TD4 : Barrière de synchronisation.

sem_init (sem_t *s, 1, unsigned int value); -- initialisation

sem_post (sem_t *s); -- jeton ++

sem_wait (sem_t *s);

↳ nb jeton > 0, jeton --

↳ sinon attent, avant de faire jeton --

Question 1 :

	s1	s2	
p1()	0	1	
p2()	1	1	
	1	0	
p2()	0	0	↳ fin de la 10ème exécution.

b1() est exécuté après a1() et a2().

b2 ——————————

=> On va exécuter tous les a puis après seulement, on va exécuter les b

↳ barrière de synchronisation.

Question 2 :

```
#define N
sem_t s[N];
for(int i=0; i<N; i++){
    sem_init(&s[i], 1, 0);
}
p(int n){
    a(n);
    for(int i=0; i<N; i++){
        if(i != n){
            sem_post(&s[i]);
        }
        for(int i=0; i<N-1; i++){
            sem_wait(&s[n]);
        }
        b(n);
}
```

=> Il y a beaucoup de sémaphore.

Question 3:

* On a 1 sémaphore qui sert à bloquer tous les thread

* _____ de mutex.

Question 4:

```
struct barrier {
    sem_t m;
    sem_t s;
    int   c;
    int   n;
};

void barrier_init (struct barrier *b, int n)
{
    b->m =
    b->c = 0;
    b->n = n;
    sem_init (&b->m, 0, 1);
    sem_init (&b->s, 0, 0);
}
```

```
void barrier_synch (struct barrier *b) {
    sem_wait(&b->m);
    b->c++;
    if (b->c < b->n) {
        sem_post(&b->m);
        sem_wait(&b->s);
    }
    else {
        for(int i=0; i<N-1; i++) {
            sem_post(&b->s);
        }
        sem_post(&b->m);
    }
}
```