Vediamo prima di tutto le **differenze tra i tre tipi di scheduling** disponibili su Linux e menzionati nel codice (SCHED_OTHER, SCHED_RR e SCHED_FIFO), e poi la **differenza tra thread a livello kernel** e **thread a livello utente**.

---

1) Politiche di scheduling in Linux

SCHED_OTHER

- È la **politica di scheduling predefinita** in Linux per i processi "normali" (non real-time).
- Utilizza un meccanismo di **time-sharing**:
  - Ogni processo ha un certo **time slice** (quantum di tempo) e una priorità dinamica che il kernel aggiusta in base al comportamento del processo (se è CPU-bound, I/O-bound, ecc.).
  - Il kernel cerca di essere "equo" (fair) nel distribuire il tempo CPU tra i vari processi.
- Non garantisce scadenze real-time; l'obiettivo è la **reattività** e la **condivisione bilanciata** della CPU.

SCHED_RR (Round Robin)

- Fa parte delle **politiche real-time** di Linux.
- I processi (o thread) con SCHED_RR hanno un **livello di priorità** che prevale su qualunque processo con SCHED_OTHER.
- Tra i processi **con la stessa priorità** real-time, si usa un algoritmo Round Robin:
  - Ciascun processo ha un time slice (fisso) e, se allo scadere del quantum non ha finito, viene messo in coda e passa il turno al prossimo processo di stessa priorità.
- Se arriva (o diventa pronto) un processo con **priorità real-time più alta**, preempie immediatamente i processi a priorità più bassa.

SCHED_FIFO (First In First Out)

- Anch'essa politica **real-time**, con precedenza su SCHED_OTHER.
- Non c'è **time slice**: un processo in esecuzione con SCHED_FIFO **continua a girare** fino a quando:
  1. Non termina,
  2. Non cede volontariamente la CPU (ad esempio, fa una chiamata di sistema bloccante, o `sched_yield()`),
  3. Non viene preemptato da un processo con priorità **più alta** che diventa pronto.
- Se più processi condividono la **stessa** priorità FIFO, vengono eseguiti in ordine di arrivo (FIFO).
- Può dare luogo a **starvation** per i processi a priorità minore se un processo ad alta priorità non rilascia mai la CPU.

> **Nota**: i processi real-time (SCHED_RR, SCHED_FIFO) devono essere usati con attenzione perché **bloccano** o **preempiono** i processi "normali" SCHED_OTHER finché non rilasciano la CPU. È un meccanismo indispensabile per applicazioni real-time, ma può causare problemi se impostato in modo improprio.

---

2) Differenza tra thread a livello kernel e thread a livello utente

Thread a livello kernel (kernel-level threads)

- Ogni thread è **visibile al kernel**: il sistema operativo mantiene una struttura (Task Control Block) per ciascun thread.
- Il **kernel** è responsabile dello scheduling di ognuno di questi thread in modo indipendente.
- **Pro**:
  - Se un thread fa una chiamata di sistema bloccante (I/O, ad esempio), **non blocca** gli altri thread dello stesso processo: il kernel può continuare a schedulare gli altri thread su CPU.
  - Possono sfruttare più core in parallelo, perché il kernel vede i thread come entità separate.
- **Contro**:
  - Creare e gestire thread a livello kernel può avere un **overhead** maggiore (ogni thread ha una struttura di kernel associata).

Oggi, la maggior parte delle implementazioni di thread in Linux (NPTL - Native POSIX Thread Library) è **kernel-level**.

Thread a livello utente (user-level threads)

- I thread **non** sono visibili come entità separate dal kernel; il kernel vede un **unico** processo.
- L'**intera gestione** (creazione, scheduling, sincronizzazione) avviene in **user space**, tramite una libreria di threading.
- **Pro**:
  - Switch di thread e creazione di thread sono **molto più leggeri** perché non coinvolgono il kernel (basta cambiare contesto in user space).
- **Contro**:
  - Se un thread fa una chiamata di sistema bloccante, **blocca l'intero processo** (e di conseguenza tutti i thread al suo interno), perché dal punto di vista del kernel è un'unica entità.
  - Non si può sfruttare facilmente il multiprocessore: il kernel assegna la CPU al processo come un "blocco unico". L'eventuale parallelismo deve essere gestito con soluzioni particolari (M:N, etc.).

Riepilogo

1. **SCHED_OTHER**: scheduling “normale” time-sharing; è la politica di default, adatta alla maggior parte delle applicazioni non real-time.
2. **SCHED_RR**: real-time Round Robin con time slice; preempta SCHED_OTHER e si alterna con processi di pari priorità.
3. **SCHED_FIFO**: real-time FIFO senza time slice; un thread continua a girare finché non termina, cede la CPU, o viene preemptato da priorità più alta.

Thread a livello kernel: gestiti dal kernel (scheduling e creazione sono a carico del kernel), più overhead ma migliore gestione del blocco e del multiprocessore.
Thread a livello utente: gestiti in user space (più leggeri e veloci da creare/commutare) ma se uno blocca, blocca tutto il processo, e non si beneficia in modo trasparente di più CPU.