Ecco una spiegazione dettagliata dei due tipi di scheduling per thread in POSIX:

---

1. `PTHREAD_SCOPE_SYSTEM` (System-Contention Scope - SCS)

- **Gestione**:
  - I thread sono schedulati direttamente dal **kernel del sistema operativo**.
  - Competono con **tutti i thread del sistema** per l'accesso alle CPU.
- **Caratteristiche**:
  - Ogni thread utente (pthread) è mappato a un **kernel thread** dedicato.
  - Il sistema operativo gestisce la priorità, l'assegnazione delle CPU e il context switching.
  - Supporta **parallelismo reale**: thread possono essere eseguiti contemporaneamente su core diversi.
- **Vantaggi**:
  - Migliore utilizzo delle CPU multicore.
  - Adatto a carichi di lavoro CPU-intensive o che richiedono bassa latenza.
- **Limitazioni**:
  - Overhead maggiore a causa del coinvolgimento del kernel.
  - Su Linux e macOS, è l'**unico scope supportato** (non è possibile usare `PTHREAD_SCOPE_PROCESS`).
- **Esempio nel codice**:

  ```
  pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
  ```

  Imposta lo scheduling a livello di sistema.

---

2. `PTHREAD_SCOPE_PROCESS` (Process-Contention Scope - PCS)

- **Gestione**:
  - I thread sono schedulati dalla **libreria threads a livello utente** (es. NPTL su Linux).
  - Competono **solo con altri thread dello stesso processo** per l'accesso alle CPU.
- **Caratteristiche**:
  - La libreria utente gestisce una o più **Lightweight Processes (LWP)** come interfaccia col kernel.

    - I thread utente sono mappati a un **pool limitato di LWP**, gestiti dalla libreria.
    - Il kernel vede solo gli LWP, non i singoli thread utente.

- **Vantaggi**:
    - Overhead ridotto per context switching (gestito in spazio utente).
    - Adatto a scenari con molti thread che non richiedono vero parallelismo.

- **Limitazioni**:
    - **Nessun vero parallelismo**: gli LWP sono limitati, quindi i thread utente condividono le stesse risorse.
    - Non supportato su Linux e macOS (solo `PTHREAD_SCOPE_SYSTEM` è disponibile).

- **Esempio teorico (non funzionante su Linux)**:

```
pthread_attr_setscope(&attr, PTHREAD_SCOPE_PROCESS);
```

  Tentativo fallito su sistemi che non supportano PCS.

Differenze Chiave

Caratteristica	`PTHREAD_SCOPE_SYSTEM` (SCS)	`PTHREAD_SCOPE_PROCESS` (PCS)
Gestione scheduling	Kernel del sistema operativo.	Libreria threads in spazio utente.
Competizione	Tutti i thread del sistema.	Solo thread dello stesso processo.
Parallelismo	Supportato (multi-core).	Limitato (LWP fissi).
Overhead	Maggiore (coinvolgimento kernel).	Minore (spazio utente).
Supporto su Linux/macOS	Sì.	No.

Implicazioni nel Codice Fornito

- **Linea critica**:

```
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
```

- Su Linux/macOS, questa riga funziona e imposta lo scheduling a livello di sistema.
- Se si prova a usare `PTHREAD_SCOPE_PROCESS`, fallisce (restituisce un errore).

- **Output del codice**:
  - Stampa `PTHREAD_SCOPE_SYSTEM`, poiché è l'unico scope disponibile.

Quando Usarli?

- **SCS**:
  - Applicazioni che richiedono massimo parallelismo (es. calcolo scientifico, rendering).
  - Ambienti dove il kernel ottimizza meglio le risorse (es. server multi-core).
- **PCS**:
  - Teoricamente utile in sistemi con molti thread "leggeri" (es. server web con connessioni concorrenti).
  - Non applicabile su Linux/macOS a causa della mancanza di supporto.

In sintesi: Su sistemi moderni come Linux e macOS, `PTHREAD_SCOPE_SYSTEM` è l'unica opzione praticabile, mentre `PTHREAD_SCOPE_PROCESS` rimane un concetto teorico o legato a sistemi obsoleti.