Tous documents et calculatrices autorisés – Durée : 3h00.
Le sujet comporte 9 page(s).

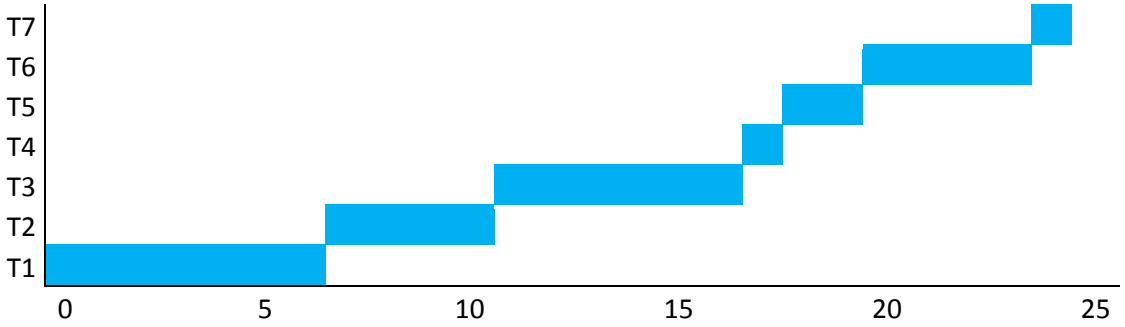
Exercice 1 (Ordonnancement) – 4 points

Soit le tableau suivant décrivant la date d'arrivée et la durée des processus T1 à T7 (en secondes) :

	T1	T2	T3	T4	T5	T6	T7
Durée	7	4	6	1	2	4	1
Date d'arrivée	0	2	3	3	4	5	6

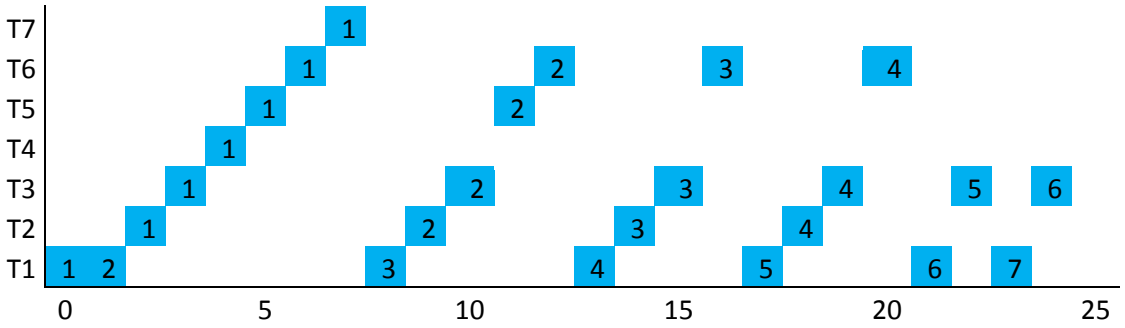
1) Montrer une exécution possible à l'aide d'un algorithme d'ordonnancement de type *FIFO*.

Correction :



2) Même question avec un algorithme d'ordonnancement de type *Round Robin*, le *quantum* de temps est de 1 seconde.

Correction :



Exercice 2 (Processus) – 4 points

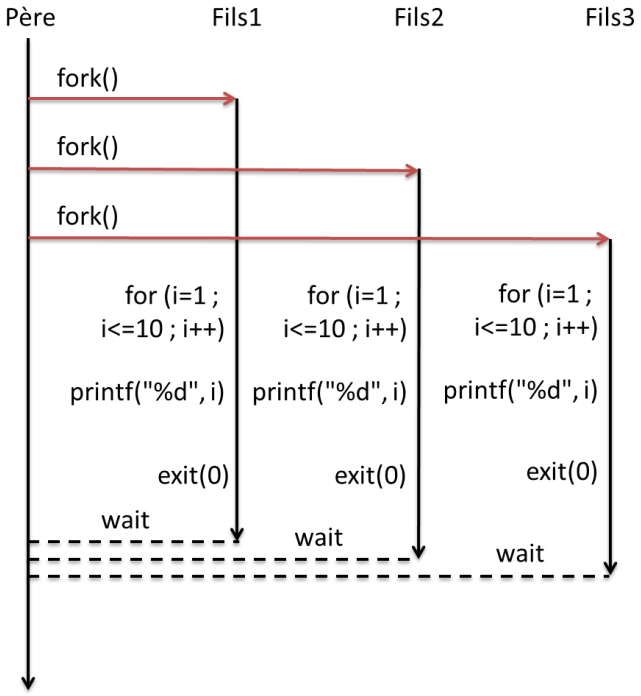
Soit un programme comportant un père et 3 fils.
Chacun des fils devra :

a) afficher un message de bienvenue, son propre PID, et le PID de son père ;
b) afficher une liste de nombres (1 à 10 par exemple) ;
c) se terminer en affichant un message.

Le père devra attendre la fin de l'exécution des fils et afficher un message pour dire que tout s'est bien déroulé.

1) Donner le schéma d'exécution d'un tel programme.

2) Annoter votre schéma avec les instructions systèmes permettant de réaliser les étapes clés du programme.



3) Donner le pseudo-code C (C ou algo) d'un tel programme.

```
#include <stdio.h>
#include <stdlib.h>

void fils() {
	int i;

	printf("[%d] FILS PID %d, PERE PID %d\n",getpid(), getpid(),
getppid());

	for (i = 1 ; i<=10 ; i++) {
		printf("[%d] %d\n",getpid(),i);
	}

	printf("[%d] FILS : au revoir !\n",getpid());
	exit(0);
}

int main() {

	int i;

	printf("[%d] bonjour, je suis le père !\n",getpid());

	for (i = 0 ; i<3 ; i++) {
		if (fork()==0) {
```

```
                fils();
            }
        }

        for (i=0 ; i<3 ; i++) {
            wait(NULL);
        }

        printf("[%d] PERE : au revoir !\n",getpid());

        return 0;
}
```

Exercice 3 (Sémaphores : synchronisation et exclusion mutuelle) – 6 points

Soit un programme exécutant 3 processus, chacun des trois processus P*i* comportent trois tâches identifiées T*i*1, T*i*2, T*i*3 (*i* = 1, 2, ou 3).

1) Rappelez les 2 primitives d'utilisation des sémaphores et précisez leurs rôles respectifs.

- P(S)

  S.cpt ← S.cpt – 1
  si S.cpt < 0 alors
    bloquer l'appelant
    le placer dans la file
  FSi

- V(S)

  S.cpt ← S.cpt + 1
  si S.cpt <= 0 alors
    sortir processus q de S.f
    reveiller(q)
  FSi

P(S) consiste à prendre un drapeau (donc attendre si celui-ci n'est pas levé).
V(S) consiste à rendre le drapeau (donc le lever).

2) Comment doit-on initialiser le compteur d'un sémaphore pour qu'il permette de gérer une exclusion mutuelle entre deux tâches de deux processus.
Vous illustrerez votre réponse :
   - en donnant un diagramme de précédence entre les tâches de ces processus,
   - en mentionnant explicitement les appels des primitives sur les pseudos-codes de chaque processus,
   - en précisant les valeurs d'initialisation des compteurs du sémaphore.

S.C0 = 1 si on veut une exclusion mutuelle.

P(S)	*P(S)*
T11	*T21*
V(S)	*V(S)*

3) Même question pour une synchronisation (ou dépendance temporelle) entre 2 tâches de votre choix.

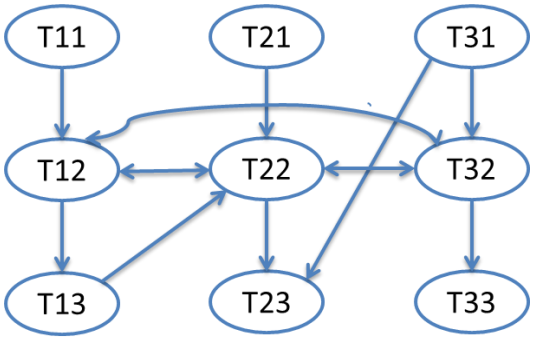
S.C0 = 0 si on veut une synchronisation.

Si début (T21) > fin (T11)

	P(S)
T11	*T21*
V(S)	

4) *Application :*
On considère les processus P1, P2, P3 :
- Chacun des processus Pi est constitué des trois tâches Ti1, Ti2, Ti3 (dans cet ordre) ;
- Ces tâches sont également régies par les relations de précédence (début(T22)>fin(T13) et début(T23) > fin(T31)) ;
- Les tâches Ti2 (i = 1, 2 ou 3) sont en exclusion mutuelle.

a) Dressez le diagramme temporel complet illustrant ces mises en concurrences.



b) Parmi les trois relations d'exclusion mutuelle, peut-on en omettre certaine(s) ? (laquelle/lesquelles ; justifiez ; est-ce judicieux de faire cela ?)

En règle générale, il ne vaut mieux pas supprimer de sémaphore, sous peine de changer le comportement du programme si on n'a pas fait un test exhaustif de toutes les possibilités.

c) Détaillez les pseudos-codes des trois processus ; vous préciserez également les valeurs d'initialisation des compteurs de ces sémaphores.

```
int main() {
S1.Init_sémaphore(0) ; // synchro début(T22)>fin(T13)
S2.Init_sémaphore(0) ; // synchro début(T23) > fin(T31)
S3.Init_sémaphore(1) ; // exclusion mutuelle T12, T22, T32

if (fork() == 0) {
   T11() ;
  P(S3) ;
   T12() ;
  V(S3) ;
   T13() ;
  V(S1) ;
  exit(0);
}

if (fork() == 0) {
   T21();
  P(S1);
  P(S3);
   T22();
  V(S3);
  P(S2);
   T23();
  exit(0);
}

if (fork()==0) {
   T31();
  V(S2);
  P(S3);
   T32();
  V(S3);
   T33();
  exit(0);
}
```

Exercice 4 (Tubes) – 6 points

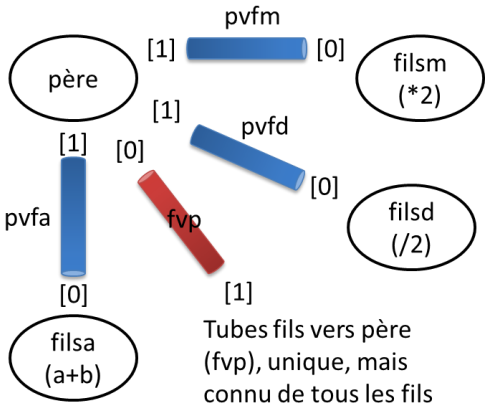
Soit un programme comportant 1 processus père et 3 processus fils.

Chaque processus fils à sa spécialisation :

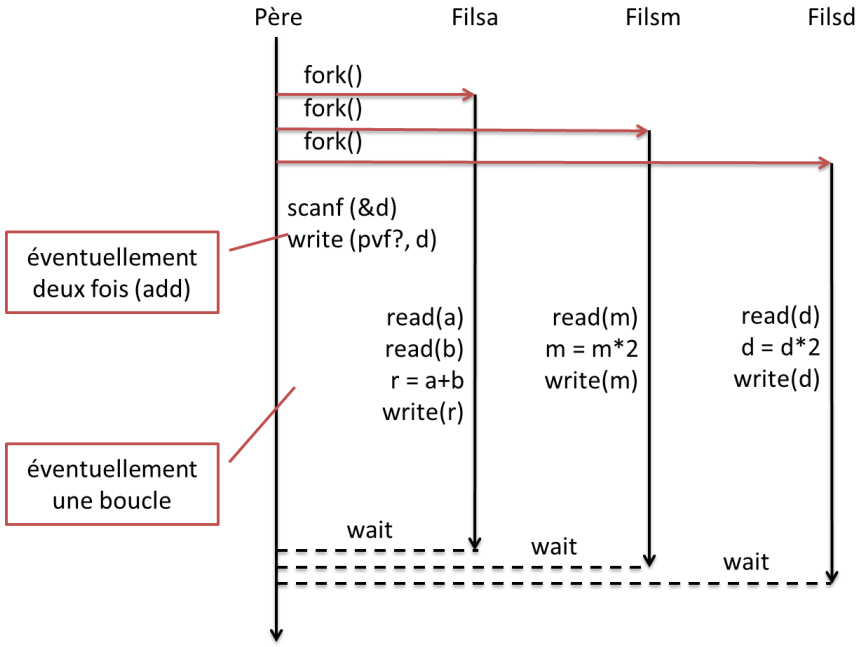
- Le premier est capable de multiplier par 2 un nombre reçu ;
- Le deuxième est capable de diviser par 2 un nombre reçu ;
- Le troisième est capable d'additionner deux nombres.

Le père enverra X et éventuellement Y à l'un des fils et attendra le résultat pour l'afficher.

1) Donner le schéma de communication faisant apparaître les processus et les tubes associés.



2) Donner le schéma d'exécution d'un tel programme.



3) Annoter votre schéma avec les instructions systèmes permettant de réaliser les étapes clés du programme.

   Cf. 2).

4) Donner le pseudo-code du père.

5) Donner le pseudo-code des fils.

```
#include <stdio.h>
#include <stdlib.h>

void filsa(int *pvfa, int *fvp) {
      int a, b, r, r1, r2;
      close(pvfa[1]);
      close(fvp[0]);

      do {
            r1 = read(pvfa[0], &a, sizeof(a));
            r2 = read(pvfa[0], &b, sizeof(b));

            if ((r1!=0) && (r2!=0)) {
                  r = a + b;
                  write(fvp[1], &r, sizeof(r));
            }
      } while ((r1!=0) && (r2!=0));

      close(pvfa[0]);
      close(fvp[1]);
      exit(0);
}

void filsm(int *pvfm, int *fvp) {
```

```
	int a, r, ro;
	close(pvfm[1]);
	close(fvp[0]);

	do {
		ro = read(pvfm[0], &a, sizeof(a));
		if (ro!=0) {
			r = a * 2;
			write(fvp[1], &r, sizeof(r));
		}
	} while (ro!=0);

	close(pvfm[0]);
	close(fvp[1]);
	exit(0);
}

void filsd(int *pvfd, int *fvp) {
	int a, r, ro;
	close(pvfd[1]);
	close(fvp[0]);

	do {
		ro = read(pvfd[0], &a, sizeof(a));
		if (ro!=0) {
			r = a / 2;
			write(fvp[1], &r, sizeof(r));
		}
	} while (ro!=0);

	close(pvfd[0]);
	close(fvp[1]);
	exit(0);
}


int main() {
	int pidfilsa, pidfilsm, pidfilsd;
	int pvfm[2], pvfd[2], pvfa[2], fvp[2];
	int c, a, b, r;

	pipe(pvfm);
	pipe(pvfd);
	pipe(pvfa);
	pipe(fvp);

	pidfilsa = fork();
	if (pidfilsa == 0 ){
		close(pvfm[0]);
		close(pvfm[1]);
		close(pvfd[0]);
		close(pvfd[1]);
		filsa(pvfa, fvp);
	}

	pidfilsm = fork();
	if (pidfilsm == 0) {
		close(pvfa[0]);
```

```
        close(pvfa[1]);
        close(pvfd[0]);
        close(pvfd[1]);
        filsm(pvfm, fvp);
}

pidfilsd = fork();
if (pidfilsd == 0) {
        close(pvfm[0]);
        close(pvfm[1]);
        close(pvfa[0]);
        close(pvfa[1]);
        filsd(pvfd, fvp);
}

close(fvp[1]);
close(pvfm[0]);
close(pvfa[0]);
close(pvfd[0]);

do {
        printf(" 1. additionner deux nombres\n");
        printf(" 2. multiplier par 2\n");
        printf(" 3. diviser par 2\n");
        printf(" 0. quitter\n");
        scanf("%d",&c);
        getchar();

        switch(c) {
        case 1 :
                printf("a = "); scanf("%d", &a); getchar();
                printf("b = "); scanf("%d", &b); getchar();
                write(pvfa[1], &a, sizeof(a));
                write(pvfa[1], &b, sizeof(b));
          break;
        case 2 :
                printf("a = "); scanf("%d", &a); getchar();
                write(pvfm[1], &a, sizeof(a));
          break;
        case 3 :
                printf("a = "); scanf("%d", &a); getchar();
                write(pvfd[1], &a, sizeof(a));
          break;
        }

        if (c!=0) {
                read(fvp[0], &r, sizeof(r));
                printf("PERE : le résultat est %d\n",r);
        }

} while (c != 0);

close(fvp[0]);
close(pvfm[1]);
close(pvfa[1]);
close(pvfd[1]);

return 0;
```

```
}
```