**3 ЛАБОРАТОРНА РОБОТА №3**

**МЕХАНІЗМИ СИНХРОНІЗАЦІЇ ПОТОКІВ**

**3.1 Мета роботи**

Ознайомитись з основними механізмами синхронізації потоків, з поняттям критичної секції, порівняти м’ютекси, змінні стану умови та монітори, а також розробити програмний застосунок, який використовує м’ютекси для синхронізації потоків.

**3.2 Виконання роботи**

#ifndef MUTEX\_H

#define MUTEX\_H

#include "thread.h"

#include <QList>

typedef struct mutex {

uint id;

QList<thread\_t \*> wait;

bool is\_locked;

thread\_t \*owner;

}mutex\_t;

uint mutex\_create();

void mutex\_aсquire(uint mutex\_id);

void mutex\_release(uint mutex\_id);

#endif // MUTEX\_H

#include "mutex.h"

#include "controller.h"

uint next\_id = 0;

uint mutex\_create()

{

mutex\_t \*mutex = (mutex\_t \*)malloc(sizeof(mutex\_t));

mutex->id = ++next\_id;

mutex->is\_locked = 0;

mutex->owner = NULL;

add\_mutex(mutex);

return mutex->id;

}

void mutex\_aсquire(uint mutex\_id)

{

mutex\* m = get\_mutex\_by\_id(mutex\_id);

thread\* t = get\_current();

if (!m->is\_locked){

m->is\_locked = 1;

m->owner = t;

} else {

m->wait.append(t);

t->state = SLEEP;

shedule();

}

}

void mutex\_release(uint mutex\_id)

{

mutex\_t \*mutex = get\_mutex\_by\_id(mutex\_id);

if(mutex->wait.size() == 0) {

mutex->is\_locked = false;

return;

}

thread \*thrd = mutex->wait.takeLast();

thrd->state = READY;

add\_thread(thrd);

}

**3.3 Висновки**

У ході виконання даної лабораторної роботи ознайомились з основними механізмами синхронізації потоків, з поняттям критичної секції, порівняли м’ютекси, змінні стану умови та монітори, а також розробили програмний застосунок, який використовує м’ютекси для синхронізації потоків.