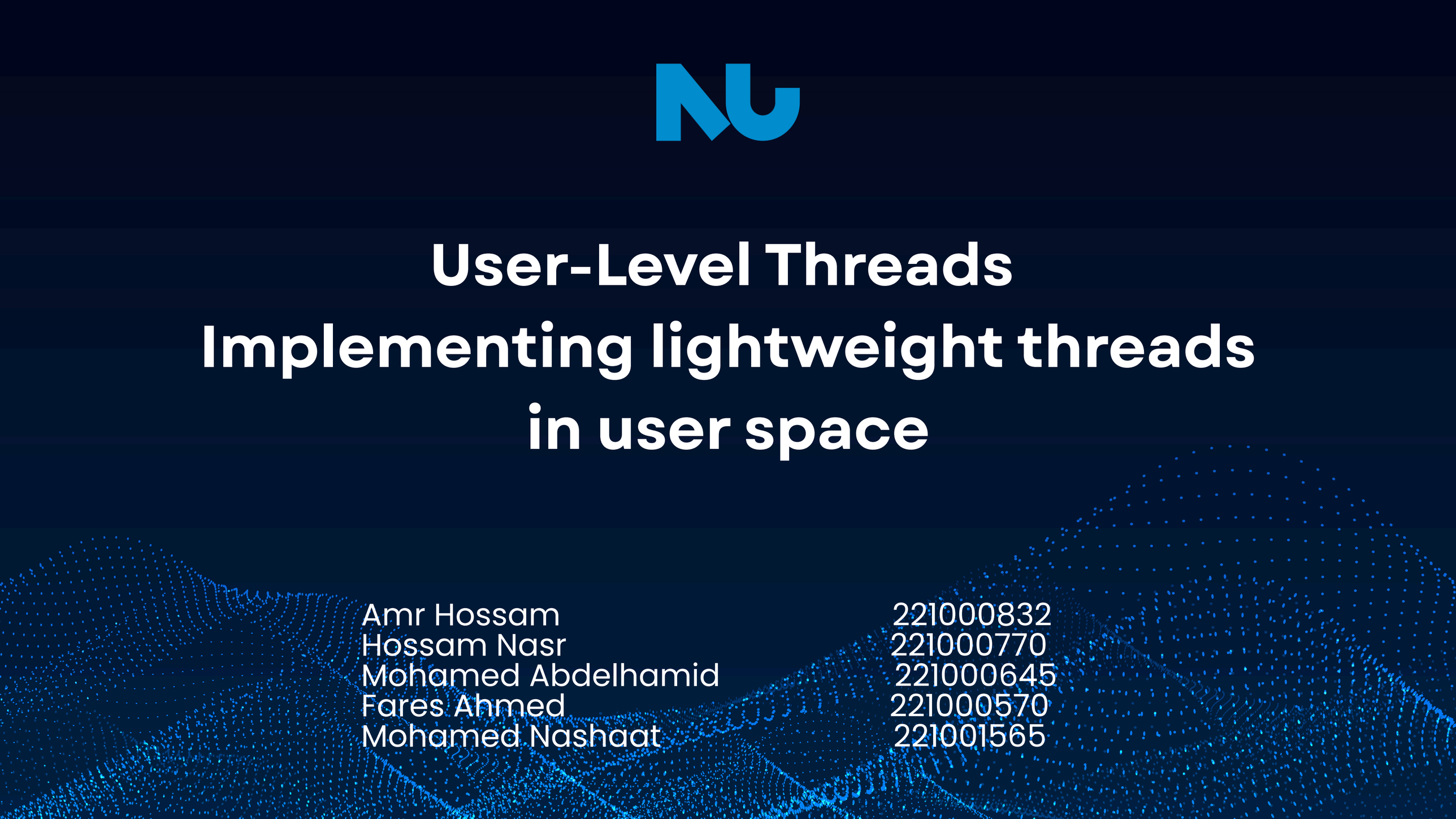

User-Level Threads
Implementing lightweight threads in user space

Amr Hossam	221000832
Hossam Nasr	221000770
Mohamed Abdelhamid	221000645
Fares Ahmed	221000570
Mohamed Nashaat	221001565

WHAT WE BUILT

User-level thread library running entirely in user space

Cooperative threading model with voluntary yielding

Custom scheduler for thread management

Context switching without kernel involvement

Thread synchronization primitives

WHY USER LEVEL THREADS

Advantages:

✅ Fast context switching (no system calls)

✅ Custom scheduling policies

✅ Portable across different kernels

✅ Low overhead thread creation

Trade-offs:

❌ No true parallelism (single CPU core)

❌ Blocking system calls block all threads

❌ No preemption without timer handling

SYSTEM ARCHITCTURE

1. uthread.h – Interface Definition

Thread data structures

API function declarations

Constants and enums

2. uthread.c - Core Implementation

Thread creation and management

Scheduler (round-robin)

Thread lifecycle functions

SYSTEM ARCHITCTURE

3. thread_switch.S - Context Switching

Assembly code for saving/restoring registers

Low-level CPU state management

4. uthread_test.c - Demonstration

Test cases showing thread functionality

THREADS IMPLEMENTATION
(uthreads.c)

Implements core thread management functions:

thread_create() — allocates stack, initializes context, marks RUNNABLE.

thread_yield() — saves current context and switches to next RUNNABLE thread.

thread_join() — waits for thread completion, manages cleanup.

thread_init() — initializes thread array and main context.

Maintains global variables for current thread and total threads.

Uses swtch() function to perform context switches.

EXPLORING HEADER
(uthreads.h)

Contains thread state constants and thread structure definition.

Declares thread functions and extern variables.

Defines STACK_SIZE for per-thread stacks.

Provides clear interface for user threads to use program.

TESTING THREADS
(uthread_test.c)

Test program demonstrating thread creation and scheduling.

Creates three threads running different functions (thread_a, thread_b, thread_c).

Threads print messages and yield control cooperatively.

Main thread yields to let others run and joins on all threads.

Shows how shared data can be used safely with threads.

CONTEXT SWITCHING
(thread_switch.S)

Assembly code implementing low-level context switching using RISC-V.

Saves registers of current thread and restores next thread’s registers.

Called from C code via swtch() function.

Critical for switching execution between user-level threads.

CONCLUSION

Successfully implemented lightweight user-level threads on Xv6.

Gained hands-on experience with context switching and scheduling.

Demonstrated cooperative multitasking without kernel support.

Learned how to manage per-thread stacks and execution context.

Enhanced understanding of OS concepts and system-level programming.

THANKS FOR YOUR ATTENTION!