M1 Systèmes Contrôle de TD 12 novembre 2009

Durée 50 minutes

Ecrivez vos réponses directement sur cette feuille. Vos notes de TD/TP/cours sont autorisés.

QCM

Cochez les réponses correctes.
Pour chaque case cochée: 0.5 correct, -0.25 incorrect. Total = max(somme,0).

1. Un signal envoyé à un processus stoppé qui ne bloque pas ce signal sera délivré
   ❍ à son réveil ❍ s'il appelle sigpending
   ❍ jamais ❍ immédiatement
2. Processus appelant les primitives wait, sigsuspend, read se met en état
   ❍ stoppé ❍ endormi
   ❍ zombie ❍ prêt
3. Un processus zombie
   ❍ peut tuer son père ❍ peut seulement recevoir SIGCHLD
   ❍ devient orphelin quand son père se termine ❍ disparaît quand son père fait waitpid
4. Un processus peut attendre l'arrivée d'un signal en utilisant la fonction
   ❍ pause ❍ wait ❍ sigsuspend ❍ sigprocmask
5. Après l'exécution du code sigemptyset(&set); sigaddset(&set,SIGALRM); sigprocmask(SIG_BLOCK,&set,NULL), le processus bloque
   ❍ bloque tous les signaux sauf SIGALRM ❍ attend SIGALRM
   ❍ bloque seulement SIGALRM ❍ bloque SIGALRM
6. La commande shell pour afficher le nombre d'erreur de compilations dans le fichier test.c c'est
   ❍ gcc test.c | grep err | wc -l
   ❍ jamais
7. Si l'appel à open("test4",O_WRONLY|O_CREAT|O_EXCL,0640) retourne une valeur différente de -1, alors le fichier test4 est créé et on sait que
   ❍ n'existait pas avant ❍ tous peuvent le lire
   ❍ il est exécutable ❍ getuid() dans ce code retournera l'UID de son propriétaire
8. L'exécution de execlp("echo","-n","marron",NULL); printf("vert")
   ❍ affiche "vert" ❍ affiche "marronvert"
   ❍ affiche "marron" ❍ n'affiche rien
9. Pendant l'exécution du traitement d'un signal, ce même signal
   ❍ est ignoré ❍ ne peut pas être envoyé par d'autre processus
   ❍ peut interrompre le traitement ❍ est bloqué
10. Au cours de sa vie un processus peut changer de
    ❍ session ❍ terminal ❍ groupe de processus ❍ pid ❍ ppid
    ❍ uid ❍ est ignoré
11. Une session
    ❍ est attachée à un terminal ❍ a au plus un groupe de processus en avant-plan
    ❍ a au plus un groupe de processus en arrière-plan ❍ a toujours un leader
13. Après l'appel à dup2(open("test",O_RDWR),1)
    ❍ la sortie standard est redirigée vers le fichier test
    ❍ l'écriture dans test sont redirigées vers la sortie standard
    ❍ descripteur de fichier pour test est fermé ❍ le processus n'a plus de terminal
14. Un processus qui a fait pipe(t); close(t[0]); write(t[1],buf,strlen(buf))
    ❍ est bloqué pour toujours ❍ ne peut plus jamais lire sur le tube
    ❍ est bloqué jusqu'à ce que les strlen(buf) caractères sont écrits
    ❍ reçoit le signal SIGPIPE

Exercice 2
Combien de processus sont créés par cette fonction? Dessiner l'arborescence. Quelle est la taille du fichier créé? Et si on remplace 3 par n?

```
void forks(int n,char c){
  int fd = open("exo2.txt",
  O_WRONLY|O_CREAT|O_TRUNC,0600);
  for(;n>0;n--){
        fork();
        write(fd,&c,1);
  }
  close(fd);

}
```

Exercice 3 S**ynchronisations**

Nous avons vu plusieurs façons de synchroniser des processus: pour qu'un processus B s'exécute avant le processus A, B doit appeler une fonction bloquante, tandis que A s'exécute puis débloque B.

1. **Tubes** Quels sont les affichages produits à l'exécution de ce programme? (4pts)

```
  char buf[1];
  int tube[2], tube2[2];
  pipe(tube2);
  pipe(tube);

switch (fork ()){
  case -1: perror("fork"); exit(-1);
  case 0: {
        read(tube[0],buf,1);
        printf("ca\n");
        write(tube2[1],"1",1);
        exit(0);
  }
  default:
      printf("wow\n");
     write(tube[1],"1",1);
      read(tube2[0],buf,1);
      printf("compile\n");
      wait(NULL);
      return 0;
    }
}
```

On veut réécrire ce programme, sans utiliser de tubes, en assurant la même synchronisation des processus. Le nouveau programme devra produire les mêmes affichages.

Modifiez les parties en gris de ce code pour implémenter la même synchronisation avec

1. la fonction de mise en sommeil sleep (2pts)
2. la fonction de terminaison de fils wait (2pts)
3. les signaux, par exemple, SIGUSR1 (5pts)

Synchronisation avec sleep

```
switch (fork ()){
  case -1: perror("fork"); exit(-1);
  case 0: {

       printf("ca\n");

       exit(0);
  }
  default:
     printf("wow\n");

     printf("compile\n");
     wait(NULL);

     return 0;
    }
}
```

Synchronisation avec wait

```
switch (fork ()){
  case -1: perror("fork"); exit(-1);
  case 0: {

       printf("ca\n");

       exit(0);
  }
  default:
     printf("wow\n");

     printf("compile\n");
     wait(NULL);

     return 0;
    }
}
```

Synchronisation avec les signaux

```
switch (fork ()){
  case -1: perror("fork"); exit(-1);
  case 0: {

       printf("ca\n");

       exit(0);
  }
  default:
     printf("wow\n");

     printf("compile\n");
     wait(NULL);

     return 0;
    }
}
```