Sessions - 34

Threads in Linux (POSIX Threads - pthreads)

🔍 What are Threads?

- A **thread** is the **smallest unit of execution** within a process.
- Multiple threads in a process **share the same memory**, file descriptors, and other resources.
- Threads allow **concurrent execution**, improving performance on multicore systems.

🧠 Difference: Process vs Thread

Feature	Process	Thread
Memory	Separate memory	Shared memory with other threads
Creation	Slow (fork())	Fast (pthread_create())
Communication	Use IPC	Share memory (easier)
Overhead	High	Low

🔧 POSIX Threads (pthreads)

Linux supports **POSIX-compliant threads**, using the **pthread** library.

You must compile with:

gcc thread.c -o thread -lpthread

✅ Basic Thread Functions

Function	Purpose
pthread_create()	Create a new thread
pthread_exit()	Exit from a thread
pthread_join()	Wait for a thread to finish
pthread_detach()	Detach a thread (no join)
pthread_self()	Get thread ID

👷 Example: Create and Join Threads

```
#include <stdio.h>
#include <pthread.h>

void* myThreadFunction(void* arg) {
    printf("Thread running: %s\n", (char*)arg);
    pthread_exit(NULL);
}

int main() {
```

```
    pthread_t thread_id;
    char* message = "Hello from thread";

    // Create a thread
    pthread_create(&thread_id, NULL, myThreadFunction, message);

    // Wait for the thread to finish
    pthread_join(thread_id, NULL);

    printf("Main thread ends\n");
    return 0;
}
```

🔎 **Explanation:**

- pthread_t → Thread ID type
- pthread_create() → Spawns a new thread
- pthread_join() → Waits for the thread to complete
- myThreadFunction() → Function run by the thread

🛡️ **Thread Safety Tips**

- Use **mutexes** (pthread_mutex_t) to avoid race conditions.
- Avoid shared memory conflicts by **locking** critical sections.
- Use **condition variables** (pthread_cond_t) for signaling between threads.

🔁 **Example with Mutex (safe counter):**

```
#include <stdio.h>
#include <pthread.h>

int counter = 0;
pthread_mutex_t lock;

void* increment(void* arg) {
    for (int i = 0; i < 100000; i++) {
        pthread_mutex_lock(&lock);
        counter++;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;

    pthread_mutex_init(&lock, NULL);
```

```
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Final counter: %d\n", counter);

    pthread_mutex_destroy(&lock);
    return 0;
}
```

multi-threaded counter program using pthread and mutex in Linux:

✅ **Full Code Recap**

```
#include <stdio.h>
#include <pthread.h>

int counter = 0;
pthread_mutex_t lock;
```

```
void* increment(void* arg) {
    for (int i = 0; i < 100000; i++) {
        pthread_mutex_lock(&lock);
        counter++;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;

    pthread_mutex_init(&lock, NULL);

    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Final counter: %d\n", counter);
    pthread_mutex_destroy(&lock);
    return 0;
}
```

🔍 Code Explanation

🔹 Line 1–2: Include Header Files

```
#include <stdio.h>
#include <pthread.h>
```

- stdio.h for printf
- pthread.h for POSIX thread functions

🔹 Global Variables

```
int counter = 0;
pthread_mutex_t lock;
```

- counter: A global integer that two threads will increment.
- lock: A mutex (mutual exclusion) object used to prevent **race conditions**.

🔸 increment() – Thread Function

```
void* increment(void* arg) {
   for (int i = 0; i < 100000; i++) {
```

```
        pthread_mutex_lock(&lock);
        counter++;
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
```

- Runs a loop 100,000 times.
- Each iteration:
    - Locks the mutex before modifying counter to **prevent simultaneous access**.
    - Increments the global variable counter.
    - Unlocks the mutex.

👉 This ensures only **one thread modifies counter at a time**, avoiding data corruption.

🔸 **main() Function**

```
pthread_t t1, t2;
```

- Declare two thread IDs.

```
pthread_mutex_init(&lock, NULL);
```

- Initializes the mutex before it is used.

```
pthread_create(&t1, NULL, increment, NULL);
pthread_create(&t2, NULL, increment, NULL);
```

- Create two threads that run the increment() function.

pthread_join(t1, NULL);

pthread_join(t2, NULL);

- Wait for both threads to finish before continuing.

printf("Final counter: %d\n", counter);

- Displays the final value of counter.
- If everything works correctly, counter should be **200,000** (100,000 + 100,000).

pthread_mutex_destroy(&lock);

- Frees up the resources used by the mutex.

🧠 Why Use Mutex?

Without mutex locking, both threads might read and update the counter at the same time, leading to **incorrect results** like 180000, 190000, etc.

✅ Expected Output

Final counter: 200000