Question 1 (5 pts) : Généralités

1.1. **[3 pts]** Considérez un système multiprogrammé de traitement par lots qui ordonnance les threads. Dans un tel système, un processus crée 2 threads noyau, se met en attente de la fin de ses threads, affiche la valeur d'une variable globale *glob* initialisée à *0*, puis se termine. Chaque thread créé exécute la fonction suivante :

```
int glob=0;
void *count(void *arg) {   int i,j ;
        for(i=0; i<1000; i++)
        {       for(j=0 ; j<100000; j++); // simule un traitement local
                glob=glob+1;
        }
        return NULL;
}
```

1.1.1. Supposez que le système est monoprocesseur. La valeur de *glob* affichée par le thread principal du processus correspond-elle toujours à la valeur attendue ? Si vous répondez non, donnez la plus petite valeur de *glob* qui pourrait être affichée à l'écran, même si elle est peu probable. Justifiez vos réponses.

1.1.2. Supposez maintenant que le même processus est exécuté dans un système multiprocesseur qui ordonnance les threads en temps partagé. La valeur de *glob* affichée par le thread principal du processus correspond-elle toujours à la valeur attendue ? Si vous répondez non, donnez la plus petite valeur de *glob* qui pourrait être affichée à l'écran, même si elle est peu probable. Justifiez vos réponses.

1.2. **[2 pts]** Considérez le programme suivant :

```
int Continuer = 1;
void action (int sig) { printf("Traitement du signal: %d\n",sig);
                        if (sig==SIGUSR2) Continuer =0;
}
int main ()
{       signal(SIGUSR1, action);
        signal(SIGUSR2, action);
        signal(SIGINT, action) ;
        while (Continuer)
        {       kill(getpid(),SIGUSR1);
                pause();
                kill(getpid(),SIGUSR2);
        }
        exit(0);
}
```

Est-ce que le processus exécutant le programme précédent peut se retrouver en pause pour toujours ? Si c'est le cas, est-il possible de forcer sa terminaison (terminaison anormale) en tapant « *control-c* » (l'équivalent du signal SIGINT) ? Justifiez vos réponses.

Question 2 (10 pts) : Processus & tubes de communication

Considérez le programme suivant :

```
const int n=3, m=2;
int main()
{     int i,j=0;
      for(i=0; i<n && j<m; i++) {
            if(fork()==0)
            {    i=0; j=j+1;}
      }
      printf(" j=%d\n",j);
      exit (0);
}
```

Supposez que tous les appels système ne retournent pas d'erreur.

2.1. [3 pts] Donnez l'arborescence des processus créés par ce programme. Donnez la valeur de *j* affichée par chacun des processus, y compris le processus principal. À quoi correspond-elle ?

2.2. [1 pt] Complétez le code précédent pour que les processus sans enfants se transforment pour exécuter le code exécutable « *feuille* » qui est supposé existant dans le répertoire courant et le *PATH*. Cette transformation est réalisée par la fonction « *execlp* ». *Le fichier exécutable* « *feuille* » n'a aucun paramètre.

2.3. [3 pts] Complétez le code (obtenu en 2.2) pour que :

a. **chaque processus parent récupère, via *wait,* le nombre total de processus fils créés par tous ses descendants**,
b. **chaque processus, y compris le processus principal, retourne, via *exit*, le nombre de processus fils créés par lui et tous ses descendants**, et
c. **le processus principal affiche à l'écran le nombre total de processus fils créés par lui et tous ses descendants juste avant de se terminer**.

2.4. [3 pts] Complétez le code (obtenu en 2.3) pour que :

a. **tous les processus sans enfants redirigent leurs sorties standards vers un pipe anonyme avant de se transformer**,
b. **le processus principal récupère et affiche à l'écran tous les messages déposés par les processus sans enfants dans le pipe anonyme, juste après la fin de tous ses fils**, et
c. **chaque processus ferme les descripteurs du tube de communication non utilisés**.

Attention : Donnez un seul code, pour les questions 2.2, 2.3 et 2.4.

Question 3 (5 pts) : Synchronisation

3.1. [3 pts] Un groupe de 3 amis se donnent rendez-vous au cinéma. Ils doivent s'attendre à l'entrée du cinéma. Ensuite, ils entrent ensemble et assistent à la projection d'un film. Le comportement de chaque **ami i, pour i=0 à 2,** est simulé par le pseudo-code suivant :

```
/*0*/ // déclarer et initialiser les sémaphores
Process ami (int i)
{
        Trajet_au_cinema(i);
        /* 1*/ // attente de tous les amis en utilisant les sémaphores
        Acces_au_cinema(i);
}
```

Utilisez les sémaphores (uniquement) pour synchroniser les processus *ami*.

3.2. [2 pts] Considérez les 3 processus *A, B et C* suivants, qui partagent 4 sémaphores (M, *SA*, *SB, SC) et un compteur entier n*. Supposez que les processus sont lancés en concurrence :

Semaphore M=1, SA=0, SB=0, SC=0; *int n=0;*		
Process A	*Process B*	*Process C*
a1 *P(M);* *n=(n+1)%3;* *if(n==0) {* *V(M); V(SB); V(SC);* *} else { V(M); P(SA); }* *a2;*	*P(M);* *n=(n+1)%3;* *if(n==0) {* *V(M); V(SA); V(SC);* *} else { V(M); P(SB); }* *b;*	*P(M);* *n=(n+1)%3;* *if(n==0) {* *V(M); V(SA); V(SB);* *} else { V(M); P(SC); }* *c;*

Donnez, sous forme d'un arbre, tous les ordres possibles d'exécution des instructions atomiques *a1, a2, b et c*.

Le corrigé

Question 1

1.1.

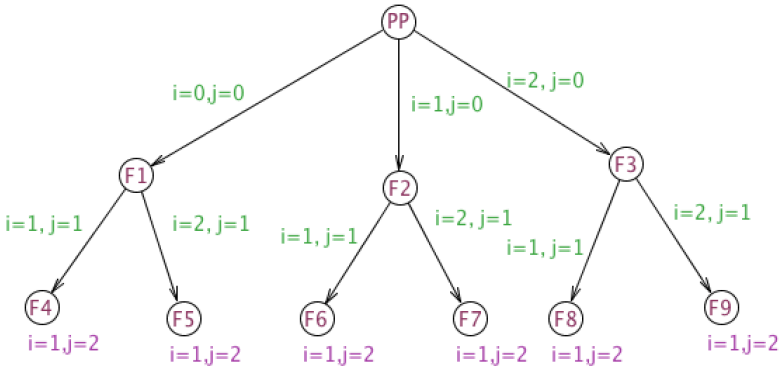
1.1.1. Oui, car dans un système de traitement par lots, lorsque le processeur est alloué à un thread, il va s'exécuter jusqu'à la fin car il ne réalise aucun appel système bloquant. La valeur de glob affichée est donc bien celle qui est attendue.

1.1.2. Non, car dans un système en temps partagé, un thread en exécution peut être suspendu même s'il ne se termine pas ou ne se bloque pas. Il peut être, par exemple, suspendu juste après avoir lu la valeur courante de glob. Si l'autre thread réalise plusieurs incrémentations de glob avant que le premier thread poursuive son exécution, le résultat serait incohérent. La plus petite valeur de glob affichée est 2. Elle est atteinte pour, par exemple, le scénario suivant :
1) le premier thread lit la valeur 0 de glob et est suspendu;
2) le second thread réalise 1000-1 incrémentations (glob=999) et est suspendu,
3) le premier thread complète sa première incrémentation (glob=1) et est suspendu,
4) le second thread lit la valeur 1 de glob et est suspendu,
5) le premier thread réalise toutes les incrémentations (glob = 1000)
6) le second thread réalise la dernière incrémentation (glob=2).

1.2. Oui, s'il traite le signal SIGUSR1 avant de se mettre en pause. Non, car control-c est traduit en un signal SIGINT et ce signal est capté par le processus. Ce dernier va poursuivre son exécution après le traitement du signal.

Question 2

2.1.



j indique le niveau du processus dans l'arbre créé par ce programme.

2.2., 2.3. et 2.4.

```
const int n=3, m=2;
int main()
{ int i,j=0;
  /*2.3*/  int f=0;
  /*2.4*/  int fd[2]; pipe(fd);

  for(i=0; i<n && j<m; i++) {
     if(fork()==0)
     {  i=0; j=j+1; }
  }
 printf(" j=%d\n",j);

 /*2.2*/ if(j==m)
          {    /*2.4*/ dup2(fd[1],1); close(fd[1]); close(fd[0]);
               /*2.2*/ execlp("./feuille","feuille",NULL);
           }
 /*2.4*/  close(fd[1]);
 /*2.4*/  if(j!=0) close(fd[0]);
 /*2.3*/  while(wait(&i)>0) { if(j<m-1) f=f+WEXITSTATUS(i) + 1; else f=f+1 ;}
 /*2.4*/  if(j==0) { char c; while(read(fd[0],&c,1)>0) write(1,&c,1); close(fd[0]);}
 /*2.3*/  if(j==0)  printf(" f=%d\n",f);
 /*2.3*/  exit(f);
}
```

Question 3

3.1.

```
Semaphore S[3]={0,0,0};
Process ami (int i)
{
        Trajet_au_cinema(i);
        /* attente de tous les amis*/
         V(S[(i+1)%3]); V(S[(i+2)%3]);  //pour signaler aux autres mon arrivée au cinéma
        P(S[i]); P(S[i]);   // pour éventuellement attendre l'arrivée des autres amis
        Acces_au_cinema(i);
}
```

3.2.

Le processus qui parvient à obtenir, en premier, le sémaphore M va incrémenter n modulo 3 (n=1), libérer le sémaphore M puis se mettre en attente de son sémaphore. Le processus suivant va incrémenter n modulo 3 (n=2), libérer le sémaphore M puis se mettre en attente de son sémaphore. Enfin, le dernier va incrémenter n modulo 3 (n=0), libérer le sémaphore M et les sémaphores des autres. Ensuite, les trois processus vont exécuter en concurrence chacun une action avant de se terminer. Cette question vous montre comment implémenter une barrière en utilisant des sémaphores. Cette barrière permet au processus A de réaliser l'action a1 avant les autres actions.

a1 -- a2 -- b -- c a1 -- a2 -- c – b a1 -- b – a2 – c
a1 -- b -- c – a2 a1 -- c – a2 – b a1 -- c -- b – a2