Sémaphores (sem_t) What is that

🚀 Switching from `pthread_mutex_t` to `sem_t` and `fork` for the Bonus of `Philosophers` 🚀

In the **bonus** part of the **Philosophers** project, we must **replace `pthread_mutex_t` with semaphores (`sem_t`)** and **use processes (`fork`)** instead of threads.

🔹 1. Understanding Semaphores (`sem_t`)

Semaphores are **synchronization mechanisms**, similar to mutexes but with key differences:

✅ **A semaphore can be shared between multiple processes (`fork`)**
✅ **A semaphore has a counter**: It can be **decremented** (`sem_wait`) or **incremented** (`sem_post`)
❌ **Unlike mutexes, multiple philosophers can take a fork if the semaphore's value is > 1**

📌 **Using semaphores in the `Philosophers` project**:

- Each fork becomes a **semaphore**.
- **Message printing (`printf`)** should be protected by a **global semaphore**.
- A semaphore **replaces `pthread_mutex_t`** for controlling **`table->finish`**.

🔹 2. Initializing a Semaphore (`sem_t`)

📌 Using `sem_open` (for multiple processes)

```c
c

#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>

sem_t *semaphore;

semaphore = sem_open("/semaphore_name", O_CREAT, 0644, 1);
```

📌 **Explanation**:

- **`/semaphore_name`** → The name of the semaphore (important for process management).
- **`O_CREAT`** → Creates the semaphore if it does not exist.
- **`0644`** → Sets access permissions.
- **`1`** → Initial value of the semaphore.

⚠️ **Semaphore names must start with `/` and be unique!**
For example: **`"/forks_semaphore"`** for the forks.

🔹 3. Using a Semaphore (`sem_wait` and `sem_post`)

Once the semaphore is **created**, it can be used as follows:

Taking a fork (`sem_wait`)

A philosopher takes a fork:

```
c

sem_wait(semaphore);
```

→ **Decreases** the semaphore's value (blocks if the value is `0`).

Releasing a fork (`sem_post`)

A philosopher releases a fork:

```
c

sem_post(semaphore);
```

→ **Increases** the semaphore's value.

🔹 4. Removing a Semaphore

At the end of the program:

```
c

sem_close(semaphore);   // Closes access to the semaphore
sem_unlink("/semaphore_name"); // Deletes the semaphore from the system
```

📌 **Note**: `sem_unlink()` is needed **only for semaphores created with** `sem_open`.

🔹 5. Using `fork` (Processes Instead of Threads)

Instead of `pthread_create`, we use `fork()` to create **separate processes**.

Example of **creating a philosopher as a process**:

```
c

pid_t pid = fork();
if (pid == 0)  // Child process (philosopher)
{
    philosopher_routine();
    exit(0);  // Terminates the process
}
```

Inside `philosopher_routine()`, we replace mutexes with **semaphores**.

🔹 6. Full Example of Using Semaphores

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>

sem_t *forks;

void philosopher_routine(int id) {
    printf("Philosopher %d is thinking...\n", id);
    sem_wait(forks);
    printf("Philosopher %d took a fork.\n", id);
    usleep(500000);
    sem_post(forks);
    printf("Philosopher %d released the fork.\n", id);
    exit(0);
}

int main() {
    int i;
    pid_t pid;
    int nb_philos = 5;

    forks = sem_open("/forks_semaphore", O_CREAT, 0644, nb_philos / 2);

    for (i = 0; i < nb_philos; i++) {
        pid = fork();
        if (pid == 0) {
            philosopher_routine(i);
        }
    }

    for (i = 0; i < nb_philos; i++) {
        waitpid(-1, NULL, 0);
    }

    sem_close(forks);
    sem_unlink("/forks_semaphore");
    return 0;
}
```

📌 **Explanation**:

- **Creates a semaphore `forks` with `nb_philos / 2`** (since half of them can eat at the same time).
- **Each philosopher takes and releases a fork with `sem_wait` and `sem_post`.**
- **Processes are created using `fork()`.**
- **`waitpid(-1, NULL, 0)` ensures all processes finish before closing the semaphore.**

🔹 7. Key Differences Between Mutex and Semaphore

Feature	Mutex (`pthread_mutex_t`)	Semaphore (`sem_t`)
Shared across threads/processes	Only among threads of the same process	Can be shared across processes (`fork()`)
Counter	Binary (`0` or `1`)	Can have a value greater than `1`
Main functions	`pthread_mutex_lock` / `pthread_mutex_unlock`	`sem_wait` / `sem_post`

Feature	Mutex (`pthread_mutex_t`)	Semaphore (`sem_t`)
Initialization	`pthread_mutex_init`	`sem_open`

🔹 8. Summary

✅ Replace `pthread_mutex_t` with `sem_t` using `sem_open`
✅ Use `fork()` instead of `pthread_create()`
✅ Use `sem_wait` and `sem_post` to control access to forks and messages
✅ Delete semaphores with `sem_close` and `sem_unlink` at the end

👉 Let me know if you need help implementing semaphores in your `Philosophers` project! 🚀