Question 1 : (16 pts)

Répondez aux questions à choix multiples en sélectionnant une ou plusieurs réponses. Les questions font référence aux systèmes d'exploitation de la famille UNIX et considèrent les options par défaut des appels système.

1) [1 pt] Lorsqu'un processus se termine en invoquant « exit », :
a) le système force la terminaison de tous ses processus fils.
b) le système force la terminaison de son thread principal uniquement.
c) son père est adopté par le processus init.
d) son état de terminaison est enregistré dans son bloc de contrôle de processus.
e) aucune de ces réponses.

2) [1 pt] Un processus père crée un processus fils en invoquant « fork » puis appelle la fonction « execvp ». Son processus fils va :
a) continuer à exécuter le code abandonné par son père.
b) se bloquer jusqu'à ce que son père se termine.
c) perdre son père (il va être adopté par le processus « init »).
d) bloquer l'exécution de son père jusqu'à ce qu'il se termine.
e) aucune de ces réponses.

3) [1 pt] Considérez le bout de code suivant :
« int i=0 ; if(fork()) {i=i+1;} fork() ; i=i+1; printf("i=%d \n", i); ». Les valeurs de i affichées par le processus principal PP et le premier fils F1 de PP sont :
a) PP : i = 2 et F1 : i = 2.
b) PP : i = 1 et F1 : i = 2.
c) PP : i = 2 et F1 : i = 1.
d) PP : i = 1 et F1 : i = 1.
e) aucune de ces réponses.

4) [1 pt] Lorsqu'un processus père crée un processus fils en utilisant « fork », les processus père et fils ont la même sortie standard STDOUT. Si le processus fils redirige STDOUT vers un fichier FICH (ce fichier est créé par le processus père avant l'appel à « fork »), :
a) la sortie standard du père est aussi redirigée vers FICH.
b) la sortie standard du père reste inchangée.
c) la sortie standard du père sera fermée.
d) le père va perdre l'accès au fichier FICH.
e) aucune de ces réponses.

5) [1 pt] Un processus P crée deux threads utilisateur puis réalise ce qui suit : «int i=100; while(i--); sleep(3); while(i++<100); exit(0);». Chaque thread incrémente une variable globale v puis se termine. **Chaque thread créé peut s'exécuter :**
a) pendant que P est en pause (*sleep(3)*) en concurrence avec l'autre thread.
b) pendant que P est en pause mais pas en concurrence avec l'autre thread.
c) avant / après la pause de P, avant « exit(0) », mais jamais en concurrence avec P ou l'autre thread.

d) après « exit(0) ».
e) aucune de ces réponses.

6) [1 pt] Considérez la commande « *ls* | *sort* > *data* ». **La sortie erreur STDERR du processus exécutant « ls » est :**
a) le fichier data.
b) le pipe.
c) l'écran.
d) le clavier.
e) aucune de ces réponses.

7) [2 pts] Le nombre de processus créés par l'instruction
« while(n>0) { fork() ; fork() ; n=n-1 ;} », où n est un entier initialisé à 1, est :
a) 2.
b) 3.
c) 4.
d) >4.
e) aucune de ces réponses.

8) [2 pts] On veut faire communiquer deux threads Posix d'un même processus via un tube anonyme (pipe). Le thread lecteur lit, caractère par caractère, du tube jusqu'à ce qu'il rencontre une fin de fichier. Le thread écrivain dépose dans le tube, caractère par caractère, le contenu d'un fichier.
Le tube anonyme doit être créé :
a) avant la création du premier thread.
b) après la création du premier thread et avant la création du second thread.
c) après la création du second thread.
d) dans chacun des deux threads.
e) aucune de ces réponses.
Le thread lecteur du pipe doit fermer le descripteur d'écriture du pipe, :
a) avant la première lecture du pipe.
b) entre le début et la fin du thread (peu importe l'endroit).
c) nulle part.
d) aucune de ces réponses.

9) [2 pts] Considérez les processus P1, P2, P3 et P4, et les sémaphores S1 et S2. Les processus sont lancés en concurrence. **Indiquez les ordres d'exécution des opérations atomiques a, b, c, d, e et f qui ne sont pas réalisables.**

Semaphore S1=2, S2=0 ;			
P1	P2	P3	P4
P(S1) ; a; b; V(S2) ;	P(S1) ; c; V(S2) ;	P(S2) ; P(S2) ; d; e; V(S2) ;	P(S2) ; f ; V(S1) ; V(S1) ;

a) a ; b ; c ; f ; d ; e ;
b) a ; c ; b ; d ; e ; f ;

c) a ; b ; c ; d ; e ; f ;
d) a ; b ; f ; c ; d ; e ;
e) aucune de ces réponses.

10) [1 pt] Considérez les processus P1, P2, P3 et P4 de la question précédente. Lesquels des processus P1, P2, P3 et P4 peuvent se retrouver interbloqués ? **Les processus interbloqués sont :**
a) P1, P2, P3 et P4.
b) P1 et P2.
c) P3 et P4.
d) aucun.
e) aucune de ces réponses.

Attention pour la sous question 11, vous devez compléter la réponse.

11) [2 pts] Considérez le pseudo-code suivant :

```
int lock =0;  // variable partagée par P1 et P2
```

Processus P1	Processus P2

{ if (lock==0) kill(P2, SIGSTOP);
lock=1
SC1();
lock = 0;
kill(P2,SIGCONT);
}
	{ if (lock==0) kill(P1, SIGSTOP);
	lock=1;
	SC2();
	lock = 0;
	kill(P1,SIGCONT);
	}

Si P1 et P2 sont exécutés en concurrence alors ils peuvent se retrouver, en même temps, :
a) dans SC1 et SC2, respectivement car P1 et P2 peuvent lire la même valeur 0 de lock. Ils réalisent respectivement la séquence d'instructions « kill(P2, SIGSTOP); lock=1; appel à SC1(); » et « kill(P1, SIGSTOP); lock=1; appel à SC2(); » avant de recevoir le signal SIGSTOP.
b) suspendus par SIGSTOP car P1 et P2 peuvent lire la même valeur 0 de lock. Ils réalisent respectivement la séquence d'instructions « kill(P2, SIGSTOP); lock=1; » et « kill(P1, SIGSTOP); lock=1; appel à SC2(); » puis recoivent et traitent le signal SIGSTOP.
c) aucune de ces réponses car …

Question 2 (5 pts) : Processus & redirection des E/S standards

1) [2.5 pts] Complétez le programme ci-dessous pour qu'il réalise le traitement suivant :

- Le processus principal crée deux tubes anonymes, crée un processus fils F1, ferme les descripteurs de tubes non utilisés puis se met en attente de la fin de son fils.
- F1 crée un processus fils F2, redirige son entrée standard vers le premier tube et sa sortie standard vers le second tube. Il ferme les descripteurs de tubes non utilisés puis se transforme en F1.exe (execlp("F1.exe", "F1.exe",NULL) ;).
- F2 redirige son entrée standard vers le second tube et sa sortie standard vers le premier tube. Il ferme les descripteurs de tubes non utilisés puis se transforme en F2.exe (execlp("F2.exe", "F2.exe",NULL) ;).

Attention : il n'est pas demandé de traiter les cas d'erreurs.

```
int main ()
{  /*1*/

  return 0;
}
```

2) [2.5 pts] Peut-on remplacer les deux tubes anonymes par des tubes nommés ? Justifiez votre réponse. Si oui, indiquez, sous forme de commentaires sur le code donné en 1, à quels niveaux devrait-on créer les tubes nommés, les ouvrir, fermer éventuellement leurs descripteurs et, enfin, si l'ordre d'ouvertures des tubes nommés est important. **Ne donnez pas de code.**

Réponses à 1 et 2 :

```
int main()
{  int p12[2], p21[2];
   pipe(p12);     // pour 2  création du premier tube nommé
   pipe(p21);     // pour 2  création du second tube nommé
   if (fork())
   { close(p12[0]); close(p12[1]); close(p21[0]); close(p21[1]); wait(NULL); }
   else  if(fork())
        {  // pour 2 :  ouverture des deux tubes nommés  premier en lecture et le
        second en écriture
          dup2(p12[0],0); dup2(p21[1],1);
          close(p12[0]); close(p12[1]); close(p21[0]); close(p21[1]);
        // pour 2 : fermeture des descripteurs retournés par l'ouverture des tubes
nommés
          execlp("F1.exe", "F1.exe",NULL) ;
        } else {  // pour 2  ouverture des deux tubes nommés  premier en écriture et
        le  second en lecture
           dup2(p21[0],0); dup2(p12[1],1);
          close(p12[0]); close(p12[1]); close(p21[0]); close(p21[1]);
```

```
// pour 2 : fermeture des descripteurs retournés par l'ouverture des tubes nommés

            execlp("F2.exe", "F2.exe",NULL) ;
          }
   return 0;
}
```

L'ordre des ouvertures des tubes doit être le même dans F1 et F2 car il y a une synchronisation sur l'ouverture. Si l'ordre n'est pas le même, il pourrait y avoir interblocage.