**Rendez-vous**

Créer un point de **synchronisation entre n processus**, c'est-à-dire que **chaque processus doit attendre les autres** pour continuer.

Deux variantes :

1. **Variante 1 (Broadcast)** : le **dernier arrivé réveille tous les autres**.
2. **Variante 2 (Chaînage)** : le **dernier arrivé réveille le précédent, qui réveille le précédent, etc.**, jusqu'au premier.

**🔧 Ressources synchronisation**

* mutex : exclusion mutuelle pour accéder à la variable partagée nb\_process.
* RDV : **un seul sémaphore** utilisé pour bloquer/réveiller les processus.
* nb\_process : compteur partagé du nombre de processus arrivés au rendez-vous.

**⚠️ Point clé pour éviter l’interblocage :**

* Toujours faire V(mutex) **avant** de faire un P(RDV) (car on **ne doit jamais bloquer avec le verrou pris** !)

**✅ CAS 1 — Le dernier réveille tous les autres (n - 1 réveils)**

**🔧 Initialisation**

mutex := 1

RDV := 0

nb\_process := 0

**📜 Algorithme pour chaque processus P\_i**

P(mutex);

nb\_process := nb\_process + 1;

Si nb\_process < n alors

V(mutex); // libérer mutex pour éviter l’interblocage

P(RDV); // attendre d’être réveillé

sinon

pour i de 1 à n-1 faire

V(RDV); // réveiller chaque autre processus

nb\_process := 0 ;

V(mutex);

✅ **Avantages** :

* Simple.
* Le dernier arrivé fait un broadcast.

**Commentaire détaillé ligne par ligne :**

P(mutex);

nb\_process := nb\_process + 1;

* On protège l’accès concurrent à la variable nb\_process.
* On compte combien de processus sont arrivés au rendez-vous.

Si nb\_process < n alors

V(mutex); // ✅ Libère mutex avant de bloquer : indispensable !

P(RDV); // 💤 Se bloque jusqu’à ce que le dernier arrive

* Les n-1 premiers processus se bloquent ici.
* Le mutex est bien libéré **avant** le P(RDV) pour éviter tout interblocage.

sinon

pour i de 1 à n-1 faire

V(RDV); // ✅ Le dernier réveille les autres (broadcast)

V(mutex); // 🔓 On libère le mutex après la boucle

* Le n-ième processus (le dernier) ne bloque pas.
* Il débloque tous les autres avec n-1 appels à V(RDV).
* Il libère mutex **après** car c’est lui qui termine la boucle critique pour que les prochain rendez-vous s’effectue correctement et il n’aura pas de chevauchement de rendez-vous.

**✅ CAS 2 — Réveil en chaîne (le dernier réveille l’avant-dernier, etc.)**

mutex := 1

RDV := 0

nb\_process := 0

P(mutex);

nb\_process := nb\_process + 1;

si nb\_process < n alors

V(mutex);

P(RDV); // attendre que le suivant me réveille

nb\_process := nb\_process - 1;

Si nb\_process > 0 alors

V(RDV);

Sinon

V(mutex);

**Programme rendez vous**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

// Sémaphores et variables partagées

sem\_t mutex;

sem\_t RDV;

int nb\_process = 0;

int N; // Nombre de threads

int mode; // Mode choisi : 1 = masse, 2 = chaîne

void \*processus(void \*arg) {

int id = \*((int \*)arg);

free(arg);

// Entrée en section critique partagée

sem\_wait(&mutex);

nb\_process++;

printf("[P%d] Arrivé au RDV. Compteur = %d\n", id, nb\_process);

if (nb\_process < N) {

sem\_post(&mutex); // Libérer avant d’attendre

printf("[P%d] Attend les autres (P sur RDV)\n", id);

sem\_wait(&RDV); // Se bloque

} else {

// Dernier processus

if (mode == 1) {

printf("[P%d] Dernier arrivé ➤ réveille tout le monde (mode 1)\n", id);

for (int i = 0; i < N - 1; i++) {

sem\_post(&RDV);

}

sem\_post(&mutex); // Libération finale

} else {

printf("[P%d] Dernier arrivé ➤ commence la chaîne (mode 2)\n", id);

sem\_post(&RDV); // Déclenche le réveil d’un seul

}

}

// Section critique simulée

printf("[P%d] Entre en section critique\n", id);

//sleep(1); // Simule une tâche

printf("[P%d] Quitte la section critique\n", id);

// Sortie pour le mode 2 (réveil en chaîne)

if (mode == 2) {

sem\_wait(&mutex);

nb\_process--;

if (nb\_process > 0) {

sem\_post(&RDV); // Réveil suivant

} else {

sem\_post(&mutex); // Dernier réveillé libère le mutex

}

}

pthread\_exit(NULL);

}

int main() {

printf("Entrez le nombre de processus (threads) : ");

scanf("%d", &N);

printf("Choisissez le mode de synchronisation :\n");

printf("1 ➤ Réveil en masse (dernier réveille tous)\n");

printf("2 ➤ Réveil en chaîne (dernier réveille le précédent, etc.)\n");

scanf("%d", &mode);

if (N <= 0 || (mode != 1 && mode != 2)) {

printf("Paramètres invalides. Fin du programme.\n");

return 1;

}

pthread\_t threads[N];

sem\_init(&mutex, 0, 1);

sem\_init(&RDV, 0, 0);

for (int i = 0; i < N; i++) {

int \*id = malloc(sizeof(int));

\*id = i + 1;

if (pthread\_create(&threads[i], NULL, processus, id) != 0) {

perror("Erreur création thread");

exit(1);

}

}

for (int i = 0; i < N; i++) {

pthread\_join(threads[i], NULL);

}

sem\_destroy(&mutex);

sem\_destroy(&RDV);

printf("\n✅ Tous les processus ont terminé.\n");

return 0;

}