**1 ЛАБОРАТОРНАЯ РОБОТА №1**

**ЗАГАЛЬНА ХАРАКТЕРИСТИКА USER LEVEL THREAD (ULT) ТА ОСНОВНІ ВІДМІННОСТІ KERNEL LEVEL THREAD (KLT) І ULT**

* 1. **Мета роботи**

Ознайомитися з поняттям потоку виконання, зі способом реалізації потоків у просторі користувача і основні відмінності з потоками виконання на рівні ядра.

* 1. **Виконання роботи**

#ifndef THREAD\_H

#define THREAD\_H

#define STACKSIZE 16384

#include <ucontext.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

typedef void\* (\*run\_func)(void\* ) ;

enum State\_Thread { RUN = 0, READY, FINESHED, SLEEP, MAIN };

typedef struct thread {

State\_Thread state;

u\_int32\_t id;

ucontext\_t \*context;

run\_func func;

stack\_t stack;

void \*data;

void \*res;

int wake\_time;

} thread\_t;

u\_int32\_t create\_thread(run\_func func, void\* data);

void thread\_worker();

void exit\_thread(thread\_t \*ptr);

void run\_thread(thread\_t \*ptr);

void sleep\_thread(int seconds);

#endif // THREAD\_H

#include "thread.h"

#include "controller.h"

#include <QDebug>

u\_int32\_t \_id = 0;

u\_int32\_t create\_thread(run\_func func, void\* data) {

//timer\_lock();

thread\_t \*ptr\_thread = (thread\_t \*)malloc(sizeof(thread\_t));

ptr\_thread->state = READY;

ptr\_thread->id = \_id ++;

ptr\_thread->func = func;

ptr\_thread->data = data;

ucontext\_t \*ptr\_cntx = (ucontext\_t \*) malloc(sizeof(ucontext\_t));

getcontext(ptr\_cntx);

stack\_t tmp\_stack;

memset(&tmp\_stack, 0, sizeof(stack\_t));

tmp\_stack.ss\_sp = malloc(STACKSIZE);

tmp\_stack.ss\_size = STACKSIZE;

ptr\_thread->stack = tmp\_stack;

ptr\_cntx->uc\_stack = ptr\_thread->stack;

ptr\_thread->context = ptr\_cntx;

makecontext(ptr\_thread->context, &thread\_worker, 0);

add\_thread(ptr\_thread);

//timer\_unlock();

return ptr\_thread->id;

}

void thread\_worker() {

thread\_t \*ptr = get\_current();

ptr->res = ptr->func(ptr->data);

exit\_thread(ptr);

}

void exit\_thread(thread\_t \*ptr)

{

ptr->state = FINESHED;

shedule();

}

void run\_thread(thread\_t \*ptr)

{

ptr->state = RUN;

thread\_t \*prev = get\_current();

set\_current(ptr);

if(prev->state == FINESHED) {

free(prev->stack.ss\_sp);

free(prev->context);

free(prev);

setcontext(ptr->context);

} else {

swapcontext(prev->context, ptr->context);

}

}

void sleep\_thread(u\_int32\_t seconds)

{

thread\_t \*ptr = get\_current();

ptr->state = SLEEP;

ptr->wake\_time = time(NULL) + seconds;

add\_sleep(ptr);

}

* 1. **Висновки**

У ході виконання даної лабораторної роботи було ознайомлено з поняттям потоку виконання, зі способом реалізації потоків у просторі користувача і основними відмінностями з потоками виконання на рівні ядра