## Question 1 : (8 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]** Si un thread d'un processus se termine en invoquant « exit », :
a) le système force la terminaison du processus et de tous ses threads.
b) le système va terminer uniquement le thread qui a invoqué « exit ».
c) Les éventuels fils du processus sont adoptés par le processus init.
d) Le père du processus est adopté par le processus init.
e) aucune de ces réponses.

**2) [1 pt]** Un processus crée, dans l'ordre, un tube anonyme (int fd[2]; pipe(fd);) et 3 threads. Il attend ensuite la fin des 3 threads avant de se terminer.
Les threads ne ferment pas et ne dupliquent pas les descripteurs du tube. Le nombre d'écrivains (descripteurs en écriture) du tube créé est :
a) 1.
b) 2.
c) 3.
d) 4.
e) aucune de ces réponses.

**3) [1 pt]** Un processus crée, dans l'ordre, un tube anonyme (int fd[2]; pipe(fd);) et 3 processus fils. Il attend ensuite la fin des 3 fils avant de se terminer. Les processus fils ne ferment pas et ne dupliquent pas les descripteurs du tube. Le nombre d'écrivains (descripteurs en écriture) du tube créé est :
a) 1.
b) 2.
c) 3.
d) 4.
e) aucune de ces réponses.

**4) [1 pt]** Un processus réalise dans l'ordre ce qui suit : « int fd[2]; pipe(fd); », crée un thread, « dup2(fd[1],1); dup2(fd[0],0); close(fd[1]); close(fd[0]);», « char c; while(read(0,&c,1)>0 && c!= '#'); », attend la fin du thread, «write(1,"fin\n",5); » puis se termine. Le thread réalise ce qui suit : « write(1,"from thread#",12); » puis se termine. L'appel à la fonction write(1,…), :
a) par le processus, va écrire dans le tube.
b) par le processus, va afficher à l'écran.
c) par le thread, va écrire dans le tube ou afficher à l'écran.
d) par le thread, va écrire dans le tube.
e) aucune de ces réponses.

**5) [1 pt]** Dans la question précédente, si le processus remplace « char c; while(read(0,&c,1)>0 && c!= '#'); » par « char c; while(read(0,&c,1)>0); » alors :
a) Le processus finira par bloquer pour toujours sur « read(0,&c,1) ».
b) Le processus va bloquer pour toujours sur « write(1,"fin\n",5) ».
c) Le thread va bloquer sur « write(1,"from thread#",12) ».
d) Le processus va attendre indéfiniment la fin de son thread.
e) aucune de ces réponses.

**6) [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 sont réalisables.**

| Semaphore S1=0, S2=1 ; | | | |
|---|---|---|---|
| P1 | P2 | P3 | P4 |
| P(S1) ;<br>a;<br>b;<br>V(S2) ; | P(S1) ;<br>c;<br>V(S2) ; | P(S2) ;<br>d;<br>e;<br>V(S1) ; | P(S1) ;<br>f ;<br>V(S1) ; |

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

**7) [1 pt]** Un processus crée un fils processus via l'appel système *fork*. Le père et le fils ne se transforment pas (aucun appel système exec). Indiquez parmi les énoncés suivants ceux qui sont corrects :
a) Le fils partage avec son père les pages du segment de code qui sont présentes en mémoire physique.
b) Le père et le fils partagent les pages du segment de données qui sont présentes en mémoire physique tant qu'ils y accèdent en lecture.
c) Toute modification d'une variable globale (déclarée avant la fonction main) par le père sera visible par le fils.
d) Le père et le fils vont partager le pointeur de fichier (pointeur de lecture/écriture) de tout fichier ouvert avant le fork.
e) Aucune de ces réponses

**Question 2 (5 pts) : Processus & signaux**

Considérez le code C suivant :

```
int n=2, p=0;
int main()
{ int i;
  for(i=0; i<n; i++)
  {  if (fork() == 0) { n--; p=0 ;}
     else p++ ;
   }
   printf(" Processus %d fils de %d, n=%d, p=%d \n ", getpid(), getppid(),n,p) ;
   if ( p )  while(wait(NULL) >0);
   else { kill(getpid(),SIGUSR1); sleep(2);}
   exit (0);
}
```

1) **[2 pts]** Donnez l'arborescence des processus créés par ce code ainsi que les valeurs de n et p affichées par chacun des processus. Que représente la valeur de p (pour chaque processus) ?

   **PP crée deux fils F1 et F2 et affiche n=2, p=2**
   **F1 ne crée pas de fils et affiche n=1, p =0**
   **F2 ne crée pas de fils et affiche n=1, p=0.**
   **Pour chaque processus, p donne le nombre de fils qu'il a créé.**

2) **[1.5 pt]** Modifiez le code précédent pour que chaque processus père affiche à l'écran un message d'erreur, pour chaque fils qui se termine anormalement. Le message d'erreur est de la forme : " Fin anormale du fils <numéro du fils> du processus <numéro du père> ". (Utilisez la macro « WIFEXITED » pour tester une fin normale.)

```
void action (int sig) { printf(" signal SIGUSR1 : %d\n ", getpid()); } // pour 3
int n=2, p=0;
int main()
{ int i;  signal(SIGUSR1,action); // pour 3
  for(i=0; i<n; i++)
  {  if (fork() == 0) { n--; p=0 ;}
     else p++ ;
   }
      printf(" Processus %d fils de %d, n=%d, p=%d \n ", getpid(), getppid(), n, p) ;
     /*pour 2*/
     if ( p )  while(p=wait(&i) >0)  {
          if(!WIFEXITED(i))
          printf(" Fin anormale du fils  %d  du processus %d\n", p, getpid()); } ;
   else { kill(getpid(),SIGUSR1); sleep(2);}
   exit (0);
}
```

3) **[1.5 pt]** Modifiez le code obtenu en 2 pour que tous les processus créés, y compris le processus principal, captent le signal SIGUSR1. Le traitement à associer au signal SIGUSR1 est : « printf(" signal SIGUSR1 : %d\n ", getpid()) ; ».
   Cette modification a-t-elle un effet sur l'état terminaison (normale/anormale) des processus qui ont reçu et traité le signal SIGUSR1 ? Justifiez votre réponse.

**Attention** : **Donnez un seul code qui inclut toutes les modifications demandées. Vous devez par contre bien indiquer les modifications associées à chaque question.**

**Question 3 (3 pts) : Processus & redirection des E/S standards**

Considérez le code C suivant :

```
char*noms[3]= { “F1”, “F2”, “F3”} ;
void ecrire(char* nom) { int iter=0; while(iter<10)  {write(1,nom,2); iter++;}}
int main()
{ /*pour 2*/  int fd[2]; pipe(fd);
/*pour 1*/
   int i;
for(i=0; i<3;i++)
        if(fork()==0)
        { dup2(fd[1],1); close(fd[1]); close(fd[0]); ecrire(noms[i]); exit(0);}
close(fd[1]);
char c; while(read(fd[0],&c,1)>0) write(1,&c,1);
close(fd[0]);
exit(0);


}
```

1) **[1.5 pt]** Complétez le code précédent pour créer trois processus fils Fi, i=1,3. Chaque fils Fi appelle la fonction ecrire (ecrire( noms[i]) ) puis se termine (exit(0)). Après la création des fils, le processus père se termine.

2) **[1.5 pt]** Complétez le code obtenu en 1 pour que :
   - la sortie standard de chaque fils soit redirigé vers un tube anonyme créé par le processus père, et
   - le processus père récupère (caractère par caractère) et affiche à l’écran les données déposées par les fils dans le tube avant se terminer.

**Attention** : **Donnez un seul code qui inclut toutes les modifications demandées. Vous devez par contre bien indiquer les modifications associées à chaque question.**

## Question 4 (4 pts) : Synchronisation

Considérez le pseudo-code de la solution au problème des producteurs / consommateurs :

```
const int N = 10;
int tampon [N];
int ip=0,ic=0;
Semaphore libre=N, occupe=0, mutex=1;
```

| Producteur (int nump) | Consommateur (int numc) |
|---|---|
| { while(1)<br>{ **P(libre);**<br>**P(mutex);**<br>tampon[ip] = nump;<br>**ip = Modulo (ip +1,N);**<br>**V(mutex);**<br>**V(occupe);**<br>}<br>} | { int item;<br>while(1)<br>{ **P(occupe) ;**<br>**P(mutex);**<br>item = tampon[ic] ;<br>**ic= Modulo (ic+1, N);**<br>**V(mutex);**<br>**V(libre);**<br>}<br>} |

Supposez qu'il y a exactement 3 producteurs (numérotés de 0 à 2) et 3 consommateurs (numérotés 0 à 2). On veut que les producteurs produisent tour à tour (producteur 0, producteur 1, producteur 2, producteur 0, etc.) et que les consommateurs consomment aussi tour à tour (consommateur 0, consommateur 1, consommateur 2, consommateur 0, etc.).

1) **[2 pts]** Modifiez le pseudo-code précédent de manière à satisfaire les requis précédents (productions tour à tour et consommations tour à tour). Pour la synchronisation, vous devez vous limiter à l'utilisation de sémaphores.
   Peut-on éliminer le sémaphore mutex ? Justifiez votre réponse.

2) **[2 pts]** On veut maintenant que chaque producteur produise deux items à chaque tour. Modifiez le pseudo-code obtenu en 1 de manière à satisfaire ce requis supplémentaire. Pour la synchronisation, vous devez vous limiter à l'utilisation de sémaphores.

**Attention** : **Donnez un seul code qui inclut toutes les modifications demandées. Vous devez par contre bien indiquer les modifications associées à chaque question.**

```
const int N = 10;
int tampon [N];
int ip=0,ic=0;
Semaphore libre=N, occupe=0, mutex=1;
```

**Semaphore prod[3] ={1,0,0}, cons[3]={1,0,0}**

```
Producteur (int pid)
{       while(1)
        {       P(prod[pid]) ;
                P(libre);
                P(libre); // pour 2
                P(mutex);
                tampon[ip] = pid;
                ip = Modulo (ip +1,N);
                 tampon[ip] = pid; // pour 2
        ip = Modulo (ip +1,N); // pour 2
                V(mutex);
                V(occupe);
                V(occupe); // pour 2
                V(prod[Modulo (pid+1,3)]) ;
        }
}
```

```
Consommateur (int cid)
 {      int item;
         while(1)
        {       P(cons[cid]) ;
                P(occupe) ;
                P(mutex);
                item = tampon[ic] ;
                ic= Modulo (ic+1, N);
                V(mutex);
                V(libre);
                V(cons[Modulo (cid+1,3)]) ;
        }
 }
```

Le sémaphore mutex n'est plus nécessaire car il n'y a ni d'accès concurrent à ip par les producteurs ni d'accès concurrent à ic par les consommateurs.