Le schéma suivant présente une chronologie de l'exécution de ce programme.



   Indiquez, **en le justifiant**, quels seront (très vraisemblablement) les 8 derniers caractères affichés.

2. Quelle est la différence entre la notion de processus et celle de programme ?
3. Sous Unix, le nom (externe) d'un fichier est-il conservé dans le i-nœud correspondant à ce fichier ? Quelle est selon vous la justification de ce choix de conception ?
4. Donnez deux stratégies générales de partage d'une ressource, et précisez comment elles se traduisent en matière d'allocation mémoire.
5. Sur un système biprocesseur, est-il concevable qu'un processus donné puisse s'exécuter sur l'un puis l'autre des deux processeurs alternativement ? Justifiez votre réponse en expliquant le mécanisme de commutation de processus.
6. Expliquer pourquoi, sous Unix, il n'existe pas de primitive permettant de détruire un fichier. Expliquer comment le noyau gère néanmoins la destruction des fichiers ordinaires.
7. Comment rendre le code de gestion de fichiers indépendant des caractéristiques matérielles des différents périphériques ? Justifiez votre réponse.
8. La politique SJF (Shortest Job First) d'allocation du processeur fonctionne de la manière suivante ;
   - chaque processus prêt annonce sa prochaine durée d'utilisation du processeur. (Il est aussi courant que cette durée d'utilisation soit estimée à partir du comportement passé du processus)
   - le processeur est alloué au processus prêt ayant la plus courte durée d'utilisation.

   Cette politique est-elle équitable ? Justifiez votre réponse.
9. Quelles sont les stratégies et techniques employées à différents niveaux par le système d'exploitation pour réduire les temps d'accès aux données sur disque ?
10. Dans le domaine de l'allocation mémoire, montrer que (à défaut, expliquer pourquoi) la politique LRU ne peut présenter l'anomalie de Belady.

2 API Unix

Le programme suivant réalise une architecture ($MapReduce$) classiquement utilisée pour le traitement parallèle de grandes masses de données (vulgairement dénommées $BigData$).

Il s'agit ici de compter le nombre d'occurrence de chacune des voyelles dans un fichier de texte à analyser (de nom $fichierAAnalyser$), supposé très volumineux. Pour ce faire :

- Le fichier va être partitionné en fragments.
- Chaque fragment sera analysé séparément par un processus « compteur ». Les différents fragments pourront ainsi être analysés en parallèle, si l'on dispose de plusieurs processus compteurs.
- Le résultat de l'analyse des fragments par un processus compteur (nombre d'occurrence de chacune des voyelles dans les fragments traités) sera transmis à un processus « agrégateur », qui va effectuer le cumul des résultats des différents processus compteurs, avant d'afficher le résultat final.

L'architecture réalisée par ce programme est la suivante :

- le processus principal (« père ») crée les processus compteurs, puis le processus agrégateur ;
- le père ouvre ensuite le fichier à analyser, puis le lit par fragments, qu'il distribue régulièrement aux différents processus compteurs ;
- les processus compteurs analysent les fragments au fur et à mesure qu'ils leur sont transmis par le père, et, une fois la distribution achevée, transmettent leur résultat au processus agrégateur ;
- les données transitent entre processus via des tubes (pipes).

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>

#define TAILLE 4096
#define NBFILS 5

char * msgOut;
int nbOcc[6];   /* cumul du nb d'occurrences de chaque voyelle sur le texte analysé */
char voyelles[13] = "aeiouyAEIOUY";

void courant(int s) {
    int i;
    printf("%d␣:\n-----\n", getpid());
    for (i=0; i<6; i++)
        printf("%c␣:␣%d␣occurrences␣\n", voyelles[i] ,nbOcc[i]);
    return;
}

void analyser_occurrences(char* bloc, int lg) {
    int i,j;
    for (i = 0; i < 6; i++) nbOcc[i]=0;
    for(i=0; i < 6; i++)
        for(j=0; j < lg; j++)
            if( bloc[j] == voyelles[i] || bloc[j] == voyelles[i+6] ) nbOcc[i]++;
    return;
}

int main(void) {
    int p[NBFILS][2]; /* tubes père -> compteurs */
    int q[NBFILS][2]; /* tubes compteurs -> agrégateur */
    int i, j, pid, desc, nlus, necrits;
    int totalOcc[6]; /* utilisé par l'agrégateur, pour totaliser les résultats */
    char buf[TAILLE];
    char bufPere[TAILLE*NBFILS];
    struct sigaction mon_action;

    mon_action.sa_handler = courant;
    sigemptyset(&mon_action.sa_mask);
    mon_action.sa_flags = 0;
    sigaction(SIGINT, &mon_action, NULL);

    for (i = 0; i < NBFILS; i++) {
        pipe(p[i]);
        pipe(q[i]);
        pid = fork();
        switch (pid) {
        case 0: /* fils (compteur) */
            for (j=0; j<=i; j++) {
                close(p[j][1]);
                close(q[j][0]);
            }

            while((nlus = read (p[i][0], buf, TAILLE)) > 0) {
                analyser_occurrences(buf, nlus);
                write(q[i][1],nbOcc,sizeof(int)*6);
            }
            close(p[i][0]);
            close(q[i][1]);
            exit(0);
        default: /* père */
            printf("Fils␣%d␣(pid␣%d)␣cree\n",i+1,pid);
            close(p[i][0]);
            close(q[i][1]);
        }
    }

    pid = fork();
    if (pid == 0) { /* agrégateur */
        for (i=0; i<NBFILS; i++) close(p[i][1]);
        for (i=0; i<6; i++) totalOcc[i]=0;

        /* collecte */
```

```
77          do {
78              j=0;
79              for (i = 0; i < NBFILS; i++) {
80                  nlus=read (q[i][0], nbOcc,sizeof(int)*6);
81                  j+=nlus;
82                  if (nlus > 0) for (j=0; j<6; j++) totalOcc[j] += nbOcc[j];
83              }
84          }  while (j > 0);
85
86          for (i=0; i<NBFILS; i++) close(q[i][0]);
87          for (i=0; i<6; i++) printf("%c : %d occurrences \n", voyelles[i] ,totalOcc[i]);
88          exit(0);
89      } else { /* pere */
90          desc = open ("fichierAAnalyser", O_RDONLY);
91
92          while ((nlus = read (desc, bufPere, TAILLE*NBFILS)) > 0) {
93              /* distribution du bloc lu entre les différents compteurs */
94              for (i = 0; i < NBFILS; i++) {
95                  if ((nlus/TAILLE) > i ) necrits = TAILLE ; /* cas standard */
96                  else if ((nlus/TAILLE) == i ) necrits = nlus%TAILLE;/* cas où le dernier */
97                  else necrits = 0;          /* bloc lu a une taille inférieure à bufPere */
98                  write (p[i][1], bufPere+i*TAILLE, necrits);
99              }
100         }
101         for (j=0; j<NBFILS; j++) close(p[j][1]);
102         pause();
103         exit(0);
104     }
105     return 0;
106 }
```

11. Dessiner le graphe des processus et pipes créés.
12. Que se passe t-il à l'exécution si l'on supprime la ligne 74 (boucle `for (i=0;i<NBFILS;i++) close(p[i][1]);`) ? Justifiez votre réponse.
13. Que se passe t-il à l'exécution si l'on supprime la ligne 54 (instruction `close(q[j][0]);`) ? Justifiez votre réponse.
14. Le fait de supprimer la ligne 103 (appel à *pause*()) aurait-il un effet sur le résultat de l'exécution ? Si oui, lequel et pourquoi ? Si non, pourquoi ?
15. Le fait d'échanger les lignes 102 et 103 (boucle *for* et appel à *pause*()) aurait-il un effet sur le résultat de l'exécution ? Si oui, lequel et pourquoi ? Si non, pourquoi ?
16. La procédure `analyser_occurrences` (lignes 23-30) totalise les occurrences de voyelles rencontrées dans un bloc de données dont l'adresse est fournie en paramètre. Cette totalisation est faite dans une variable globale `nbOcc`. Dans ces conditions, est-il possible que la totalisation effectuée par l'un des fils (compteur) soit modifiée par un autre fils (compteur) s'exécutant au même moment (et donc que le résultat final soit incorrect) ? Justifiez votre réponse.

Le processus agrégateur recueille les résultats du comptage par une série de lectures bloquantes sur les différents tubes dédiés à la collecte (lignes 78 à 85).

17. Aurait-il été plus approprié/efficace d'utiliser un schéma basé sur la primitive *select*(..) pour réaliser ce recueil ? Justifiez votre réponse
18. Aurait-il été plus approprié/efficace d'utiliser une scrutation et des E/S non bloquantes (primitive $fcntl$(..)) pour réaliser ce recueil ? Justifiez votre réponse.

On suppose que la taille du fichier à analyser est suffisamment grande pour que les processus compteurs restent actifs durant plusieurs secondes (et suffisamment petite pour qu'il n'y ait pas de débordement des variables comptabilisant les occurrences...). On lance le programme, et l'on suppose alors que les processus compteurs ont pour pid respectifs 1000, 1001, 1002, 1003, et 1004.

19. Le signal *SIGINT* est transmis au moyen de la frappe de `ctrl-C` au clavier. Quel affichage pourra-t-on observer à la suite de cette frappe ? Il n'est pas nécessaire de donner le détail de l'affichage, mais il faut décrire précisément son contenu et donner des éléments suffisants sur son ordre.
20. Le signal *SIGINT* est maintenant transmis depuis un autre terminal au moyen de la commande `kill -INT 1000`. Quel affichage pourra-t-on alors observer ? Il n'est pas nécessaire de donner le détail de l'affichage, mais il faut décrire précisément son contenu et donner des éléments suffisants sur son ordre.