fonction list

📌 Explanation of Each Function (In English)

These functions are essential for **process management and synchronization** in the **Philosophers** project **bonus**.

1️⃣ `fork()`

📌 Creates a new process (a child process).

```c
pid_t pid = fork();
if (pid == 0) {
    // This is the child process
    printf("I am the child process!\n");
    exit(0);
} else {
    // This is the parent process
    printf("I am the parent process!\n");
}
```

✅ **Returns**:

- **`0` in the child process**
- **The child's `PID` (Process ID) in the parent process**
- **`-1` if an error occurs**

🛠️ **Usage**: Used to create separate philosopher processes.

2️⃣ `kill()`

📌 Sends a signal to a process (can be used to terminate a process).

```c
kill(pid, SIGTERM); // Terminates process with given PID
```

✅ **Parameters**:

- `pid` → The process ID to send the signal to.
- `signal` → The signal type (e.g., `SIGTERM` to terminate the process).

🛠️ **Usage**: Used to **stop** philosopher processes when the simulation ends.

3️⃣ `exit()`

📌 Immediately terminates the calling process.

```c
if (error)
    exit(1); // Exit with failure code
exit(0); // Exit successfully
```

✅ **Returns**:

- **Does not return** (it stops the process).
- The **argument (`exit_code`)** can be used by the parent process.

🛠️ **Usage**: Used in the philosopher process to **end execution**.

4️⃣ `waitpid()`

📌 **Waits for a child process to finish.**

```c
int status;
waitpid(pid, &status, 0);
```

✅ **Parameters**:

- `pid` → The **process ID** to wait for (`-1` waits for **any child process**).
- `&status` → Stores the process **exit status**.
- `0` → **Default behavior** (blocks execution until the child exits).

✅ **Returns**:

- The **PID of the terminated child**.
- `-1` **on error**.

🛠️ **Usage**: Used to **wait** for all philosopher processes to finish before cleaning up.

5️⃣ `sem_open()`

📌 **Creates or opens a named semaphore (for inter-process synchronization).**

```c
sem_t *forks = sem_open("/forks", O_CREAT, 0644, 5);
```

✅ **Parameters**:

- `"/name"` → **Unique** semaphore name.
- `O_CREAT` → **Creates** the semaphore if it doesn't exist.
- `0644` → **Access permissions**.
- `5` → **Initial value** (number of available forks).

✅ **Returns**:

- A **pointer** to the semaphore.
- `SEM_FAILED` **on error**.

🛠️ **Usage**: Used to **control access to forks and messages** in the philosophers project.

6️⃣ `sem_close()`

📌 **Closes the semaphore (but does not remove it).**

```c
sem_close(forks);
```

✅ **Parameters**:

- `sem_t *sem` → The **semaphore** to close.

🛠️ **Usage**: Used at the **end of the program** to **release resources**.

7️⃣ `sem_post()`

📌 **Increments (unlocks) the semaphore (increases its value).**

```c
sem_post(forks); // Releases a fork (increases semaphore count)
```

✅ **Effect**:

- Increments the semaphore counter.
- Allows **another process to proceed** if it was blocked.

🛠️ **Usage**: Used when a philosopher **releases a fork**.

8️⃣ `sem_wait()`

📌 **Decrements (locks) the semaphore (blocks if value is `0`).**

```c
sem_wait(forks); // Takes a fork (decreases semaphore count)
```

✅ **Effect**:

- Decreases the semaphore counter.
- **Blocks** if the value is `0` (no forks available).

🛠️ **Usage**: Used when a philosopher **takes a fork**.

9️⃣ `sem_unlink()`

📌 **Deletes a named semaphore from the system.**

```c
sem_unlink("/forks");
```

✅ **Effect**:

- **Removes** the semaphore from the system.
- Only necessary for **semaphores created with** `sem_open()`.

🛠️ **Usage**: Used **at the very end** of the program to **fully remove semaphores**.

🔹 Summary Table

Function	Purpose
`fork()`	Creates a new process.
`kill()`	Sends a signal to a process (e.g., to terminate it).
`exit()`	Terminates the current process.
`waitpid()`	Waits for a child process to finish.
`sem_open()`	Creates or opens a named semaphore.
`sem_close()`	Closes a semaphore (but does not remove it).
`sem_post()`	Increments the semaphore (unlocks it).
`sem_wait()`	Decrements the semaphore (locks it, blocking if `0`).
`sem_unlink()`	Removes a named semaphore from the system.

🚀 Practical Example: Using `fork()` and Semaphores

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

sem_t *forks;

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

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

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

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

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

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

🚀 Key Takeaways

✅ `fork()` creates a **new process** for each philosopher.
✅ `sem_open()` initializes a semaphore **shared between processes**.
✅ `sem_wait()` **and** `sem_post()` are used to **control fork usage**.
✅ `sem_close()` **and** `sem_unlink()` clean up the semaphore at the end.

🎯 **This is how you replace `pthread_mutex_t` and threads with semaphores and processes in the `Philosophers` project bonus!** 🚀