# IF-3-SYS : Examen du 10 juin 2024

**Q1.** Vrai ; Faux ; Vrai ; Vrai

**Q2.** `strace`

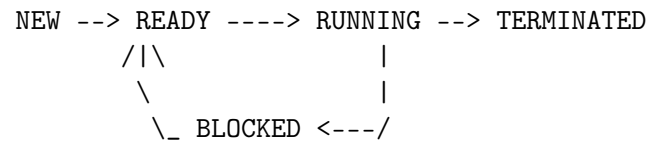
**Q3.**

```
int main() {
    if(!fork()) {
        for(int i=1;i<=50; i++)
            printf("%d\n",i);
        return 0; // important: child must exit
    }

    if (!fork()) {
        for(int i=51;i<=100; i++)
            printf("%d\n",i);
        return 0; // important: child must exit
    }

    wait(NULL);wait(NULL); // wait for both children
    printf("au revoir\n");
    return 0;
}
```

**Q4.** Pas de transition depuis running vers ready :




**Q5.**

```
time 0   2   4   6   8   9  10    15  17  19  21
 CPU | B | E | B | A | D | G |  A  | C | F | C |
```

**Q6.** Faux ; Faux ; Vrai ; Vrai

**Q7.** $2^{32} \div 2^{20} = 2^{12} = 4096$ $\Rightarrow$ Un processus peut avoir jusqu'à 4 kilo-pages de 1Mio chacune.

**Q8.** il est BLOQUÉ, aka SUSPENDU : l'ordonnanceur ne chercher pas à distinguer entre un défaut de page et une une requête IO volontaire.

**Q9.**

```
int *tab=malloc(N*sizeof(int));
for(int i=0; i<N; i++)
{
   *(tab+i)=0;
}
```

**Q10.** BF : X et Y dans B, Z dans C et T dans A.
FF : X, Y, Z dans A puis échec pour T

**Q11.** minimum 1 thread, par définition (les constantes sont inutiles ici)

**Q12.**
```
mkdir dossier1
cd dossier1
touch fichierA
mkdir dossier2
cd dossier2
touch fichierB
cd ..
cd ..
mkdir dossier3
cd dossier3
mkdir dossier4
cd dossier4
touch fichierC
```

**Q13.** Le volume contient 16 Gio = $16 \times 2^{30} = 2^{34}$ octets.
C'est à dire $2^{34-12} = 2^{22}$ blocs de 4 kio, donc autant de pointeurs dans la FAT.
$2^{22} \times$ 4 octets = $2^{24}$ octets = 16 Mio.

**Q14.** Exercice traité en TD. On y voit deux solutions : l'une avec un tableau de N sémaphores, l'autre avec un compteur de threads.

**Solution avec tableau** Partagé : Semaphore S[N]=0...0

barrière(i) : pour tout $j \neq i$ : V(S[ j ])
répéter N−1 fois : P(S[ i ])

voire même barrière(i) : pour tout j : V(S[ j ])
répéter N fois : P(S[ i ])

Remarques : à gauche, solution directement inspirée de la version à deux threads. À droite, une version «régularisée», plus élégante.

**Solution avec compteur** Partagé : Semaphore mutex=1, Semaphore Q=0, int count=0 ;

```
barrière(i) : P(mutex)
              count += 1
              si count == N :
                  répéter N fois : V(Q)
              V(mutex)
              P(Q)
```

Remarques : L'idée du P(Q) est de suspendre tous les threads sauf le dernier. On utilise le sémaphore Q comme une "queue" au sens du little book of semaphores §3.8.
Aussi, on pourrait utiliser le patron «turnstile» plutôt que répéter N fois le V(Q).

**Q15.**

**Solution avec tableau** (version de gauche) Le thread n° i consomme N−1 jetons dans S[ i ] chaque fois qu'il veut franchir la barrière. Or, ce sémaphore S[ i ] ne reçoit qu'un seul jeton par *autre* thread qui se présente. Il est donc nécessaire que tous les autres threads aient fait un tour de boucle pour pouvoir débloquer le thread n° i.
L'argument pour la version de droite est similaire.

**Solution avec compteur** Même si on réinitialise le compteur dans le if, la barrière n'est pas réutilisable : En effet, les $N-1$ premiers threads pourraient se faire préempter pile entre le V(mutex) et le P(Q), alors qu'ils ont déjà incrémenté 'count'. Q resterait donc à zéro !
Le dernier thread, lorsqu'il arrive, va faire N fois V(Q) et ainsi passer Q à +N. À priori, rien ne l'empêcherait de continuer sur sa lancée : après P(Q) on a Q=N−1, puis il refait un calcul, il repasse par la barrière, et ainsi de suite.

**Q16.** La solution «avec tableau» ne convient pas car elle demande autant de sémaphores qu'on a de threads. Du coup on repart ici de la solution «avec compteur» :

**Solution de wikipedia** Partagé : Sem mutex=1, Sem Q=0, int count=0, Sem parti=0 ;

```
barrière(i) : P(mutex)
        count += 1
        si count == N :
            count = 0
            répéter N−1 fois : V(Q)
            répéter N−1 fois : P(parti)
            V(mutex)
        sinon :
            V(mutex)
            P(Q)
            V(parti)
```

L'idée principale est d'empêcher notre dernier thread de faire deux tours d'affilée en le forçant à attendre que tous les autres soient repartis de la barrière. il doit donc attendre les N−1 V(parti). cf `https://fr.wikipedia.org/wiki/Barriere_de_synchronisation#Algorithme_multi-barri%C3%A8re`

**Solution du Little book of semaphores (§3.7.6)**

Partagé : Sem mutex=1, Sem Q1=0, sem Q2=0, int count=0 ;

```
barrière(i) : P(mutex)
        count += 1
        si count == N :
            répéter N fois : V(Q1)
        V(mutex)
        P(Q1)
        P(mutex)
        count -= 1
        si count == 0 :
            répéter N fois : V(Q2)
        V(mutex)
        P(Q2)
```

L'idée ici est de réinitialiser collectivement le compteur. Un thread très motivé qui voudrait faire deux tours d'affilée ne pourrait pas dépasser le P(Q2).
M'enfin, comme dit le little book of semaphores (§3.7.5) : *Unfortunately, this solution is typical of most non-trivial synchronization code : it is difficult to be sure that a solution is correct. Often there is a subtle way that a particular path through the program can cause an error.*