Examen du 20 octobre 2016 (1h)

RS : Réseaux et Systèmes

Deuxième année

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction. Lisez entièrement le sujet avant de commencer calmement.

Documents interdits, à l'exception d'une feuille A4 recto-verso manuscrite à rendre avec votre copie.

★ Questions de cours.(3pts)

▷ **Question 1:** Qu'est-ce qu'un appel système ? Est-ce que `printf` est un appel système ? (Sinon, qu'est-ce que c'est ?)

Réponse

Voir le chapitre 1 du cours. appel système = appel de l'interface (API) du noyau (souvent standardisée dans POSIX). C'est une manière pour un processus en espace utilisateur de demander au noyau d'exécuter une opération, souvent critique pour la sécurité ou l'isolation inter-processus. printf n'est pas un appel système, c'est un appel de la bibliothèque standard C (libc). L'appel système appelé (une fois la préparation terminée, notamment la gestion des chaines de formatage) quand on fait un printf est write(). (on en a parlé dans le chapitre 3, "différentes interfaces d'usage des fichiers")

Fin réponse

▷ **Question 2:** L'exécution de `gcc -Wall -o monprog monprog.c` produit l'erreur ci-dessous. S'agit-il d'une erreur de compilation ou d'édition de lien ? Expliquez le problème, et comment il pourrait être résolu.

Réponse

Voir le chapitre 4 du cours. C'est un problème à l'édition de lien (d'ailleurs on le voit dans la sortie, car (1) c'est ld qui rale ; (2) il rale à propos du fichier .o. Le fichier .o contient une référence vers une fonction acosl qui n'est pas satisfaite lors de l'édition de lien, car il faudrait se lier à un autre fichier objet (typiquement une bibliothèque) la fournissant. Concrètement, c'est libm (-lm, la bibliothèque mathématique) qu'on voudrait ici.

Fin réponse

```
/tmp/cc260JQF.o: In function ‘main’:
monprog.c:(.text+0x13): undefined reference to ‘acosl’
collect2: error: ld returned 1 exit status
```

★ Exercice 1: Utilisation de `fork`, `exec`, `wait`, `waitpid`, `pipe`, `dup`, `dup2` (6pts)

Donner le code C (sans utiliser la fonction `system`) correspondant à ce que fait un *shell* lorsqu'on tape les lignes de commandes suivantes. Afin d'obtenir un code lisible et court, vous êtes dispensés des tests d'erreur des appels systèmes.

▷ **Question 1:** `prog1 || prog2`

(Il faut donc : lancer un programme *prog1*, attendre sa fin, récupérer son code de retour, et si ce code de retour est différent de 0, lancer un programme *prog2* et attendre sa fin)

Réponse

(ça ressemble fortement au "multido" du TP1, ou à l'E2Q1 du TD1)

```
        int status;
        if (fork() == 0) {
                execlp("prog1", "prog1", NULL);
        }
        wait(&status);
        if (WEXITED(status) && (WEXITSTATUS(status) != 0))
        if (fork() == 0) {
                execlp("prog2", "prog2", NULL);
        }
        wait(NULL);
```

Fin réponse

▷ **Question 2:** `prog1 | prog2`

(Il faut donc : lancer deux programmes *prog1* et *prog2*, en liant la sortie standard de *prog1* à l'entrée de *prog2*)

Réponse

(c'est une question de cours, cf slide 98)

```
        int fd[2];
        pipe(fd);
        if (fork() == 0) {
                dup2(fd[1], 1);
                close(fd[0]);
                close(fd[1]); // juste pour éviter de "fuiter" un fd en trop à prog1
                execlp("prog1", "prog1", NULL);
        }
        if (fork() == 0) {
                dup2(fd[0], 0);
                close(fd[0]);
                close(fd[1]);
                execlp("prog2", "prog2", NULL);
        }
        wait(NULL);
        wait(NULL);
```

Fin réponse

Extraits de quelques pages de manuel

```
pid_t fork(void);
int execlp(const char *file, const char *arg, ...);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
WIFEXITED(status) -- returns true if the child terminated normally
WEXITSTATUS(status) -- returns the exit status of the child
int pipe(int pipefd[2]);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

★ Exercice 2: Schémas d'exécution et ordonnancement (5pts)

▷ **Question 1:** Discutez les différents affichages possibles de chacun des programmes ci-dessous (avec schémas).

Programme 1

```
int main() {
  int status;
  pid_t pid;
  printf("Hello\n");
  if (pid=fork()) {
    printf("%d\n",!pid);
    waitpid(-1, &status,0);
    if (WIFEXITED(status))
      printf("%d\n",WEXITSTATUS(status));
    printf("End\n");
  }
  exit(!pid);
}
```

Programme 2

```
pid_t plop() {
  fork();
  printf("plop\n");
  return fork();
}

int main(){
  if(plop())
    printf("yes\n");
}
```

Réponse

```
        programme 1:
        Hello, 0 (père), 1 (retcode du fils affiché par le père), End (père toujours)
        c'est la seule exécution possible

        programme 2:
        plop yes plop yes
        ou plop plop yes yes
        (on remonte 4 fois dans main, mais seulement deux fois avec une valeur de
        retour != 0
```

Fin réponse

★ **Exercice 3: Savoir reconnaître les schémas de synchronisation classiques et utiliser les sémaphores (6pts)**

Partie 1 :

L'infâme pirate Barbe-Noire et ses 3 non moins infâmes lieutenants (LeBorgne, Long Jean d'Argent et Rackham le Rose) fêtent avec force rhum la prise d'un magnifique galion espagnol. Chacun des 4 hommes se met à avoir le comportement suivant :

- RÉPÉTER à l'infini
  - Essayer d'entrer dans la cale pour admirer le trésor
  - Admirer le trésor dans la cale pendant T_a secondes
  - Sortir de la cale
  - Cuver son rhum pendant T_c secondes
- FIN RÉPÉTER

(les valeurs de T_a et T_c importent peu)

▷ **Question 1:** Les pirates étant plutôt solitaires, ils préfèrent être seuls pour admirer le trésor. Proposez une solution (écrivez l'algorithme `Pirate`) à base de sémaphore garantissant que seul un pirate pourra entrer dans la cale.

▷ **Question 2:** De quel schéma de synchronisation classique ce problème se rapproche-t-il ?

Réponse

```
        c'est de l'exclusion mutuelle

        Pirate:
    while true:
        P(cale)
        admirer
        V(cale)
        cuver
```

Fin réponse

Partie 2 :

La soirée avançant, les pirates relâchent les règles, et décident que :

- Barbe-Noire doit être seul quand il contemple le trésor :
  - Lorsque Barbe-Noire admire le trésor, si un de ses lieutenants essaye d'entrer dans la cale, il attend que Barbe-Noire en soit sorti avant d'entrer à son tour.
  - Si Barbe-Noire veut entrer dans la cale alors qu'un (ou plusieurs) de ses lieutenant(s) admire(nt) le trésor, Barbe-Noire attend qu'il n'y ait plus personne dans la cale avant d'entrer à son tour.
- Les lieutenants peuvent admirer le trésor ensemble :
  Ainsi si un lieutenant est en train d'admirer le trésor dans la cale et qu'un autre lieutenant essaye d'entrer dans la cale, cet autre lieutenant entre dans la cale sans attendre.
- Personne ne double Barbe-Noire :
  Si un (ou plusieurs) lieutenant(s) admire(nt) le trésor et que Barbe-Noire attend d'entrer dans la cale, les autres lieutenants qui auraient envie d'admirer le trésor doivent attendre que Barbe-Noire soit entré dans la cale, ait admiré le trésor et soit sorti de la cale avant d'entrer à leur tour.

▷ **Question 3:** De quel schéma de synchronisation ce problème se rapproche-t-il maintenant ?

▷ **Question 4:** La règle « *Personne ne double Barbe-Noire* » permet d'éviter un problème courant. Quel est le nom de ce problème ? Que pourrait-il se produire si cette règle n'était pas présente ?

▷ **Question 5:** Implémentez les processus `BarbeNoire` et `Lieutenant` respectant les règles ci-dessus grâce à des sémaphores (et une variable globale).

Réponse

```
1         lecteurs/rédacteur
2
3         famine: BarbeNoire pourrait ne jamais rentrer
4
5
6         Implem:
7         sem entree = 1; # protège le processus d'entrée
8         int lieutenants = 0; # nb lieutenants dans la cave
9         sem cave = 1; # protège la cave
10        sem modif_lieutenants = 1; # modifs concurrentes de la variable lieutenants
11
12        BarbeNoire:
13     while true:
14        P(entree)
15          P(cale)
16        V(entree)
17        admirer
18        V(cale)
19
20        Lieutenants:
21     while true:
22        P(entree)
23          P(modif_lieutenants) # pour éviter une condition de compétition avec des
24                               # lieutenants en cours de sortie
25          if lieutenants = 0
26            P(cale)
27          lieutemants++
28          V(modif_lieutenants)
29        V(entree)
30        admirer
31
32        P(modif_lieutenants) # pour éviter une condition de compétition
33          lieutenants--
34          if lieutenants == 0
35            V(cale)
36        V(modif_lieutenants)
```

Fin réponse