SY5 – Systèmes d'exploitation
Examen de 1re session – 10 janvier 2024
Durée : 3 heures

Aucun document autorisé excepté une feuille A4 manuscrite
Appareils électroniques éteints et rangés

Tous les programmes demandés doivent être écrits en C, de la manière la plus lisible possible, c'est-à-dire ***bien indentés*** *et* ***commentés*** *aux endroits où cela paraît nécessaire.*

En revanche, s'agissant d'une épreuve sur papier, il ne vous est pas demandé un code irréprochable ; en particulier il est inutile d'indiquer les inclusions nécessaires, et vous pouvez vous contenter d'une gestion minimale des erreurs (par exemple en utilisant la fonction `assert()`*, ou même simplement avec un commentaire* `/* erreur à gérer */`*).*

Exercice 1 : histoires de famille

On considère une commande « `zombies` » dont l'exécution provoque la situation suivante :

```
poulalho@lulu:SY5$ ./zombies & sleep 5 ; ps j
   PPID     PID    PGID     SID TTY      STAT   UID   TIME COMMAND
1952538 1952539 1952539 1952539 pts/20   Ss    8159   0:00 -bash
1952539 1963818 1963818 1952539 pts/20   S     8159   0:00 zombies
1963818 1963822 1963818 1952539 pts/20   S     8159   0:00 zombies
1963818 1963823 1963818 1952539 pts/20   S     8159   0:00 zombies
1963822 1963824 1963818 1952539 pts/20   Z     8159   0:00 [sleep] <defunct>
1963823 1963825 1963818 1952539 pts/20   Z     8159   0:00 [sleep] <defunct>
1952539 1963861 1963861 1952539 pts/20   R+    8159   0:00 ps j
poulalho@lulu:SY5$ sleep 5; ps j
   PPID     PID    PGID     SID TTY      STAT   UID   TIME COMMAND
1952538 1952539 1952539 1952539 pts/20   Ss    8159   0:00 -bash
1952539 1963818 1963818 1952539 pts/20   S     8159   0:00 zombies
1952539 1963932 1963932 1952539 pts/20   R+    8159   0:00 ps j
```

1. Dessiner la généalogie des processus créés par l'exécution de « `zombies` ».
2. Expliquer précisément l'état de chacun de ces processus au bout de 5 secondes (*i.e.* lors de la première exécution de « `ps j` »).
3. Que s'est-il passé durant les 5 secondes suivantes (*i.e.* entre les deux exécutions de « `ps j` ») ?
4. Écrire un programme `zombies.c` ayant exactement le comportement observé.

Exercice 2 : autour des tampons de la bibliothèque standard

On considère les deux programmes suivants :

```
int main() { /* pif.c */
    printf("pif ");
    sleep(1); printf("paf\n");
    sleep(1); printf("pouf");
    exit(0);
}
```

```
int main() { /* _pif.c */
    printf("pif ");
    sleep(1); printf("paf\n");
    sleep(1); printf("pouf");
    _exit(0);
}
```

On exécute les deux programmes via « `strace` », en redirigeant la sortie soit vers un terminal, soit vers un fichier ordinaire. Déterminer quelles sont les traces possibles parmi les traces présentées en dernière page ; donner un exemple de ligne de commande produisant chaque trace possible, et expliquer pourquoi les autres ne le sont pas.

Exercice 3 : copie d'arborescence

Dans cet exercice, il est interdit d'exécuter une autre commande que le programme à écrire (en particulier, pas de « `cp` »), et d'utiliser les bibliothèques `ftw` et `fts` ; l'utilisation de `stdio` doit se limiter à la gestion des erreurs et au formatage de chaîne de caractères.

1. Écrire une fonction `int my_cp(char *src, char *dst)` qui, si `src` est la référence d'un fichier ordinaire, en effectue une copie dans un fichier de référence `dst` ; ce fichier est créé si nécessaire, avec les mêmes droits que le fichier `src`, et écrasé s'il existe (sans changement de droits). La fonction `my_cp` doit pouvoir copier de manière efficace n'importe quel fichier, quelle que soit sa taille. Elle renvoie 0 en cas de succès, et -1 en cas d'échec.
2. Écrire un programme `my-cp-rec.c` tel que « `./my-cp-rec src dst` » effectue une copie de *toute l'arborescence* de racine `src` dans un nouveau répertoire `dst`, si `src` est la référence d'un répertoire et `dst` une référence valide mais non préexistante, et en dehors de l'arborescence de racine `src` (ce qu'on ne demande pas de tester). Les droits d'accès doivent être préservés.
   On supposera pour simplifier que l'arborescence de racine `src` est entièrement accessible et ne contient que des répertoires et des fichiers ordinaires.

Exercice 4 : descendance d'un processus

On rappelle que l'option « `--ppid` » de « `ps` » permet de filtrer les processus affichés en fonction de leur processus parent, et que l'option « `-opid=` » permet de limiter l'affichage aux identifiants des processus listés. Par exemple :

```
poulalho@lulu:SY5$ (sleep 10 & sleep 10) & (sleep 10 & sleep 10) &
      # lancement de 2 sous-shells en parallèle, chacun lançant 2 sleep en parallèle
[1] 1129102
[2] 1129103
poulalho@lulu:SY5$ ps --ppid $$ -opid=      # $$ désigne le pid du shell courant
1129102          # pid du 1er sous-shell
1129103          # pid du 2e sous-shell
1129114          # pid de ps
poulalho@lulu:SY5$ ps --ppid "1129102 1129103" -opid=
1129104          # pids des 4 sleep lancés par les 2 sous-shells
1129105
1129106
1129107
```

1. Écrire un programme `liste_fils.c` prenant exactement un argument, et utilisant « `ps` » pour lister, *sur une ligne*, les processus fils du processus dont l'identifiant est passé en argument. Votre programme ne doit exécuter aucune autre commande que « `ps` ». L'usage de la bibliothèque `stdio` doit être limité à la gestion des erreurs et à l'affichage final. On supposera que le nombre de processus est limité à 512 et que chaque pid tient sur (au plus) 7 caractères.
2. Modifier le programme précédent pour lister toute la descendance du processus, en affichant une génération par ligne.

Exercice 5 : des balles et des tubes

Écrire un programme qui simule le jeu suivant entre un (processus) père et ses `N` fils : le père met `M` balles en jeu ($M < N$) en distribuant aléatoirement une balle à certains de ses fils ; durant toute la durée du jeu, chaque porteur de balle cherche à s'en débarrasser en l'envoyant à un de ses frères qui a les mains libres ; au bout d'un temps aléatoire, le père siffle la fin du jeu, les porteurs de balles ont perdu et chaque fils affiche alors un message de joie ou de dépit.

La transmission des balles se fera par tube(s) anonyme(s), et le père jouera un rôle de meneur de jeu garant du respect des règles : toutes les requêtes de transmission de balle doivent passer par lui, et il transmet la balle au destinataire seulement si celui-ci n'en a pas encore (en prévenant l'envoyeur de la réussite de l'envoi) ; dans le cas contraire, il la retourne à l'envoyeur. Les fils ne pouvant pas savoir qui a déjà une balle, le choix du destinataire est aléatoire.

Préciser en particulier le nombre de tubes nécessaires, le format des messages, et le mécanisme permettant aux fils de détecter la fin du jeu.

Exercice 6 : un problème de synchronisation

1. Quels affichages le programme suivant peut-il produire ?

```
int main() {
  if (fork() == 0) printf("ping ");
  else if (fork() == 0) printf("pong ");
  else printf("pang ");
}
```

2. Proposer une modification minimale du programme permettant d'assurer que l'affichage soit toujours `"ping pong pang "` (sauf en cas d'erreur de `fork()`, naturellement).

3. Cette solution est-elle adaptable pour assurer un autre affichage (sans changer la généalogie, ni le message affiché par chaque processus) ?

4. Proposer une solution pour assurer l'affichage `"pang pong ping "`, sans changer l'ordre des `fork` et des `printf`, en synchronisant les processus à l'aide de tube(s) anonyme(s).

5. Proposer une autre solution de synchronisation reposant sur l'envoi de signaux. Expliquer en particulier quand le(s) gestionnaire(s) doi(ven)t être mis en place. S'il y a un risque de blocage, expliquer comment l'éviter.

Petit mémento

```
int open(const char *pathname, int flags, mode_t mode);
int mkdir((const char *pathname, mode_t mode);
int dup2(int oldfd, int newfd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
DIR *opendir(const char *name);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
struct dirent {    /* contient entre autres : */
    ino_t d_ino; char d_name[];
};
int stat(const char *pathname, struct stat *statbuf);
struct stat {   /* contient entre autres : */
    dev_t st_dev; ino_t st_ino; mode_t st_mode;
};
int execlp(const char *file, const char *arg, ... /* (char *) NULL */);
int execvp(const char *file, char *const argv[]);
int kill(pid_t pid, int sig);
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
struct sigaction {  /* contient entre autres : */
  void (*sa_handler)(int); sigset_t sa_mask; int sa_flags;
}
```

Exercice 2 : autour des tampons de la bibliothèque standard

1. a.
```
[...]
fstat(1, {st_mode=S_IFCHR|0620, [...]}) = 0
[...]
write(1, "pif ", 4)                     = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "paf\n", 4)                    = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

b.
```
[...]
fstat(1, {st_mode=S_IFCHR|0620, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

c.
```
[...]
fstat(1, {st_mode=S_IFCHR|0620, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\npouf", 12)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
```

d.
```
[...]
fstat(1, {st_mode=S_IFCHR|0620, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

e.
```
[...]
fstat(1, {st_mode=S_IFCHR|0620, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

2. a.
```
[...]
fstat(1, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
write(1, "pif ", 4)                     = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "paf\n", 4)                    = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

b.
```
[...]
fstat(1, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

c.
```
[...]
fstat(1, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\npouf", 12)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
```

d.
```
[...]
fstat(1, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(1, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

e.
```
[...]
fstat(1, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

3. a.
```
[...]
fstat(3, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
write(3, "pif ", 4)                     = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "paf\n", 4)                    = 4
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

b.
```
[...]
fstat(3, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "pouf", 4)                     = 4
exit_group(0)                           = ?
+++ exited with 0 +++
```

c.
```
[...]
fstat(3, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "pif paf\npouf", 12)           = 12
exit_group(0)                           = ?
+++ exited with 0 +++
```

d.
```
[...]
fstat(3, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
write(3, "pif paf\n", 8)                = 8
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

e.
```
[...]
fstat(3, {st_mode=S_IFREG|0644, [...]}) = 0
[...]
clock_nanosleep([...], tv_sec=1, [...]) = 0
clock_nanosleep([...], tv_sec=1, [...]) = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```